A

Optimization Models for Three On-chip Network Problems

NILAY VAISH, University of Wisconsin-Madison
MICHAEL C. FERRIS, University of Wisconsin-Madison
DAVID A. WOOD, University of Wisconsin-Madison

We model three on-chip network design problems—memory controller placement, resource allocation in heterogeneous on-chip networks, and their combination—as mathematical optimization problems. We model the first two problems as mixed integer linear programs. We model the third problem as a mixed integer non-linear program, which we then linearize exactly. Sophisticated optimization algorithms enable solutions to be obtained much more efficiently. Detailed simulations using synthetic traffic and benchmark applications validate that our designs provide better performance than previously proposed solutions. Our work provides further evidence towards suitability of optimization models in searching / pruning architectural design space.

CCS Concepts:•**Computer systems organization** → **Interconnection architectures; Multicore architectures;**

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: On-chip Network, Optimization Models

ACM Reference Format:
Nilay Vaish, Michael C. Ferris and David A. Wood, 2015. Optimization Models for Three On-chip Network Problems *ACM Trans. Architec. Code Optim.* V, N, Article A (January YYYY), 25 pages.
DOI: 0000001.0000001

1. INTRODUCTION

As Moore's Law continues to deliver exponential increases in the number of transistors, chip multiprocessors have begun to move from multicores to many-cores. As a result, designers are facing a combinatorial explosion in the design space, with an enormous number of ways (quantified in later sections) to allocate a chip's limited resources among cores, caches, on-chip interconnect, and memory controllers. Different designs have different computation and communication performance. Therefore, we need to search the design space for best possible designs. But even highly-constrained resource allocation problems, such as where to place memory controllers in the network [Abts et al. 2009], result in far more configurations than can be evaluated using traditional simulation-based techniques. Hence, we need models that can help in exploring the design space efficiently.

In this work[1], we present mathematical optimization based models for three problems related to on-chip networks for tiled processors like one shown in Figure 1. We assume the processor has a homogeneous set of cores each with two levels of private caches. These caches are kept coherent using a directory-based MOESI coherence protocol. The first two problems we model are: memory controller placement [Abts et al. 2009] (section 3), and resource allocation in heterogeneous on-chip networks [Mishra et al. 2011] (section 4). The combination of these two problems is a much more challenging non-convex optimization problem but explores the trade-offs between shared resources

[1]A portion of this work appears in chapter 6 of Nowatzki *et al.* [Nowatzki et al. 2013]. The work presented in sections 3 and 4 does not appear in Nowatzki *et al.* [Nowatzki et al. 2013]. The work presented in section 5 has more detailed analysis and evaluation of the model and of the designs obtained as solutions to the model. Figures 17 and 18 have been reused from Nowatzki *et al.* with permission from Morgan & Claypool Publishers.

Author's addresses: Nilay Vaish, Michael C. Ferris and David A. Wood, Department of Computer Sciences, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, Wisconsin, 53706, USA. email: nilay@cs.wisc.edu, ferris@cs.wisc.edu, david@cs.wisc.edu.

DOI: 0000001.0000001

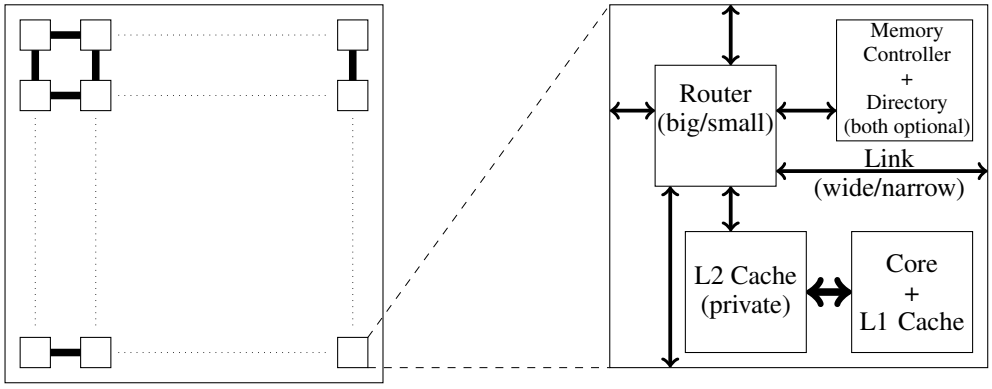


Fig. 1: Processor Layout (not to scale). On the left is an $n \times n$-tiled processor. The tiles are connected using a mesh / torus network. The figure on the right shows a single tile. The memory controller is optional. Routers for different tiles may have different amounts of resources. The links in between tiles can be wide or narrow.

in the design (section 5). We present our computational experience with the non-convex model and an equivalent linearized model, that can be solved much more efficiently.

Simulation-based experiments provide evidence that our solution for the combined problem provides 5.4% and 22% better performance on multi-programmed workloads composed of applications from SPEC CPU2006 [Henning 2006] and NAS Parallel Benchmarks [Bailey et al. 1991] respectively. While mathematical optimization has been used before in different facets of computer architecture including on-chip network design [Srinivasan et al. 2004; Kinsy et al. 2009; Marculescu and Bogdan 2009], compilers [Fu and Wilken 2002; Abdel-Gawad and Thottethodi 2011] and design space exploration [Azizi et al. 2010], we believe we are the first to use it for these three problems.

2. OVERVIEW OF MATHEMATICAL OPTIMIZATION

Mathematical Optimization deals with the problem of making the best possible choice from a large set of feasible alternatives. Choices are expressed as values taken by a collection of *variables*. *Feasible* choices are specified by constraints on the values of these variables. A designated function that evaluates a particular criterion quantifies the best choice. Mathematical optimization helps us make the *best possible* choice efficiently using the power of calculus, logic and mathematics.

Formally, an optimization problem has the form [Boyd and Vandenberghe 2004] –

$$\begin{aligned} &minimize \quad f_0(x) \\ &subject\ to \ \ f_i(x) \leq 0, \ \ i = 1, \cdots, m \end{aligned}$$

Here $x = (x_1, \cdots, x_n)$ is the vector of *optimization variables*, the function $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ is the *objective function*, the functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ are the *constraint functions*. A feasible vector x^* is called *optimal* if it has the smallest objective function value among all vectors $z \in \mathbb{R}^n$ that satisfy the constraints i.e. $f_0(x^*) \leq f_0(z)$ for any z for which $f_i(z) \leq 0, \ i = 1, \cdots, m$.

2.1. Types of Optimization Problems

Optimization problems can be divided into several different classes depending on the particular forms of the objective and the constraint functions. In our work, we make use of two different classes:

— *Mixed Integer Linear Program* (MILP) : In an MILP, the objective and the constraint functions are linear and one or more variables in the program take on integer-only values.

— *Mixed Integer Non-Linear Program* (MINLP) : This class includes problems that have at least one function (objective or constraint) that is not linear. Also, one or more variables take integer-only values.

Theoretically, problems in these classes are known to be NP-hard [Papadimitriou 1981]. Over the years, researchers have developed algorithms and tools that in many instances can solve these problems efficiently. We briefly describe the tools used in our work. General Algebraic Modeling System (GAMS) [GAMS Development Corporation 2015] is a language for representing optimization models. A particular data instance of the model is compiled into an efficient scalar representation, which is passed on to a solver. A solver is a tool that implements algorithms for solving optimization models. For example, Gurobi [Gurobi Optimization, Inc. 2015] is a solver for mixed integer linear and convex quadratic optimization problems, and Baron [Tawarmalani and Sahinidis 2005] is a solver for optimization problems defined with general nonlinear functions f_i. While a solver may not always be successful in finding an optimal solution for the model due to NP-hardness, the above solvers give solution guarantees relating the values found to the best possible ones.

2.2. Convexity

Certain mathematical concepts facilitate more efficient solution of these problems. We use notions related to convexity throughout this paper; interested readers may go through any standard text on mathematical optimization for a detailed discussion, e.g. [Boyd and Vandenberghe 2004]. The core concepts we use are:

— *Convex Set*: A set $C \subseteq \mathbb{R}^n$ is said to be convex if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have $\theta x_1 + (1-\theta)x_2 \in C$.
— *Convex Function*: A function $f : D \to \mathbb{R}$ is called convex if its **domain** $D \subset \mathbb{R}^n$ is a convex set and for all $x, y \in D$, and any θ with $0 \leq \theta \leq 1$, we have $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$.
— An optimization problem is called convex if it is of the form

$$\begin{aligned} &minimize && f_0(x) \\ &subject\ to && f_i(x) \leq 0, \quad i = 1, \cdots, m \\ &&& a_i^T x = b_i, \quad i = 1, \cdots, p \end{aligned}$$

where $f_0, \cdots, f_m$ are convex functions, a_i are n-dimensional real vector constants and b_i are scalar constants.

A point $x^* \in \mathbb{R}^n$ is locally optimal if it is feasible and its objective value is no worse than the objective value at all neighboring feasible points. It is globally optimal if the neighborhood is the whole space. For a convex optimization problem, any locally optimal solution is globally optimal. Local optimality is easily verified and thus efficient methods exist for solving convex problems.

Non-convex optimization problems, on the other hand, do not have efficient solution procedures in general. Methods exist that can provide solutions that are locally optimal but such a solution might be very far from the global optimum. In particular cases, it is possible to use convex problems to approximate or even exactly solve non-convex problems. For example, suppose we need to a solve a non-convex minimization problem $\mathcal{NCP}$. One may construct a convex problem, $\mathcal{CP}$, such that the objective function value for $\mathcal{CP}$ is at most the objective function value for $\mathcal{NCP}$ at all feasible points i.e. $\mathcal{CP}$ underestimates $\mathcal{NCP}$. Then, the optimal objective function value for $\mathcal{CP}$ gives us a lower bound on the optimal objective function value for $\mathcal{NCP}$. We show this pictorially in Figure 2. For integer linear programs, such bounds can be obtained by removing the integrality constraints. The bound can be used to compare the quality of any given feasible solution with the *unknown* optimal solution.

2.3. Optimization for Design Space Exploration

In our work, we focus on placement and allocation of network resources for many-core designs. The design space is so large that it is infeasible to enumerate / simulate its entirety. But the performance

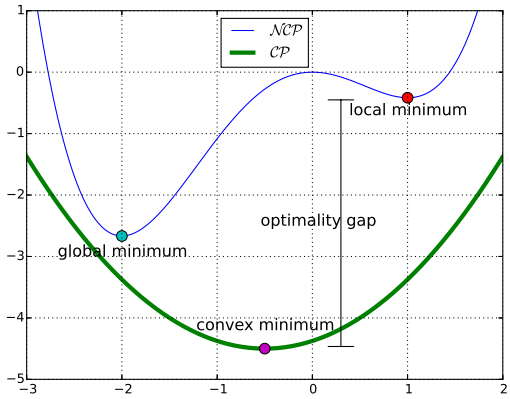


Fig. 2: Example for Convex Underestimation. $\mathcal{NCP}$ is non-convex function and $\mathcal{CP}$ is its convex under-estimate. Suppose some algorithm outputs the local minimum (shown in red) for $\mathcal{NCP}$. The minimum for $\mathcal{CP}$ can be used to compute the maximum possible gap between the *unknown* global minimum for $\mathcal{NCP}$ and the known local minimum.

objective and the design constraints can be effectively modeled using integer variables combined with linear / convex functions. While these optimization models are not as detailed as a simulation model, they are good enough to guide the exploration to regions of good performance. This is confirmed through simulation experiments where our designs perform better than previously proposed solutions.

3. PLACEMENT OF MEMORY CONTROLLERS

Chip Multiprocessors (CMPs) need abundant DRAM bandwidth to feed the increasing number of cores. Due to limited pin bandwidth [Abts et al. 2009], there will be more cores compared to the number of memory controllers on a chip. For example, the Oracle Sparc T5 [Feehrer et al. 2013] has 4 memory controllers for 16 cores. This raises the question of how to place the memory controllers within the on-chip network. Careful placement of the controllers can lead to lower latency and better bandwidth utilization for on-chip communication. This problem was introduced by Abts *et al.* [Abts et al. 2009].

Consider a design problem with n cores and m memory ports. Assuming the memory ports are co-located with the cores, there are $\binom{n}{m}$ possible ways of placing the memory controllers. For a 64-core, 16-port design, this number is about 4.9×10^{14}. It is not possible to explore each and every placement of the memory controllers. Abts *et al*. search this design space using a combination of intuition (experience), exhaustive simulation of smaller designs, and a *Genetic Algorithm* (GA) based approach to arrive at a reasonably good placement. Instead, we use mathematical optimization for searching the design space by creating an optimization-based model for the problem. We next discuss the assumptions used in the model.

3.1. Assumptions

Similar to Abts *et al*., we assume that the cores are laid out on a 2D-plane and are connected using an on-chip network. The memory ports are co-located with the cores. The on-chip network uses a deterministic routing protocol, i.e., all messages from node A to node B always traverse the same path [Jerger and Peh 2009]. Lastly, since CMPs typically distribute memory addresses across controllers using lower-order address bits [Feehrer et al. 2013], we assume the traffic is distributed uniformly across all memory controllers.

$$
\begin{aligned}
&\text{minimize} && z && \\
&\text{subject to} && && \\
&\sum_{(x,y)} I_{x,y} = && \text{number of memory controllers} && (1)\\
&LoadOnLink(l) = && \sum_{(x,y,x',y'):l\in Path(x',y',x,y)} I_{x,y}(R+K) && \\
&+ && \sum_{(x,y,x',y'):l\in Path(x,y,x',y')} I_{x,y}(RK+1) && (2)\\
&\text{where } l \text{ varies} && \text{over all the links in the network} && \\
&LoadOnLink(l) \leq && z && (3)\\
&\text{where } l \text{ varies} && \text{over all the links in the network} && \\
&I_{x,y} \in \{0,1\}, && z \in \mathbb{R}_+ &&
\end{aligned}
$$

Fig. 3: MILP for Placing Memory Controllers

Since the memory controllers are distributed (possibly) over the entire chip area, the chip would need to be manufactured such that the I/O pins cover the entire area of the chip and not just the periphery, as is the case with flip chip technology [Tong et al. 2013]. The design constraints on these I/O pins are not considered in our model. We also ignore the difference in areas of cores that have memory ports and those without.

3.2. Notation Used in the Model

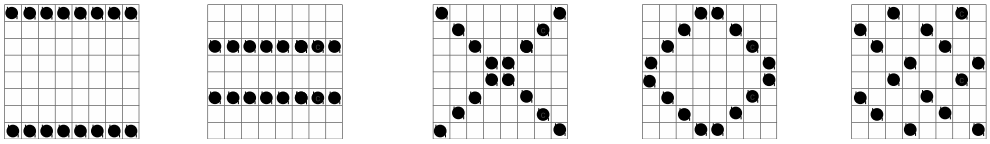
We model the problem as a Mixed Integer Linear Program (MILP). Figure 3 shows the formulated model. In the model, each memory controller is denoted as a point (x, y) on the 2-D plane. Similarly, a core is referred to with coordinates (x', y'). For each (x, y), $I_{x,y}$ is a binary variable denoting whether a memory controller is placed at (x, y). $LoadOnLink(l)$ denotes the load on the link l due to the communication between the cores and the memory controllers. This load depends on the placement of the controllers. The set $Path(x, y, x', y')$ contains all the links l that are used for going from (x, y) to (x', y'); since the routing protocol is deterministic, $Path()$ is an input to the problem. z bounds the maximum load a link can be assigned. We further assume that the read to write requests have a ratio of $R : 1$ and that a packet with data is K-times the size of a packet with no data.

3.3. Description of the Model

There are three constraints involved in the model.

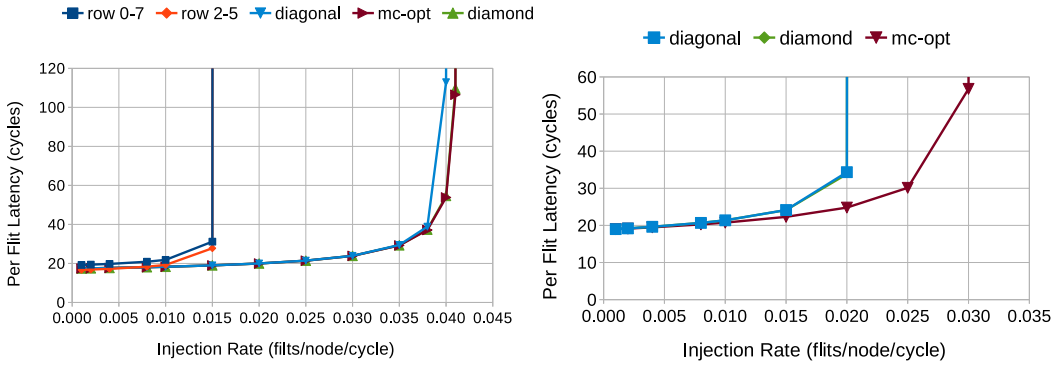
— Equation (1) enforces that a specified number of memory controllers (m) are placed on the chip. $I_{x,y}$ can be either 0 or 1. So when the equation is satisfied, exactly m of the n $I_{x,y}$-variables are set to 1, the rest are 0.

— Equation (2) defines the load on each link in the on-chip network. The first term on the right hand side of the constraint is a sum over all pairs (x, y) and (x', y') such that a memory controller is placed at (x, y) and the path from (x', y') to (x, y) uses link l. This term represents the request traffic going from the cores to the memory controllers. We assume the traffic has $R : 1$ ratio for reads and writes. Each read request requires a single flit, while a write request needs K flits. Hence, the total request traffic is proportional to $R + K$. The second term, which represents the response traffic from the memory controllers to the cores, can be interpreted in a similar fashion. The total response traffic is proportional to $RK + 1$, where RK is for flits with response data for read requests and 1 is for flits with acknowledgment for write requests.



(a) row 0-7 (b) row 2-5 (c) diagonal (d) diamond (e) mc-opt

Fig. 4: Different Controller Placements for an 8×8 Mesh/Torus Network. Tiles with black-colored sphere represent a memory port co-located with a core.



(a) Mesh Network (b) Torus Network

Fig. 5: Average Latency versus Injection Rate (Uniform Random Traffic)

— Equation (3) bounds the load on any link to be at most as large as z.

The objective of the problem is to minimize z, the maximum load on any of the links i.e. the *maximum channel load*. Thus an optimal solution for the model would place the memory controllers such that the maximum load placed on any of the links is as low as possible. A lower value for z means less congestion in the network and lower queueing delays. Alternately, we would like to use the available bandwidth efficiently by spreading the load across as many links as possible in a uniform manner. We chose this objective so as to keep the model linear.

3.4. Solving the Model

For solving the model, we assume the cores are connected using a $k \times k$ 2D- mesh or torus on-chip network. The number of memory controllers that need to be placed is $2k$. We also assume that the network uses dimension-ordered routing. Abts *et al*. made similar assumptions.

We express the model using GAMS [GAMS Development Corporation 2015] and solve it using Gurobi [Gurobi Optimization, Inc. 2015]. Figure 4 shows some of the possible placements for $k = 8$. The `diamond` placement, shown in Figure 4d, was reported as the best placement by Abts *et al*. Figure 4e, here on termed as `mc-opt`, is the placement obtained as a solution to the model when R and K are both set to 1. The same placement is optimal for both mesh and torus networks. Note that many other placements are also optimal.

3.5. Evaluation of the Model

Our model takes a very simplified view of an on-chip network. To validate that the simplified view is sufficient for the purpose of pruning the design space, we carry out different simulation-based experiments. All experiments use 8×8 mesh and torus networks.

Table I: Processor and Cache Simulation Parameters

Parameter	Value
Processors	64 out-of-order cores operating at 2GHz
L1 I Cache	32 KB, 2 way set associative, 2 cycle latency
L1 D Cache	64 KB, 2 way set associative, 2 cycle latency
L2 Cache (Private)	2 MB, 8 way set associative, 10 cycle latency
Main Memory	48 GB, 16 DDR-style controllers, addressed using bits 15:12.

3.5.1. Performance with Equi-probable Read and Write Synthetic Traffic. Our first experiment uses a detailed on-chip network simulator, which is part of gem5 [Binkert et al. 2011] and is based on GARNET [Agarwal et al. 2009]. During the simulation, all the cores inject request packets into the network at a fixed rate. The request addresses are generated in a manner so that the requests are uniformly distributed amongst the memory controllers. The memory controllers then respond to these requests, thus completing the request-response loop. Statistical results are collected after running the simulation for $2,000,000$ cycles.

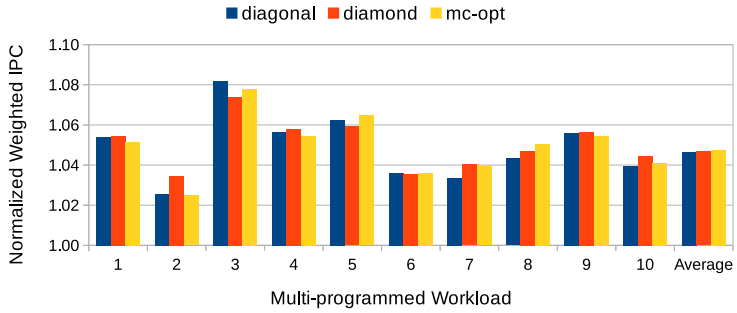
Figure 5 shows the plots of average flit latency versus the rate of request injection. Figure 5a is for a mesh network, while Figure 5b is for a torus network. From the graphs, we conclude that for the mesh network, `mc-opt` has performance similar to that of `diamond` and `diagonal`. For the torus network, `mc-opt` supports about 50% higher request injection rate before saturation, compared to `diagonal` and `diamond`. It also provides lower average latency. Thus, the design obtained from our optimization-based model works well when the traffic is uniformly distributed.

3.5.2. Performance with SPEC CPU2006 Applications.. In our second experiment, we evaluate the designs by simulating applications from the SPEC CPU2006 benchmark suite [Henning 2006] on the gem5 simulator. Such an experiment evaluates the efficacy of the design suggested by optimization-based model in a close to realistic situation.

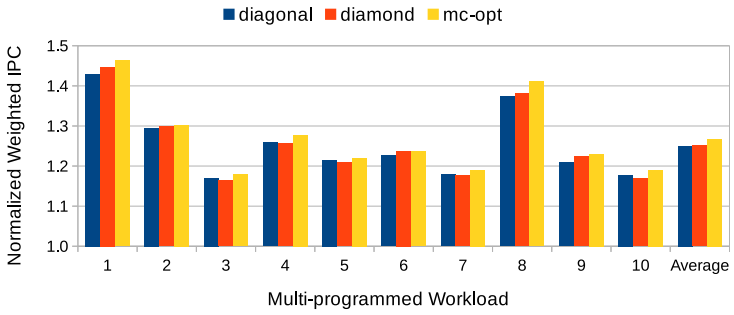
We briefly describe our experimental setup. We chose ten applications from the suite, namely, *astar, lbm, mcf, milc, omnetpp, libquantum, leslie3d, soplex, sphinx3,* and *GemsFDTD*. We created checkpoints for these applications after skipping first $100\ billion$ instructions. For each simulation, we dropped two applications and simulate eight copies of each of the eight remaining applications. These applications were mapped to the cores randomly. Each simulation was allowed to run till every core had executed at least $25\ million$ instructions. On average about $4\ billion$ instructions were simulated in each simulation. We begin statistics collection after the first $2\ million$ cycles complete. Relevant simulation parameters appear in Table I.

While there are a total of $\binom{10}{2}$ different combinations of the applications, we present results for only ten of these due to space constraints. We show the average over all the combinations. In Figure 6, we present the weighted speedup [Snavely and Tullsen 2000] obtained for the different application combinations by the different designs. The speedup is normalized to that for the `row 0-7` design. For the mesh network, `mc-opt` performs as well as `diagonal` and `diamond` designs. All these designs have, on average, about 4.5% better performance than the `row 0-7` design. For the torus network, `mc-opt` performs better than `diagonal` and `diamond` on almost all the combinations. On average, `mc-opt` improves performance by slightly more than 1% over `diagonal` and `diamond`.

Thus, the solution obtained from the model works well even in real situations. We expect `mc-opt` to perform even better on workloads that have higher cache miss rates and hence access the main memory more frequently.



(a) Mesh Network



(b) Torus Network

Fig. 6: Weighted speedup for different application combinations and controller placements normalized to `row 0-7`.

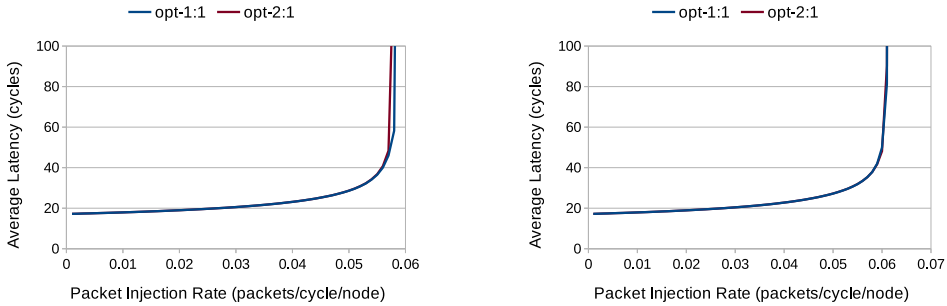
3.6. Why `mc-opt` Performs Better

For a torus network, `mc-opt` performs better than `diamond` and `diagonal` (which are *isomorphic* for this network). Our synthetic simulations show that `mc-opt` better spreads out the traffic. `diagonal` exhibits traffic hotspots around the clusters of four adjacent memory controllers, resulting in only 6.25% of the links observing at least 90% of the maximum channel load. In contrast, about 25% of links in `mc-opt` observe at least 90% of the maximum channel load. By more uniformly distributing memory controllers, `mc-opt` achieves more uniformly distributed traffic.

3.7. Effect of Read and Write Traffic Ratio on Controller Placement

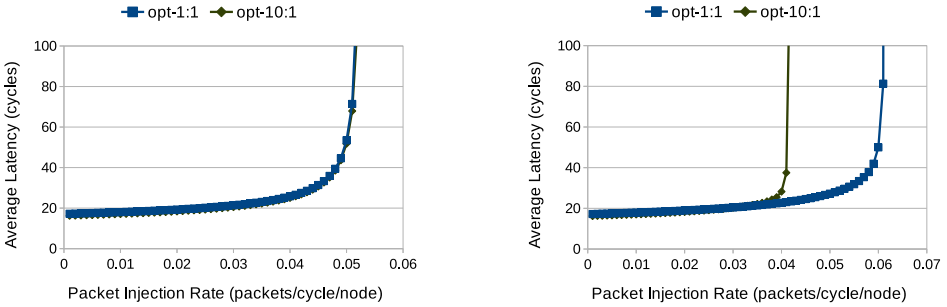
To gauge the effect of ratio of read versus write traffic on placement of controllers, we also solved our optimization model for two more settings of parameters: $(R = 2, K = 5)$ and $(R = 10, K = 5)$. The first of these settings represents the typical ratio of $2 : 1$ for reads to writes, while the second one is a rather extreme setting of $10 : 1$ for reads to writes ratio. A design obtained for read to write ratio $R : 1$ would be referred to as $opt - R : 1$. For example, the design with read to write ratio $2 : 1$ is referred to as $opt - 2 : 1$.

We evaluated the designs obtained using the on-chip network simulator described earlier. We show the results in Figure 7 and 8. As can be seen in the figures, we observed almost no difference in performance of the designs $opt - 1 : 1$ and $opt - 2 : 1$. We think this is because the difference in the volume of traffic from cores to memory controllers and from memory controllers to cores is not that significant for the designs considered. We also observed that $opt - 1 : 1$ performs slightly worse than $opt - 10 : 1$ on traffic with reads ten times as probable as writes. But $opt - 10 : 1$ performs



(a) Simulated traffic with read to write ratio 2:1 (b) Simulated traffic read to write ratio 1:1

Fig. 7: Average Latency versus Injection Rate (Uniform Random Traffic). The figure on the right is for traffic with read and write requests being equi-probable. The figure on the left is for traffic with read requests twice as probable as write requests. Both figures show optimal designs obtained by solving the model. $opt-1:1$ is for the setting $R=1$ and $opt-2:1$ is for the setting $R=2$.



(a) Simulated traffic with read to write ratio 10:1 (b) Simulated traffic read to write ratio 1:1

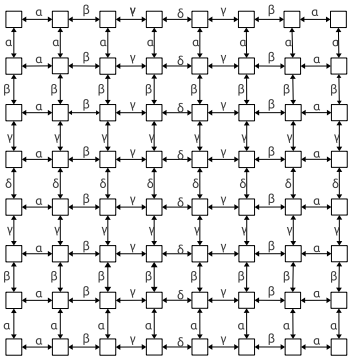
Fig. 8: Average Latency versus Injection Rate (Uniform Random Traffic). The figure on the right is for traffic with read and write requests being equi-probable. The figure on the left is for traffic with read requests ten times as probable as write requests. Both figures show optimal designs obtained by solving the model. $opt-1:1$ is for the setting $R=1$ and $opt-10:1$ is for the setting $R=10$.

significantly worse on traffic with read to write ratio 1:1. This is because $opt-10:1$ is optimized for the case when most of the traffic flows from memory controllers to cores.

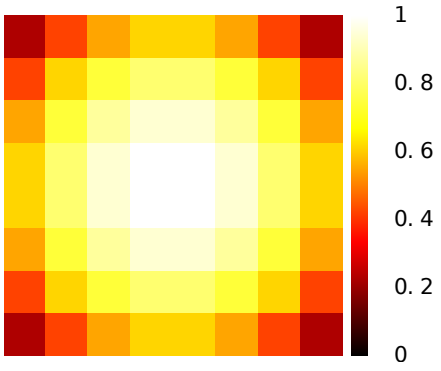
4. DESIGNING ON-CHIP NETWORK

Mishra *et al.* observed that in an on-chip network based on a mesh topology, the routers and links closer to the center of the mesh handle more traffic than those near the edge of the mesh [Mishra et al. 2011]. This is represented pictorially in Figure 9. But the routers, whether at the center or at the edge, are typically provided with the same amount of buffers and virtual channels. Mishra *et al.* therefore raised the question on how the resources should be distributed across the routers as it appears that allocating each router the same resources may not be optimal. These observations were made assuming that the routing protocol is deterministic and dimension-ordered, and that the traffic is uniformly distributed over all the paths permissible under the routing protocol.

As an answer, Mishra *et al.* designed a heterogeneous mesh network composed of two types of links—*wide* and *narrow*—and two types of routers – *big* and *small*. To arrive at a distribution of



(a) Distribution of traffic on different links. Each link is marked with the amount of traffic it observes relative to the links that observe the maximum amount of traffic. $\alpha = 0.4375,\ \beta = 0.75,\ \gamma = 0.9375,\ \delta = 1.$



(b) Heatmap for traffic distribution in a Mesh Network under uniform traffic assumption. Each cell represents a router. Its color represents the amount traffic passing through the router.

Fig. 9: Traffic Distribution in a Mesh Network.

these links and routers for an 8×8 mesh network, they evaluated several thousand design configurations for a 4×4 mesh network. The best configurations for the 4×4 mesh network were extrapolated to the 8×8 network. Figure 12 shows the designs proposed by Mishra *et al.*

An 8×8 mesh network requires 64 routers. Assuming $16-$big and $48-$small routers, there are $\binom{64}{48} \approx 4.89 \times 10^{14}$ possible ways in which these routers can be placed in the network. If the assumption that routers can only be *big* and *small* is dropped, the solution space explodes further. Given the size of the solution space, we use mathematical optimization for solving this resource allocation problem. In a nutshell, the designer has a fixed budget of resources and he/she needs to figure out the optimal resource division amongst the routers. The following sections describe a Mixed Integer Linear Program (MILP) for the problem.

4.1. Assumptions Made in the Model

To facilitate comparison, our optimization model relies on similar assumptions as Mishra *et al*. The resources available for designing an on-chip network are—links, virtual channels and buffers. We can vary the bandwidth of the physical links connecting different routers, the number of virtual channels and the number of buffers associated with each physical link. A physical link can be either *wide* or *narrow*. A wide link has twice the bandwidth of a narrow link. The sum total of the bandwidths of all the links in the network is bounded by the *link budget*. This budget is only for the links between the routers. The links between the cache controllers and the routers are assumed to be thin. The total number of virtual channels and buffers across all routers are bounded by the *vc budget* and the *buffer budget* respectively. There is an upper bound on the number of virtual channels and buffers that can be associated with a physical link. Apart from these assumptions on the resources, the traffic distribution in the network is assumed to be known a priori.

4.2. Notation Used in the Model

Our model appears in Figure 10. For each uni-directional link l in the mesh network, we introduce the following variables. W_l is an integer variable denoting whether link l is wide or narrow. A narrow link has width of 1, while a wide link has a width of 2. VC_l is an integer variable denoting the number of virtual channels associated with the physical link l. B_l is an integer variable for the number of buffers associated with link l. Variables t, s, and w denote the bandwidth, virtual channels, and buffers allocated to a link that has a load of one unit. Last of all, the function $LoadOnLink(l)$

$$
\begin{aligned}
& maximize && t + s + w \\
& subject\ to && \\
& \sum_l W_l \ \leq && \text{link budget} && (4)\\
& \sum_l VC_l \ \leq && \text{vc budget} && (5)\\
& \sum_l B_l \ \leq && \text{buffer budget} && (6)\\
& W_l \ \geq && LoadOnLink(l) * t && (7)\\
& VC_l \ \geq && LoadOnLink(l) * s && (8)\\
& B_l \ \geq && LoadOnLink(l) * w && (9)\\
& \text{where } l \text{ varies over all} && \text{the links in the network} && \\
& W_l \in \{1, 2\}, && VC_l, B_l \in \mathbb{Z}_+, s, t, w \in \mathbb{R}_+ &&
\end{aligned}
$$

Fig. 10: MILP for Distributing Network Resources

gives the load on link l. This function is an input to the problem as the traffic distribution is known beforehand.

4.3. Description of the Model

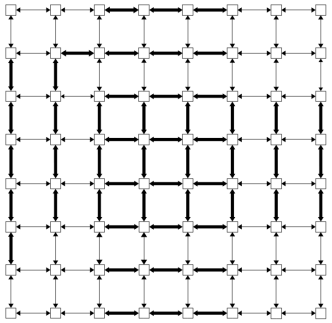
Now we describe our model in detail. Equations (4), (5) and (6) impose the budgetary constraints on the design. Intuitively, each link should have resources in proportion to the traffic that goes over that link. Such a distribution would allocate more resources to the routers at the center of the mesh network compared to the routers on the edge. Equations (7), (8) and (9) impose this proportionality constraint. We use $\geq$ (greater than or equals) relation in these constraints because W_l, VC_l and B_l are integer-valued and the terms appearing on the right-hand side are non-negative reals. The objective of the model is to maximize the sum: $t + s + w$. We chose this objective for two reasons:

— As mentioned before, we would like to assign resources to each link in proportion to the traffic going over that link. In our model, variables t, s and w represent the constants of proportionality for the distribution of different resources. Equations (7), (8) and (9) ensure the proportionality constraints. But the resources, left after the proportional distribution, can be distributed arbitrarily. For example, if t, s and w were set to 0, all the resources can be distributed arbitrarily amongst the links. On the other hand, if these variables took the maximum value they can possibly have, then almost all the resources will be distributed proportionately.
— The model also minimizes the maximum amount of traffic handled by a unit amount of resource. Consider equation (8). We can rewrite it as: $\frac{1}{s} \geq \frac{LoadOnLink(l)}{VC_l}$. Since s is maximized, $\frac{1}{s}$ is minimized. Thus the maximum load that a single virtual channel needs to bear is minimized.

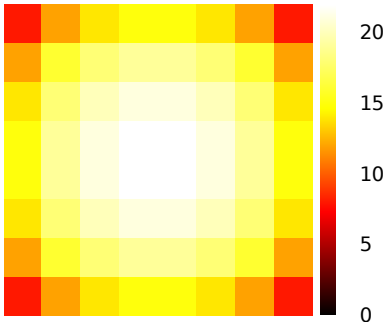
From the model, it can be seen that t, s, and w are independent of each other. Therefore, each of them will be independently maximized. In fact, the overall problem could be split into three independent optimizations.

4.4. Design Obtained from the Model

We solve the model under certain assumptions. The network is assumed to be an 8×8 mesh network, with dimension-order routing. Routers have up to six input and output ports. Two of these ports are for the private cache and the shared memory controllers respectively. The other four ports are for the four different directions. Lastly we assume that the traffic is distributed uniformly i.e. each cache

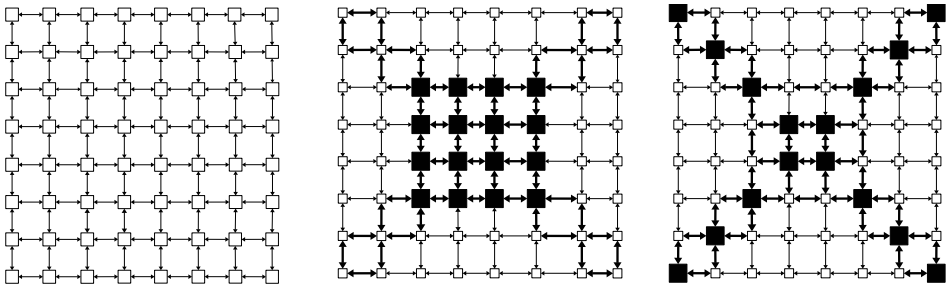


(a) Link Width Distribution. Wide links have been shown wider compared to narrow links.



(b) Virtual Channel Distribution. Each cell represents a router. Its color represents the number of virtual channels assigned to the router.

Fig. 11: `net-opt` Network Design



(a) base (b) center (c) diagonal

Fig. 12: Different Network Designs. Each box represents a router. Shaded routers are big. Wider links have been shown wider.

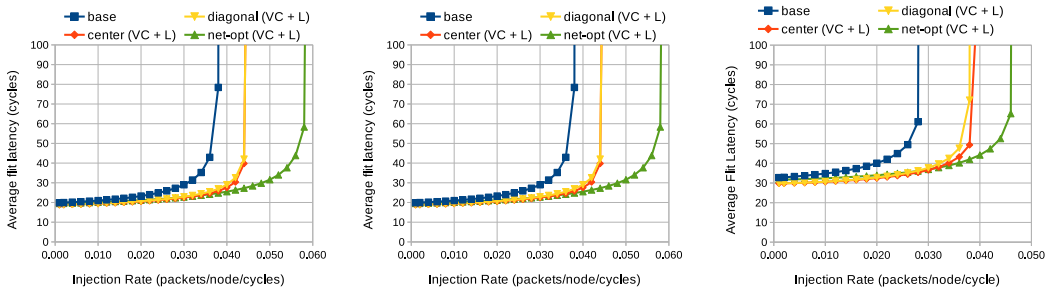
controller generates requests for any of the memory controllers with equal probability. Thus the traffic that a link needs to service is directly proportional to the number of paths using that link.

Figure 11 shows the design obtained on solving the model. Figure 11a shows the distribution of wide (256-bit) and narrow (128-bit) links across the 8×8 mesh network. All the links which observe heavy load (marked with γ and δ in Figure 9a) are wide in our design. Figure 11b shows a heat map for the distribution of the virtual channels amongst the routers. It can be observed that the number of virtual channels go down on moving from the center of the mesh towards the periphery. Since the traffic is assumed to be uniformly distributed, the routers at the center of the mesh service more traffic than those at the periphery, as shown in Figure 9b. Therefore, the design has most resources assigned to the routers at the center, and least to the ones farthest from the center. In the sequel, this design will be referred to as the `net-opt` design.

Note that there is slight asymmetry in the links in Figure 11a. Wider links, other than those in the middle three rows and columns, were assigned more bandwidth because of the available budget. If symmetry is important (e.g., to simplify layout), it is straight forward to add symmetry constraints.

4.5. Evaluation of the Design

We compare our design against the three designs analyzed by Mishra *et al*. These designs have been shown in Figure 12.



(a) Uniform Traffic (b) Tornado Traffic (c) Bit-Complement Traffic

Fig. 13: Average Latency vs Request Injection Rate for different request patterns

— `base`: All routers are provided equal resources. Each port has 3 virtual channels per virtual network. Each link is 192-bit wide.
— `center`: 16 routers in the center are big (6 virtual channels/ virtual network / port). The rest are small (2 virtual channels / virtual network / port). Half the links are wide (256-bit), while others are narrow (128-bit) as shown in Figure 12b.
— `diagonal`: Routers along the diagonals are big, rest are small. Again, half the links are wide and others are narrow. The design appears in Figure 12c. Mishra *et al* found this design to be the best.

All the designs in our experiments adhere to the same resource limits. Further, we assume that `base` operates at a clock frequency of 2.20 GHz and all the other designs operate at 2.07 GHz. This assumption is required since the routers in `center`, `diagonal` and `net-opt` are possibly bigger than the routers in `base` design. The frequencies assumed are same as those assumed by Mishra *et al*. Since `net-opt` makes use of routers that are smaller than the big routers in `center` and `diagonal` designs, our assumption on the frequency of these routers should be sufficient.

4.5.1. Performance With Synthetic Traffic.. We evaluate the candidate designs using the network simulator available in gem5 [Binkert et al. 2011]. The standard simulator supports a homogeneous network in which each physical link has the same width and the same number of virtual channels. We modified the simulator so that different links can be assigned different number of virtual channels and link widths.

In each simulation, the L2 controllers inject request packets into the network at a fixed rate. A memory controller, on receiving a request, injects the response packet for that request, thus completing the request-response loop. The simulation is allowed to run 2,000,000 cycles. At the end of the simulation, we note the average latency involved in transporting a flit from its source to the destination. The experiment is repeated with different rates of request injection, and with three different request patterns—uniform random, tornado, and bit-complement [Dally and Towles 2003].

Figure 13 shows the graphs for the average latency of a flit versus the rate of request injection into the network for these request patterns. As can be seen in the graphs, for all three request patterns, the `net-opt` design has a higher saturation bandwidth and provides lower average latency compared all other designs. This experiment shows the design obtained from the optimization-based model works well when a detailed network simulation is carried out, even when the uniformity assumption for the traffic is not observed. `net-opt` performs better than `diagonal` and `center` because the latter two designs distribute resources better than `base` but do not do so completely. In particular all the routers and links in rows and columns: 2, 3, 4 and 5 of the mesh network observe more traffic than rest as shown in Figure 9. `net-opt` provides most resources to these routers and links. In contrast, `diagonal` and `center` provide more resources to routers and links lying along the diagonals.

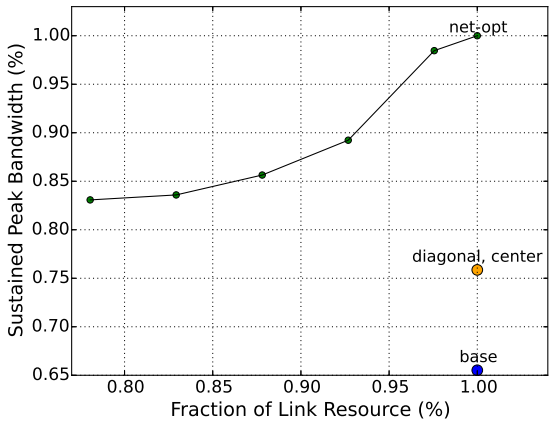


Fig. 14: Sustained peak bandwidth as a function of the link resources used.

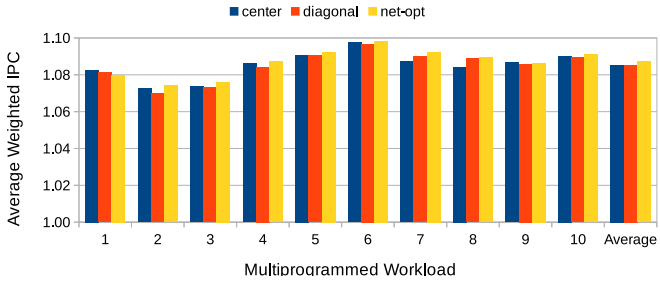


Fig. 15: Weighted Speedup for different application combinations and network designs normalized to `base`.

With the same setup as described above, we also simulated network designs that use lesser amount of link resources by reducing the number of wider links. In Figure 14, we show how the peak sustained bandwidth varies as a function of the amount of link resources. As we can see from the figure, optimal distribution of the link resources provides for similar or better performance even when the number of wider links used in the design is 30% less compared to `diagonal` and `center`.

4.5.2. Performance with SPEC CPU2006 Applications. We further evaluated the designs by simulating applications from the SPEC CPU2006 benchmark suite. The experimental setup was described in Section 3.5.2. In Figure 15, we present the weighted speedup [Snavely and Tullsen 2000] obtained for the different application combinations by the different designs. The speedup is normalized to that for the `base` design. `net-opt` performs as well as `diagonal` and `center` designs. All these designs have, on average, about 8% better performance than the `base` design. `net-opt` fails to perform better than `diagonal` and `center` as the network utilization is very low and all the designs observe nearly zero load latency for all the workloads.

5. ON-CHIP NETWORK DESIGN COMBINED WITH PLACEMENT OF CONTROLLERS

The model in the previous section assumes the traffic distribution as an input. But this distribution depends on the placement of the memory controllers, which in turn may depend upon the network design. Ideally we would like to place memory controllers and allocate network resources in a single combined problem. This may result in a better design than obtained by solving the two problems sequentially. With this intuition, we formulated the optimization model presented in Figure 16.

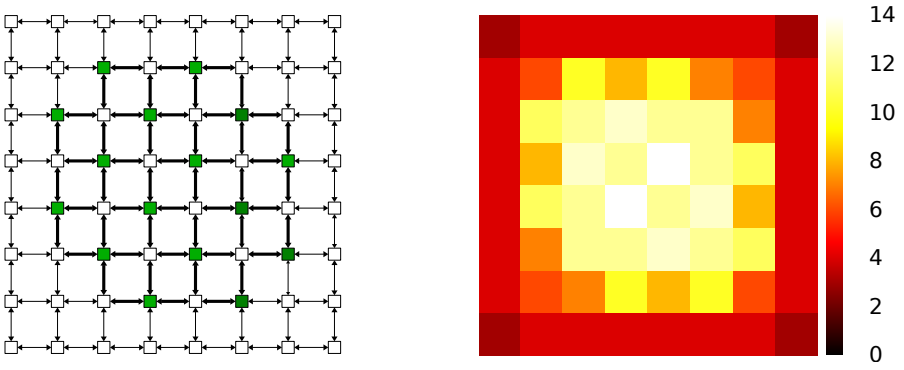
$$
\begin{aligned}
\text{minimize} \quad & W + S + T \\
\text{subject to} \quad & \\
\sum_l W_l \;&\leq\; \text{link budget} && (10)\\
\sum_l VC_l \;&\leq\; \text{vc budget} && (11)\\
\sum_l B_l \;&\leq\; \text{buffer budget} && (12)\\
W_l * T \;&\geq\; LoadOnLink(l) && (13)\\
VC_l * S \;&\geq\; LoadOnLink(l) && (14)\\
B_l * W \;&\geq\; LoadOnLink(l) && (15)\\
\text{where } l \text{ varies over} \quad & \text{all the links in the network} \\
LoadOnLink(l) \;&=\; \sum_{(x,y,x',y'):l\in Path(x',y',x,y)} I_{xy}(R+K) \\
&+\; \sum_{(x,y,x',y'):l\in Path(x,y,x',y')} I_{xy}(RK+1) \\
\text{where } l \text{ varies over} \quad & \text{all the links in the network} && (16)\\
\sum_{(x,y)} I_{x,y} \;&=\; \text{total memory controllers} && (17)\\
I_{x,y} + I_{x,y+1} \;&\leq\; 1 && (18)\\
I_{x,y} + I_{x+1,y} \;&\leq\; 1 && (19)\\
I_{x,y} \in \{0,1\},\;\; S,T,W \;&\in\; \mathbb{R}_+, W_l \in \{1,2\}, VC_l, B_l \in \mathbb{Z}_+
\end{aligned}
$$

Fig. 16: Non-linear Program for the Combined Problem

5.1. Analysis of the Model

Most of the variables and the constraints used in the model have been described in sections 3 and 4. We describe the additional variables and constraints:

— Variables T, S and W represent the load per unit bandwidth, virtual channel, and buffer respectively. They are the inverses of the variables t, s and w introduced in section 4.2. Note that while t, s and w can be independently optimized, T, S and W cannot be since they are linked by variables I_{xy}.
— Constraints (18) and (19) together avoid designs in which adjacent cells in the mesh network have memory controllers. We observed that without these (or similar) constraints, all the memory controllers are placed in the center portion of the chip. Such a design is likely to cause congested wire routing and thermal hot spots, hence the additional constraints in the model.
— Since memory controller placement determines traffic patterns, the function $LoadOnLink(l)$ is no longer an input. Also, we assume that caches are private to the cores and the on-chip network is only used for communication between the last level (private) caches and the memory controllers. Hence constraint (16) only accounts for traffic to and from memory controllers.
— Constraints (13), (14), and (15) have terms where two variables are being multiplied. These product terms make these constraints non-linear. Hence, the model is a *mixed integer non-linear program* (MINLP).



(a) Distribution of Memory Controllers and Link Widths (b) Virtual Channel Distribution

Fig. 17: `com-opt` design for the Combined Problem [Reproduced from Nowatzki *et al.* [Nowatzki et al. 2013] with permission from Morgan & Claypool Publishers]

— The reasons behind the choice of objective function are similar to the ones discussed in section 4.3.

Our model is **non-convex** because of two reasons. In our model, variables: W_l, VC_l, B_l and $I_{x,y}$ are constrained to be integer-valued and models with integrality constraints are non-convex. Even if we were to drop the integrality requirements, the model remains non-convex because constraints (13), (14), and (15) are not convex. These three constraints are of the form: $f(x, y, z) = z - xy \leq 0$. If we choose, for example, $a = (1, 1, 1)$ and $b = (2, 2, 4)$, then $f(a) = f(b) = 0$ but with $\theta = 0.5$, $f(\theta a + (1 - \theta)b) = 0.25$ so that

$$f(\theta a + (1 - \theta)b) \leq \theta f(a) + (1 - \theta) f(b)$$

is not satisfied. Thus f is not convex, implying that constraints (13), (14), and (15) and the overall model is not convex.

5.2. Solving the Model

We made several assumptions for solving the model. The network was assumed to be an 8×8 mesh network, with dimension-ordered routing, and uniformly distributed traffic.

No polynomial-time methods are known for solving non-convex MINLPs [Boyd and Vandenberghe 2004]. We explored the solution space for our problem using Baron [Tawarmalani and Sahinidis 2005], an NLP solver. The solver uses a branching scheme to optimize over the (bounded) feasible set. A good initial solution is very useful for pruning the search tree in order to find the globally optimal solution. To get closer to the optimal design, we seeded the solver with multiple initial designs, and experimented with the bounds on different variables. We used the placements described in section 3 as initial designs for the solver. The solver ultimately computed designs with improved objective function values. Figure 17 shows the best design we found.

As we show in Section 5.3, this is in fact the globally optimal solution. However, in the given time limit, Baron could not prove the global optimality of this solution. Rather, by relaxing the non-convex constraints and iteratively performing a branching procedure, it provides a lower bound on the optimal value of the objective function. The design shown in Figure 17 has an objective value which is 14% higher than the algorithmically computed lower bound. In comparison, the design of Mishra *et al*. (shown in Figure 18) has an objective value 55% higher than the lower bound.

5.3. Linearized Model

Our initial model has a linear objective function and nonlinear constraints with the form: $f(x, y, z) = z - xy \leq 0$. These are called bilinear terms since each is the product of two vari-

ables and the whole model is called a Bilinear Program (BLP). Moreover, each bilinear term is the product of a continuous variable and an integer variable, which can be converted to linear terms using binary expansion [Gupte et al. 2012]. After this conversion, we can solve the model using MILP techniques.

In our model, constraints (13), (14), and (15) are of the form: $z \leq xy$, where x is a continuous variable and y is an integer variable. Further, both x and y are non-negative and bounded from above i.e. $0 \leq x \leq a$ and $0 \leq y \leq b$. The set of points satisfying such a constraint can be represented as:

$$\mathcal{P} = \left\{ (x, y, z) \in \mathbb{R}_+ \times \mathbb{Z}_+ \times \mathbb{R} : \; z \leq xy, \; x \leq a, \; y \leq b \right\} \tag{20}$$

Using y's binary expansion, we get: $y = \sum_{i=1}^{k} 2^{i-1} w_i$. Here w_i are 0-1 integer variables and $k = \lfloor \log_2 b \rfloor + 1$. Define $\mathcal{B}$ as:

$$\begin{aligned} \mathcal{B} = \Big\{ & (x, y, z, w, v) \in \mathbb{R} \times \mathbb{Z} \times \mathbb{R} \times \{0,1\}^k \times \mathbb{R}^k : \\ & y = \sum_{i=1}^{k} 2^{i-1} w_i, \; y \leq b, \; z \leq \sum_{i=1}^{k} 2^{i-1} v_i, \; 0 \leq v_i \leq a w_i, \\ & v_i \leq x, \; v_i \geq x + a w_i - a, \text{for all } i \in \{1, \cdots, k\} \Big\} \end{aligned} \tag{21}$$

It can be shown that $\mathcal{P} = \text{Proj}_{x,y,z}(\mathcal{B})$. Here $\text{Proj}_{x,y,z}$ represents the projection operator that maps (x, y, z, w, v) to (x, y, z). Since $\mathcal{B}$ does not have any nonlinear term in its representation, it is an exact linearization of $\mathcal{P}$. We linearize our model by replacing constraints (13), (14), and (15) with the constraints used in defining $\mathcal{B}$. With this new model, it took CPLEX [IBM Decision Optimization 2015], another solver for MILP, less than 5 minutes to prove that the design presented in Figure 17 is in fact optimal.

5.4. Optimal Design

Figure 17 illustrates the design — referred to as `com-opt` — obtained from solving the model. Figure 17a shows the distribution of the memory controllers (solid boxes) and the wide (bold) and narrow (thin) links in the network. Figure 17b shows a heat map indicating the distribution of virtual channels. Note the significant differences from the optimal solutions to the individual problems in Figures 4e and 11.

We also obtained the design `center-opt` as the solution to our model from Figure 16 without constraints (18) and (19). In this design, the memory controllers are placed in the center of the chip. The objective value for `center-opt` is better than that of `com-opt` by 1.25%. We therefore expect `center-opt` to perform marginally better than `com-opt`.

5.5. Sensitivity of the Optimal Design vis-à-vis the Objective Function

The objective function in the combined model essentially gives equal consideration to links, virtual channels, and buffers. To test the sensitivity to this assumption, we generalized the objective function to provide weights to the variables S, T and W. Specifically, the objective from Figure 16 was changed to $\omega W + \psi S + \tau T$. We considered the following cases:

(1) $\omega = \psi = \tau = 1$: this case has been evaluated in section 5.4 and `com-opt` was obtained as the optimal design for the model.
(2) $\omega = 2, \psi = \tau = 1$: the design `com-opt` is optimal for this setting as well.
(3) $\omega = 1, \psi = 2, \tau = 1$: `com-opt` found to be optimal.
(4) $\omega = 10, \psi = 1, \tau = 1$: `com-opt` found to be optimal.
(5) $\omega = 1, \psi = 10, \tau = 1$: `com-opt` was the best design found, but the solver CPLEX was not able to prove that it is the optimal design. The optimality gap i.e. the gap between the objective of the best design found and the best under-estimate, was 18%.

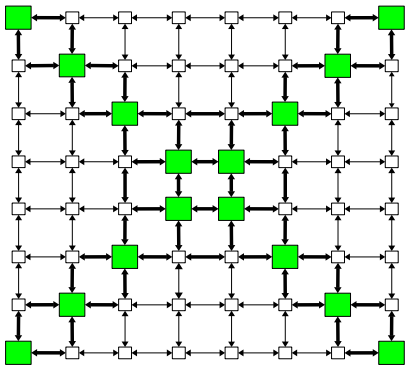


Fig. 18: Distribution of Memory Controllers, Buffers and Link Widths for `diagonal`. [Reproduced from Nowatzki *et al.* [Nowatzki et al. 2013] with permission from Morgan & Claypool Publishers]

5.6. Evaluation of the Designs

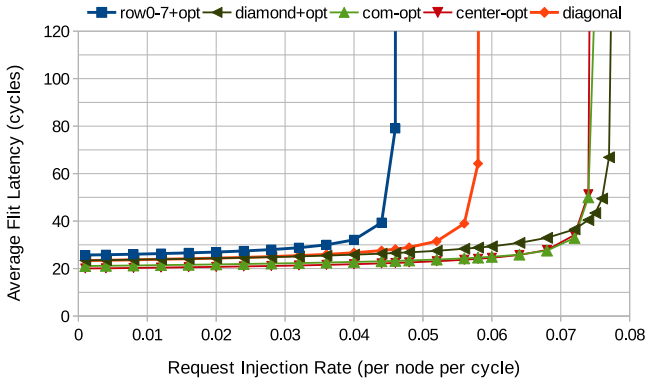
The objective function value for `com-opt` and `center-opt` is better compared to that for the design proposed by Mishra *et al*. Their design, referred to as `diagonal` and shown in Figure 18, places memory controllers on the diagonal nodes, along with big routers (6 virtual channels / port) and wide links. Routers on non-diagonal nodes are small (2 virtual channels / port) and use narrow links. To validate this result, we compare these designs using simulation. We also evaluate two other designs: `row0-7+opt` and `diamond+opt`. The placement of memory controllers for these is shown in Figure 4. The distribution of link widths and virtual channels was obtained by solving our optimization model with variables for memory controllers ($I_{x,y}$) set to fixed values. All designs use the same number of virtual channels, wide and narrow links, and buffers.

5.6.1. Synthetic Traffic.. We evaluated the designs using the NoC simulator described in section 4.5.1. In Figure 19a, we present the average latency experienced by a flit as a function of the rate of request injection into the network. From the graph, we can observe that `com-opt`, `center-opt` and `diamond+opt` support about 30% higher saturation bandwidth and provide lower latency compared to `diagonal`.

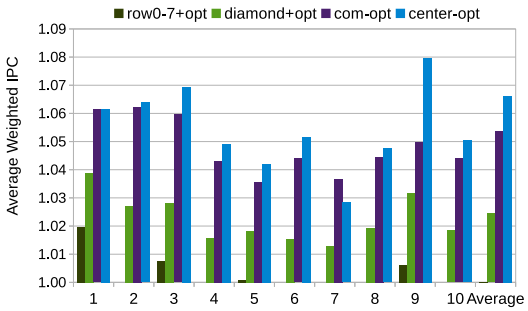
5.6.2. SPEC CPU2006 Benchmark.. We also evaluated the designs by simulating combinations of SPEC CPU2006 benchmarks [Henning 2006] on the gem5 simulator as described in section 3.5.2. In Figure 19b, we present the weighted speedup obtained for the different combinations. The speedup is normalized to the `diagonal` design. It can be observed that, on average, `com-opt` improves weighted speedup by 5.4% and `center-opt` improves weighted speedup by 6.7% over `diagonal`.

5.6.3. NAS Parallel Benchmark.. We further evaluated the designs by executing applications from NAS Parallel Benchmarks (NPB) [Bailey et al. 1991]. There are eight applications that comprise NPB. We provide results only for the ones that gem5 can execute properly. Others fail due to lack of support for x87 instructions in gem5. Each application is executed with 4 threads running on adjacent cores. Thus, we run 16 applications for a particular simulation to cover all the 64 cores. We also ran some workloads with a mix of these applications. The applications were mapped randomly to the cores for such workloads. Note that in these simulations, the on-chip network traffic is not uniformly distributed since the cache-to-cache transfers take place amongst caches private to cores executing threads from the same application.

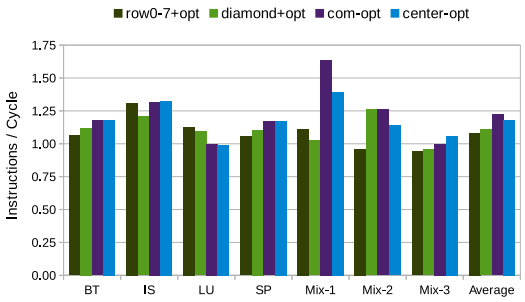
We executed each workload five times with different random seeds to harmonize any effects arising due to scheduling of threads. In Figure 19c, we present the improvement in IPC obtained



(a) Graph for Average Flit Latency vs Request Injection Rate for synthetically generated uniform random traffic.



(b) Weighted Speedup, normalized to `diagonal` for multiprogrammmed workloads composed from SPEC CPU2006 Applications.



(c) Instructions per cycle, normalized to `diagonal` for multithreaded workloads composed from NAS Parallel Benchmarks.

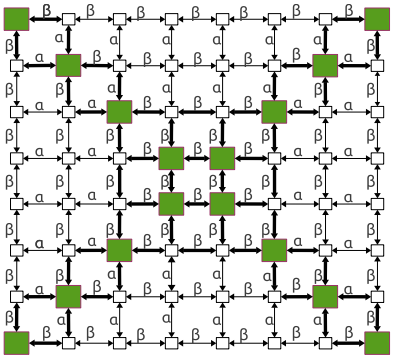
Fig. 19: Graphs from experimental evaluation of designs for the combined problem.

for different designs. The speedup is normalized to the `diagonal` design. It can be observed that `com-opt` performs about 22% better than the `diagonal`.

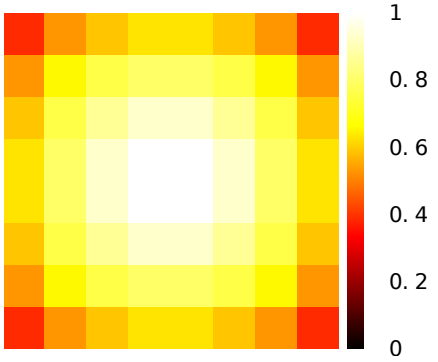
5.7. Analysis of the Designs

`com-opt` performs better than `diagonal` for two reasons:

— The zero load latency of `com-opt` is slightly lower than that of `diagonal`, as illustrated in Figure 19a. Hence, under low traffic intensity `com-opt` results in lower latency.
— For higher traffic intensity `com-opt` performs better since it does a better job of matching network resources to network traffic. Figures 20 and 21 illustrate the traffic load observed by the links and the routers for the two designs. Links marked α observe less traffic compared to links marked β. Ideally, the α links should be narrow and the β links should be wide. But `diagonal` has 16 wide α and 32 narrow β links, indicating a mismatched resource allocation. Similarly, `diagonal` assigns more virtual channels to routers near the corners even though they observe less traffic. By simultaneously placing memory controllers and allocating network resources, `com-opt` eliminates these resource mismatches.

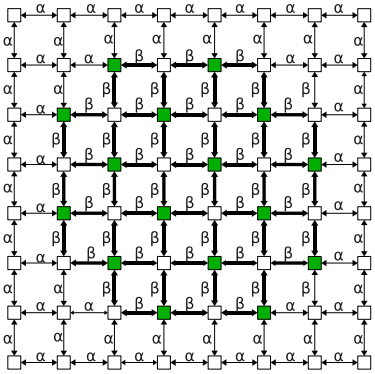


(a) Traffic load on different links. Links marked α are lightly loaded, while links marked β are heavily loaded.

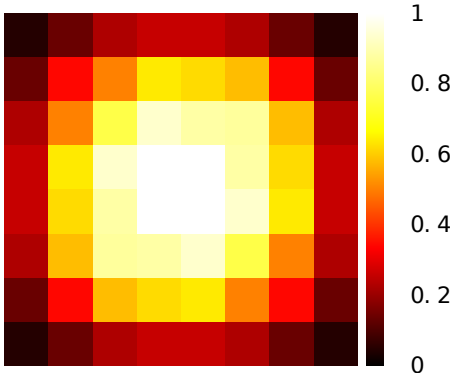


(b) Per Router Traffic Load

Fig. 20: Traffic Distribution for the `diagonal` Design



(a) Traffic load on different links. Links marked α are lightly loaded, while links marked β are heavily loaded.



(b) Per Router Traffic Load

Fig. 21: Traffic Distribution for the `com-opt` Design

6. IMPROVING THE MODELS

The design problems discussed in the paper have several possible variations that can be handled with out significant changes to the approach presented. The formulations provided can handle different types of resources, routing protocols, and topologies. But there are other dimensions along which the models can be improved to better relate with actual designs.

The models presented in this paper optimize the bandwidth available for on-chip traffic, following the lead of Abts *et al.*. However a different objective function may provide a better design point. As seen in Figures 5, 13 and 19, even though our designs provide much higher peak bandwidth, applications do not substantially benefit from the improved bandwidth. This indicates that *average delay* may be a better objective as many applications tend to be latency sensitive. A major part of delays in on-chip network is the serialization delay which depends on the bandwidth links have. Since we allocate bandwidth to links in our models, a model optimizing for average delay should account for changes to serialization delay. One may also use concepts from theory of queueing systems [Bertsekas and Gallager 1992] for modeling average delay.

Our models assume that on-chip traffic is uniformly distributed. If the traffic is not distributed uniformly, a weighting function should be introduced to associate a weight with each path. For ex-

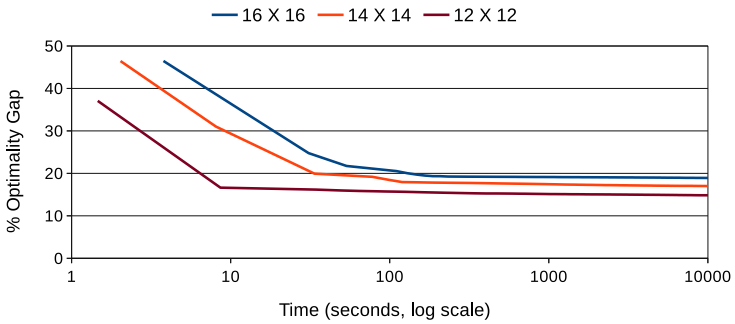


Fig. 22: Reduction in optimality gap as a function of time. An $n \times n$ problem refers to a problem with n^2 processors and $2n$ memory controllers. Optimality gap is the % difference in the objective value for the best design found and the best under-estimate.

ample, in asymmetric designs, it is likely that the traffic distribution would not be uniform. A subset of cores and memory controller may observe more traffic than the rest. Formulations would have to be adapted to take care of the asymmetry. Also, our models do not account for the effect of the cache hierarchy on the network traffic. Recent work on accurately modeling traffic distributions [Bogdan et al. 2010; Bogdan 2015] may be leveraged to improve the models. In design problems with time-varying traffic distributions, designing for multiple different traffic distributions using a stochastic optimization-based approach is also conceivable.

The formulations can also be extended by adding constraints on the power/energy consumed by the designs and thermal constraints that seek to avoid hot spots on the chip.

7. WHY USE OPTIMIZATION

Mathematical optimization, as a performance analysis tool, has several potential advantages over other approaches.

7.1. Scalability

For the memory controller placement problem, the genetic-algorithm approach of Abts*et al.* required running heuristic-based algorithms on multiple machines over more than a day [Gibson 2012]. Our approach found an optimal solution to the 64-processor, 16-port problem in less than a minute on a four core, eight thread machine. The same machine took less than 15 minutes to solve the 100-processor, 20-port problem. For bigger design problems, we were not able to prove the optimality of our solutions. In Figure 22, we show how the gap between the best design found and the best under-estimate (defined in section 2) for these design problems goes down with time. In each case, designs close to the estimated lower bound were found quickly. But closing the gap further would require a better model.

Similarly, Mishra *et al.* evaluated only several thousands of the possible 4×4 mesh network designs to arrive at what they thought was the best design for an 8×8 mesh network. Evaluating each design took 5-10 minutes, requiring multiple machines in parallel to reduce the total time [Mishra 2012]. Since the design space is huge, exploration via exhaustive / randomized simulation is impractical. In comparison, our optimization-based model took less than a minute for computing the design which is optimal under the given constraints.

Thus, mathematical optimization can potentially reduce the time required for design space exploration and is better suited for exploring larger and more complex design problems.

7.2. Flexibility

Extrapolation and divide-and-conquer based approaches require the design problem to be symmetric. These approaches use the symmetry to reduce the complexity of the solution space. For example,

the methods used by Abts *et al.* and Mishra *et al.* required the on-chip network to have the same dimension along the X and the Y axes. This was because the authors were trying to extrapolate results from a smaller 4×4 on-chip network to a larger 8×8 on-chip network. While optimization-based models also benefit from using symmetry, given the speed of computation, it may not be necessary to restrict the design space to symmetric designs only. For example, in optimization-based model, the X and the Y dimensions can differ. Similarly, it is not required that the number of memory ports be an integral multiple of the dimensions of the mesh / torus network. This was required in the original approach, again because of the need for extrapolation.

8. LIMITATIONS OF OPTIMIZATION

Despite the benefits, mathematical optimization may not be applicable to a large set of computer architecture problems. Optimization requires that the problem be represented using algebraic functions, which is not always possible. Many design problems require detailed models which may not be suitable for optimization techniques.

Secondly, optimization-based models may also face computational challenges in exploring the design space. MILPs are known to be NP-hard [Papadimitriou 1981], so it is not always possible to obtain an optimal solution in an acceptable time frame, limiting its use on time constrained operational problems. No practical algorithms are known for solving problems involving non-linear, and/or non-convex function, such as ED or ED^2, to global optimality [Boyd and Vandenberghe 2004]. Available solvers use different heuristics which might result in high computational effort or sub-optimal solutions.

Moreover, many problems in computer architecture require performing a trade-off amongst multiple objectives. For example, one may need to design an on-chip network where both the network bandwidth and the network latency need to be optimized. For such problems, one may have to solve multiple instances of the optimization model to trace out a Pareto frontier. This is only possible if solving individual instances is fast enough. Otherwise, one may have to weight the different objectives based on expert opinion.

Lastly, a given solution is only optimal for the specific instance of the model. Changes to inputs or the model's structure may make the solution suboptimal or infeasible. One may have to use techniques like stochastic optimization [Shapiro et al. 2009] if only probability distributions are known for the input parameters or robust optimization when only bounds on data uncertainty are given.

9. RELATED WORK

We have made a case for using mathematical optimization for solving design problems in the field of Computer Architecture. Our work is highly influenced by the work of Abts *et al.* [Abts et al. 2009] and Mishra *et al.* [Mishra et al. 2011]. In this section, we highlight other recent work in the area and discuss why our approach is different.

On-chip Networks: Designing on-chip networks is a fertile area of research. Significant effort has been devoted towards improving the performance and reducing the cost of on-chip networks. Prior work has proposed adaptive routing [Ma et al. 2011; Ma et al. 2012], bufferless NoC [Hayenga et al. 2009; Moscibroda and Mutlu 2009], and QOS support for NoCs [Grot et al. 2011]. These approaches alter the dynamic behavior of the network. Ben-Itzhak *et al.* proposed using simulated annealing for designing heterogeneous NoCs [Ben-Itzhak et al. 2012]. We discussed earlier that mathematical optimization works better when the problem can be expressed as a linear / integer linear / convex program. Our work shows that this is true for certain problems related to NoC-design. Heuristic-based approaches may work better for non-convex design problems. Jang *et al.* analyze traffic patterns for GPGPU applications partitioning schemes for virtual channels that improve resource utilization for on-chip networks [Jang et al. 2015]. They also evaluate how these schemes interact with different routing protocols and memory controller placements.

Mathematical optimization has been used by other researchers to solve on-chip network design problems. Kinsy *et al.* use MILP-based approach for producing deadlock-free routes [Kinsy et al.

2009]. Abdel-Gawad and Thottethodi propose program transformations for more efficient stream communication [Abdel-Gawad and Thottethodi 2011] . These transformations are modeled using MILP which are solved at compile time. Srinivasan *et al.* present an MILP-based approach for designing on-chip networks with homogeneous routers [Srinivasan et al. 2004]. The problem we study looks at distributing link bandwidths, virtual channels and buffers to heterogeneous routers. We also combine the heterogeneous network design and memory controller placement problems together, which has not been solved using MILP before.

On-chip Placement: Prior work [Awasthi et al. 2010; Wang and O'Boyle 2009] has focused on figuring out the best mapping for applications and data on to cores and memory controllers at execution time, while we have presented a design time approach. A lot work for on-chip placement has been done in the SoC domain. These works [Zhou et al. ; Hung et al.] propose using genetic algorithms for generating solutions. Xu *et al.* also tackled the problem of placing memory controllers for chip multiprocessors [Xu et al. 2011]. They solved the problem for a 4×4 CMP through exhaustive search. To find the best placement for the 8×8 problem, they exhaustively search through solutions obtained by stitching solutions obtained for the 4×4 problem. This reduces the solution space that needs to be searched, but the idea is not generic. It assumes that the chip can be divided into smaller regions and solutions for the smaller regions can composed to generate optimal solutions for larger regions. This may not hold true in general. Our approach of using mathematical optimization does not rely on any such assumption.

Theory: Network design problem has been widely studied in theoretical computer science [Magnanti and Wong 1984; Goemans et al. 1994], particularly with respect to designing distribution, transportation, telecommunication and other types networks. These works mainly focus on designing approximation algorithms for the different variants of the network design problem and on analyzing their theoretical complexity. We focus on on-chip network design and on-chip placement.

10. CONCLUSION

In this paper, we solved three problems related to on-chip networks. The solutions were obtained by using mathematical optimization models for searching and pruning the design space. We believe there are other computer architecture problems that can benefit from optimization. It might be worthwhile to explore whether there are optimization techniques that are better suited for problems in computer architecture.

11. ACKNOWLEDGMENTS

We thank the editors, the anonymous reviewers and members of the Multifacet group for their insightful comments and feedback on the paper. Nilay would like to thank Siddharth Barman, Arkaprava Basu, Brad Beckmann, Dan Gibson, Taedong Kim, Asit K Mishra, Somayeh Sardashti, Rathijit Sen, and Srikrishna Sridhar for their help. This material is based upon work partially supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contract number DE-AC02-06CH11357 through the Project "Multifaceted Mathematics for Complex Energy Systems", AFOSR Grant FA9550-15-1-0212, and NSF (IIS-1227530, CCF-1218323, CNS-1302260, CCF-1438992, and CCF-1533885). Professor Wood has a significant financial interest in AMD.

REFERENCES

Ahmed H. Abdel-Gawad and Mithuna Thottethodi. 2011. TransCom: Transforming Stream Communication for Load Balance and Efficiency in Networks-on-chip. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, USA, 237–247. DOI:http://dx.doi.org/10.1145/2155620.2155648

Dennis Abts, Natalie D. Enright Jerger, John Kim, Dan Gibson, and Mikko H. Lipasti. 2009. Achieving Predictable Performance Through Better Memory Controller Placement in Many-core CMPs. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 451–461. DOI:http://dx.doi.org/10.1145/1555754.1555810

Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator.. In *ISPASS* (2009-05-26). IEEE, 33–42.

Manu Awasthi, David W. Nellans, Kshitij Sudan, Rajeev Balasubramonian, and Al Davis. 2010. Handling the Problems and Opportunities Posed by Multiple On-chip Memory Controllers. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10)*. ACM, New York, NY, USA, 319–330. DOI: http://dx.doi.org/10.1145/1854273.1854314

Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis. In *Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10)*. ACM, New York, NY, USA, 26–36. DOI: http://dx.doi.org/10.1145/1815961.1815967

D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. 1991. The NAS Parallel Benchmarks&Mdash;Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing (Supercomputing '91)*. ACM, New York, NY, USA, 158–165. DOI: http://dx.doi.org/10.1145/125826.125925

Yaniv Ben-Itzhak, Israel Cidon, and Avinoam Kolodny. 2012. Optimizing Heterogeneous NoC Design. In *Proceedings of the International Workshop on System Level Interconnect Prediction (SLIP '12)*. ACM, New York, NY, USA, 32–39. DOI: http://dx.doi.org/10.1145/2347655.2347670

Dimitri Bertsekas and Robert Gallager. 1992. *Data Networks*. Prentice Hall.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39 (Aug. 2011), 1–7. Issue 2. DOI: http://dx.doi.org/10.1145/2024716.2024718

Paul Bogdan. 2015. Mathematical Modeling and Control of Multifractal Workloads for Data-Center-on-a-Chip Optimization. In *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS '15)*. ACM, New York, NY, USA, Article 21, 8 pages. DOI: http://dx.doi.org/10.1145/2786572.2786592

Paul Bogdan, Miray Kas, Radu Marculescu, and Onur Mutlu. 2010. QuaLe: A Quantum-Leap Inspired Model for Non-stationary Analysis of NoC Traffic in Chip Multi-processors. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS '10)*. IEEE Computer Society, Washington, DC, USA, 241–248. DOI: http://dx.doi.org/10.1109/NOCS.2010.34

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, NY, USA.

William Dally and Brian Towles. 2003. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

J. Feehrer, S. Jairath, P. Loewenstein, R. Sivaramakrishnan, D. Smentek, S. Turullols, and A. Vahidsafa. 2013. The Oracle Sparc T5 16-Core Processor Scales to Eight Sockets. *Micro, IEEE* 33, 2 (March 2013), 48–57. DOI: http://dx.doi.org/10.1109/MM.2013.49

Changqing Fu and Kent Wilken. 2002. A Faster Optimal Register Allocator. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO 35)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 245–256. http://dl.acm.org/citation.cfm?id=774861.774888

GAMS Development Corporation. 2015. General Algebraic Modeling System (GAMS) Release 24.4.3. Washington, DC, USA. (2015). http://www.gams.com/

Dan Gibson. 2012. Private communication. (2012).

M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, É. Tardos, and D. P. Williamson. 1994. Improved Approximation Algorithms for Network Design Problems. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '94)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 223–232. http://dl.acm.org/citation.cfm?id=314464.314497

Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. 2011. Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 401–412. DOI: http://dx.doi.org/10.1145/2000064.2000112

Akshay Gupte, Shabbir Ahmed, Myun Seok Cheon, and Santanu S Dey. 2012. Solving Mixed Integer Bilinear Problems using MIP formulations. (2012).

Gurobi Optimization, Inc. 2015. Gurobi Optimizer Reference Manual. (2015). http://www.gurobi.com

Mitchell Hayenga, Natalie Enright Jerger, and Mikko Lipasti. 2009. SCARAB: A Single Cycle Adaptive Routing and Bufferless Network. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. ACM, New York, NY, USA, 244–254. DOI: http://dx.doi.org/10.1145/1669112.1669144

John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News* 34, 4 (Sept. 2006), 1–17. DOI: http://dx.doi.org/10.1145/1186736.1186737

W.-L. Hung, Y. Xie, N. Vijaykrishnan, C. Addo-Quaye, T. Theocharides, and M.J. Irwin. Thermal-aware floorplanning using genetic algorithms. In *International Symposium on Quality of Electronic Design, 2005.* 634 – 639. DOI: http://dx.doi.org/10.1109/ISQED.2005.122

IBM Decision Optimization. 2015. IBM ILOG CPLEX Optimizer. (2015). "http://www.cplex.com"

Hyunjun Jang, Jinchun Kim, Paul Gratz, Ki Hwan Yum, and Eun Jung Kim. 2015. Bandwidth-efficient On-chip Interconnect Designs for GPGPUs. In *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, Article 9, 6 pages. DOI: http://dx.doi.org/10.1145/2744769.2744803

Natalie D. Enright Jerger and Li-Shiuan Peh. 2009. *On-Chip Networks*. Morgan & Claypool Publishers.

Michel A. Kinsy, Myong Hyon Cho, Tina Wen, Edward Suh, Marten van Dijk, and Srinivas Devadas. 2009. Application-aware Deadlock-free Oblivious Routing. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 208–219. DOI: http://dx.doi.org/10.1145/1555754.1555782

Sheng Ma, Natalie Enright Jerger, and Zhiying Wang. 2011. DBAR: An Efficient Routing Algorithm to Support Multiple Concurrent Applications in Networks-on-chip. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 413–424. DOI: http://dx.doi.org/10.1145/2000064.2000113

Sheng Ma, Natalie Enright Jerger, and Zhiying Wang. 2012. Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA '12)*. IEEE Computer Society, Washington, DC, USA, 1–12. DOI: http://dx.doi.org/10.1109/HPCA.2012.6169049

T. L. Magnanti and R. T. Wong. 1984. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science* 18 (1984), 1–56.

Radu Marculescu and Paul Bogdan. 2009. *The Chip Is the Network: Toward a Science of Network-on-Chip Design*. http://dx.doi.org/10.1561/1000000011

Asit K. Mishra. 2012. Private communication. (2012).

Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. 2011. A Case for Heterogeneous On-chip Interconnects for CMPs. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*. ACM, New York, NY, USA, 389–400. DOI: http://dx.doi.org/10.1145/2000064.2000111

Thomas Moscibroda and Onur Mutlu. 2009. A Case for Bufferless Routing in On-chip Networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*. ACM, New York, NY, USA, 196–207. DOI: http://dx.doi.org/10.1145/1555754.1555781

Tony Nowatzki, Michael Ferris, Karthikeyan Sankaralingam, Cristian Estan, Nilay Vaish, and David Wood. 2013. Optimization and Mathematical Modeling in Computer Architecture. *Synthesis Lectures on Computer Architecture* 8, 4 (2013), 1–144. DOI: http://dx.doi.org/10.2200/S00531ED1V01Y201308CAC026

Christos H. Papadimitriou. 1981. On the Complexity of Integer Programming. *J. ACM* 28, 4 (Oct. 1981), 765–768. DOI: http://dx.doi.org/10.1145/322276.322287

Alexander Shapiro, D Dentcheva, and A Ruszczynski. 2009. *Lectures on Stochastic Programming*. SIAM.

Allan Snavely and Dean M. Tullsen. 2000. Symbiotic Jobscheduling for a Simultaneous Multithreaded Processor. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*. ACM, New York, NY, USA, 234–244. DOI: http://dx.doi.org/10.1145/378993.379244

K. Srinivasan, K.S. Chatha, and G. Konjevod. 2004. Linear programming based techniques for synthesis of network-on-chip architectures. In *ICCD*. 422–429. http://dx.doi.org/10.1109/ICCD.2004.1347957

Mohit Tawarmalani and Nikolaos V. Sahinidis. 2005. A polyhedral branch-and-cut approach to global optimization. *Math. Program.* 103, 2 (June 2005), 225–249. DOI: http://dx.doi.org/10.1007/s10107-005-0581-8

Ho-Ming Tong, Yi-Shao Lai, and C.P. Wong. 2013. *Advanced Flip Chip Packaging*. Springer.

Zheng Wang and Michael F.P. O'Boyle. 2009. Mapping Parallelism to Multi-cores: A Machine Learning Based Approach. In *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '09)*. ACM, New York, NY, USA, 75–84. DOI: http://dx.doi.org/10.1145/1504176.1504189

Thomas Canhao Xu, Pasi Liljeberg, and Hannu Tenhunen. 2011. Optimal Memory Controller Placement for Chip Multiprocessor. In *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '11)*. ACM, New York, NY, USA, 217–226. DOI: http://dx.doi.org/10.1145/2039370.2039405

Wenbiao Zhou, Yan Zhang, and Zhigang Mao. Pareto based Multi-objective Mapping IP Cores onto NoC Architectures. In *IEEE Asia Pacific Conference on Circuits and Systems, 2006.* 331 –334. DOI: http://dx.doi.org/10.1109/APCCAS.2006.342418