POSEIDON: A Framework for Application-Specific Network-on-Chip Synthesis for Heterogeneous Chip Multiprocessors

Soohyun Kwon[1], Sudeep Pasricha[2], Jeonghun Cho[1]
[1]Electrical Engineering and Computer Science, Kyungpook National University, Daegu, Korea
[2]Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO, USA
[1]E-mail: {shkwon, jcho}@ee.knu.ac.kr, [2]E-mail: sudeep@colostate.edu

Abstract

In recent years, the rise in the number of cores being integrated on a single chip has led to a greater emphasis on scalable communication fabrics that can overcome data transfer bottlenecks. Network-on-Chip (NoC) architectures have been gaining widespread acceptance as communication backbones for multi-core systems, due to their high scalability, predictability, and performance. However, NoCs are also power hungry, and synthesizing a NoC fabric for a particular application requires solving a multitude of non-trivial design problems. Due to the large design space associated with various possible NoC configurations and design constraints, it is critical to automate the exploration process and arrive at a customized NoC that meets performance goals, while minimizing power and peak temperature. In this paper, we present a novel application specific NoC synthesis framework (POSEIDON) that combines multiple algorithms and heuristics to efficiently explore the solution space. Our results indicate that the proposed framework provides a reduction of up to 15.7% in power consumption, 21.08% in average latency, 27.05% in total energy, and 42.7% in energy-delay product compared to state-of-the-art approaches, as well as a 4.2% reduction in peak temperature when the framework is customized for thermal-aware synthesis.

Keywords

Networks on chip, synthesis, heterogeneous chip multiprocessors

1. Introduction

With the increase in complexity of chip multiprocessors (CMPs) in recent years, the need for scalable and efficient communication architectures to support inter-core data transfers has become paramount. Network-on-Chip (NoC) fabrics [1] have been shown to provide superior communication bandwidth, scalability, and modularity compared to traditional bus-based architectures. Not surprisingly then, NoCs have gradually gained acceptance as the dominant interconnection paradigm for emerging CMP systems with tens to hundreds of cores [2]-[4].

NoCs can be broadly classified as regular or application-specific. For regular NoCs, a majority of the existing solutions [5]-[8] assume a uniform mesh-based NoC topology to connect cores of the same size, with the focus being on optimally mapping cores to the mesh. For application specific NoCs, the design challenges are different because of the irregular core sizes, core layout constraints, and diverse communication flow requirements [9]-[12]. Most modern CMP systems contain numerous heterogeneous cores such as programmable processors, accelerators, ASICs, and memories that are non-uniform in size. Designing NoCs for such many-core application-specific systems requires an elaborate design phase that involves solving a variety of design problems: topology selection and mapping, physical planning, routing, and other optimization tasks. The large number of options and constraints makes it impossible to fully explore the solution space in a reasonable amount of time. The focus of this paper is on automating the design exploration phase to generate optimized NoC fabrics that satisfy application performance constraints.

One of the key challenges in designing a viable application-specific NoC fabric is to minimize the overhead of switches, network interfaces, and links, all of which consume area and power. To raise performance-per-watt in emerging CMPs, it is vital to reduce the NoC power consumption, especially as some recent prototypes have shown that NoCs can consume a significant portion of overall system power: ~30% in the Intel 80-core teraflop chip [2] and ~40% in the MIT RAW chip [3]. NoC power consumption is influenced by several factors such as total wire length, network interface (NI) and switch placement, and communication path selection. The same factors also impact the thermal profile of the system. The high levels of integration in nanometer technologies has given rise to thermal hotspots and gradients, which not only reduce performance (e.g., 71% increase in wire delay due to a 150^{o}C hotspot, and 15% wire delay increase due to a 50^{o}C gradient, at 32nm [13]), but also increase leakage power consumption, and raise the probability of chip failure. Consequently, NoC fabrics must be designed not only for low power, but also in a manner that reduces peak temperature.

This paper presents a unified framework (POSEIDON) for synthesizing application-specific NoCs for emerging CMP systems. Unlike existing approaches, we incorporate thermal awareness into our NoC synthesis framework, with the goal of reducing peak temperature on the die. Other novel aspects of our framework include incorporating switch sizing constraints based on their clocked frequency, considering custom link sizing, and exploring NI placement to aggressively reduce power consumption. Our proposed framework is shown to provide much lower average power consumption, average latency, total energy, and energy-delay product than current state-of-the-art approaches. With thermal optimizations enabled, our framework also leads to lower on-die peak temperatures, which is critical in the nanometer era.

2. Related Work

In recent years, there have been a number of research efforts focusing on regular NoC topology synthesis [5]-[8]. These works primarily focus on mapping uniform sized cores and their communication flows on regular mesh topologies to optimize energy and performance. It has however been shown that for heterogeneous multi-core systems, application-specific custom NoCs are superior to regular topologies in terms of energy and area [9][14]. Consequently, recent works have targeted application-specific NoC synthesis [9]-[12]. We summarize these efforts below.

Murali et al. [9] presented an application-specific NoC synthesis technique that utilized min-cut partitioning to allocate switches to groups of cores and minimize NoC power consumption. However, the partitioning step is carried out before the floorplanning step, resulting in physical-level information such as distances between cores not being taken into account during synthesis. Besides, area of switches and NIs is not considered during topology generation. Srinivasan et al [10][11] presented a combination of heuristics and integer linear programming (ILP) to generate a low power custom NoC topology. However, the problem of NI placement and switch sizing is ignored. Chan et al. [12] proposed an iterative refinement strategy to generate an optimized NoC topology that supports both packet-switched networks and point to point connections, using the methodology from [9], along with its limitations. Several works

have explored thermal aware core to die mapping [15] and task scheduling [16], as well as runtime NoC reconfiguration to avoid hotspots [17], but these efforts have not considered thermal awareness during application-specific NoC topology synthesis.

In this paper, we present the POSEIDON framework that extends the state-of-the-art in application-specific NoC topology synthesis in two ways. First, we develop a collection of heuristics and algorithms for custom NoC synthesis that more aggressively reduces communication power, energy, and latency compared to earlier approaches. Secondly, we incorporate the option of thermal-aware NoC topology and communication flow synthesis in the framework, and show how we can trade-off lower peak temperatures with other design constraints.

3. Problem Formulation

In this section, we present relevant definitions and then describe the application-specific NoC synthesis problem. We assume that hardware/software partitioning has already been performed, and the application tasks have been mapped to the appropriate cores.

Definition 1 (Core Graph (CG)): The core graph, $G(V,E)$ is a directed weighted graph, where each vertex $v_i \in V$ represents a core, and each directed edge $e_{i,j} \in E$ represents communication between v_i and v_j. Every edge has a weight w_{ij} given by:

$$w_{ij} = L_P\,(min\_lat/lat_{ij}) + B_P\,(bw_{ij}/max\_bw) \quad (1)$$

where max_bw is the maximum bandwidth value over all flows in CG, min_lat is the tightest latency constraint over all flows in CG, bw_{ij} is the bandwidth requirement (bits/cycle), lat_{ij} is the latency constraint (cycles), and L_P and B_P are the latency and bandwidth prioritization parameters respectively, such that $L_P + B_P = 1$.

Definition 2 (Floorplan-enhanced Core graph (FCG)): The floorplan-enhanced core graph is similar to the core graph, with the exception that the edge weight from *(1)* is enhanced with information about the distance between cores:

$$\grave{w}_{ij} = L_P(min\_lat/lat_{ij}) + B_P(bw_{ij}/max\_bw) + D_P(min\_dist/dist_{ij}) \quad (2)$$

where min_dist is the minimum Manhattan distance between any two cores, $dist_{ij}$ is the Manhattan distance between v_i and v_j, and D_P is a distance prioritization parameter such that $L_P + B_P + D_P = 1$.

Definition 3 (Link Weight Graph (LWG)): The link weight graph is an undirected graph, $G(S,L)$ with each vertex $s_i \in S$ denotes a switch and the link $l_{ij} \in L$ between s_i and s_j denotes communication from s_i to s_j. The weight of link l_{ij} is calculated as:

$$\grave{\grave{w}}_{ij} = -1 \times \{L_P\,(min\_lat/lat_{ij}) + D_P\,(min\_dist/dist_{ij})\} \quad (3)$$

where prioritization parameters $L_P + D_P = 1$.

Definition 4 (NoC Power Model): The NoC fabric *power* (P_{NoC}) is:

$$P_{NoC} = P_{Link} + P_{Switch} \quad (4)$$

$$P_{Link} = P_{inter\text{-}link} + P_{intra\text{-}link} \quad (5)$$

$$P_{Switch} = P_{static} + P_{dynamic} \quad (6)$$

where P_{Switch} *and* P_{Link} denote the power consumption of switches and links in the NoC, respectively. $P_{inter\text{-}link}$ and $P_{intra\text{-}link}$ denote power consumption of links from switch to switch and from switch to core, respectively. P_{static} denotes the static power due to the static current drawn from the power supply and $P_{dynamic}$ is the dynamic power due to switching activity. P_{static} depends on temperature, and this relation can be expressed as [18]:

$$P_{static} = V_{dd} \cdot I_{leak} \quad (7)$$

$$I_{leak} = I_s \cdot (A \cdot T^2 \cdot e^{((\alpha \cdot Vdd + \beta)/T)} + B \cdot e^{(\gamma \cdot Vdd + \delta)}) \quad (8)$$

where V_{dd} is supply voltage and I_{leak} is average leakage current; I_S is the leakage current for a pre-determined reference temperature and supply voltage, T is the system's operating temperature, and A, B, α, β, γ, and δ are technology dependent constants.

Problem Definition 1 (Non-thermal aware (N-TA) NoC Synthesis): Given a set of k cores $C = \{c_1, c_2, \ldots, c_k\}$, a core graph with inter-core bandwidth and latency constraints, and implementation technology dependent core and NoC area and power models, find the application-specific NoC architecture that satisfies all application performance constraints while minimizing power consumption in *(4)*.

Problem Definition 2 (Thermal aware (TA) NoC Synthesis): Given a set of k cores $C = \{c_1, c_2, \ldots, c_k\}$, a core graph with inter-core bandwidth and latency constraints, and implementation technology dependent core and NoC area and power models and thermal profile, find the application-specific NoC architecture that satisfies all application performance constraints while minimizing peak temperature of the chip.

4. POSEIDON Synthesis Framework

In this section, we describe the POSEIDON application-specific NoC synthesis framework. Figure 1 shows the overall design flow, with figure 1(a) showing the non-thermal-aware (N-TA) synthesis approach, and figure 1(b) showing the thermal-aware (TA) synthesis approach. The N-TA flow has the following inputs: an application core graph, a core description file with technology dependent area and power consumption, and a NoC description file with technology dependent area and power consumption for links and switches. First, a high level floorplanner is used to generate a compact floorplan for the cores on a die. Next, an opportunistic grouping algorithm is used to create clusters of cores that will share switches. Subsequently, a heuristic is used to obtain switch positions and integer linear programming (ILP) is used for determining NI positions. This is followed by a Rectilinear Minimum Weight Spanning Tree (RMWST) based approach to determine link placement. Finally, the optimization phase explores link sizing as well as variations in core grouping to obtain the best synthesized NoC solution. The TA synthesis flow is similar to the N-TA flow except that the floorplanning phase is now thermal-aware, and the switch placement heuristics incorporate thermal optimization in their constraint set. In the following sections, we elaborate on the various phases of the design flow in more detail.

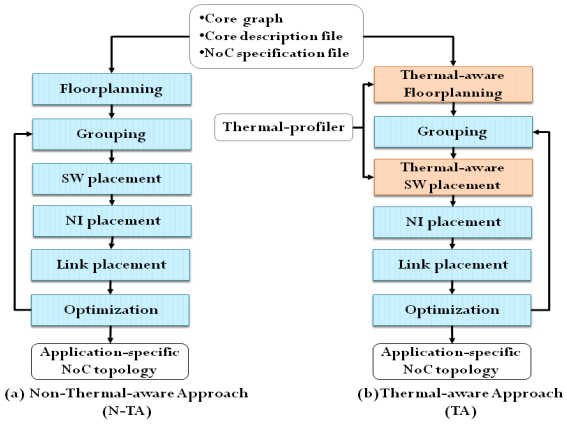

Figure 1: POSEIDON synthesis framework

4.1. Floorplanning

4.1.1 N-TA Floorplanning

The floorplanning phase determines the layout and position of cores on the die. The goal is to obtain a compact layout while ensuring that cores with stringent communication requirements are placed closer together. Placing highly communicating cores closer to each other reduces communication power consumption. At the

same time, cores with tighter latency constraints also need to be placed closer to each other to minimize the number of clock cycles for inter-core transfers. As it is harder to meet latency constraints between two cores that are placed farther away, than to meet bandwidth constraints, we ensure that the value for L_P is larger than B_P in *(1)*, which is the input to the floorplanning phase, along with areas of each of the cores. We make use of a simulated annealing (SA) based floorplanning approach and use CBL [19] to represent every floorplan generated. The SA cost function is:

$$\Psi_{N\text{-}TA} = \lambda_a A + \lambda_c C \qquad (9)$$

where A is floorplan area and $C = \sum w_{ij}^{`}$ represents the sum of communication weights between cores from *(2)*, which are updated at every iteration during floorplanning. The parameters λ_a and λ_c adjust relative weighting between the contributing factors.

4.1.2 TA Floorplanning

The TA floorplanning phase enhances the N-TA floorplanner with thermal awareness during core placement. Cores in CMPs can have a variety of power densities, which governs the thermal profile on the die. If cores with higher power densities are placed close together, lateral heat transfer leads to higher collective temperature than the individual core temperatures. Regions of high temperature lead to hotspots on the die that reduce performance and may also lead to catastrophic failure. The TA floorplanning phase thus attempts to reduce the likelihood of such hotspots by avoiding placing cores with higher power densities close to each other, while simultaneously attempting to generate a compact floorplan that satisfies communication constraints. The simulated annealing N-TA floorplanner is enhanced with the addition of a thermal profiler [20] that can generate the temperature die map for a floorplan iteration, and the cost function is modified as:

$$\Psi_{TA} = \lambda_a A + \lambda_c C + \lambda_t T_p \qquad (10)$$

where T_p is the peak temperature calculated by the thermal profiler, and parameters λ_a, λ_c, and λ_t can be used to adjust relative weighting between the contributing factors.

4.2. Grouping

The goal of this phase is to cluster cores with high bandwidth requirements and tight latency constraints together to reduce communication power and also meet performance constraints. We make use of an opportunistic grouping algorithm that creates clusters with a dedicated switch assigned to them. Algorithm 1 below describes our grouping approach in more detail.

Algorithm 1. Grouping

input: FCG(V, E), σ_{SW} , m groups
output : solution with m groups
1: create list L of rank ordered edge weights in non-increasing order
2: **for** $i = 1$ to $|V(FCG)|$ **do**
3: create group $G_i = \{v_i\}$
4: **end for**
5: $\gamma = |V(FCG)|$
6: **while** $\gamma \neq m$ **do**
7: remove $w_{ij}^{`}$ from top of list L and select cores v_i, v_j
8: //check the group G_i of v_i and G_j of v_j
9: **if** $\{(G_i \neq G_j)$ && $(SW\_size(G_i + G_j) < \sigma_{SW})\}$
10: merge G_i and G_j
11: $\gamma = \gamma - 1$
12: **end if**
13: **end while**

The input to the grouping algorithm is the floorplan-enhanced core graph (*FCG*), a switch size constraint (σ_{SW}), and a grouping constraint (m). The floorplan-enhanced core graph (*FCG*) is created from the core graph (*CG*) by updating edge weights with inter-core distances as described in *(2)*, using the floorplan generated in the previous phase. The switch size constraint (σ_{SW}) determines the maximum number of (input or output) ports a switch can have, based on a target frequency selected by the designer for the NoC. This constraint reflects the practical limitations of synthesizing NoC switches using commercial tools [21][22], which lead to switches with lower maximum clock frequencies as switch complexity increases. Finally, we assume that the maximum number of NoC switches in the synthesized solution must be less than or equal to n, where n is the total number of cores $|V(FCG)|$. The grouping phase clusters cores together such that every group has a dedicated switch allocated to it. The grouping constraint (m) value lies between 1 and n, and specifies the number of groups to be created in the solution generated by the grouping algorithm. The grouping phase is invoked n times during a synthesis run, to create solutions that have 1, 2, …, n groups.

In the algorithm, as the first step, the edges in the *FCG* are rank ordered based on their weights, in non-increasing order to create a list L (line 1). Then each of the cores is initialized to be in its own group, creating a total of n groups (lines 2-4). Subsequently, we perform pairwise grouping of cores, starting with the cores connected by the edge at the top of sorted list L (lines 5-12). The grouping continues till we obtain the desired number of groups (m), at which point the algorithm exits. Although our algorithm may sometimes create groups that are not balanced relative to each other (in contrast to a min-cut partitioning approach [9]), it does a better job of placing cores with critical latency and bandwidth requirements in the same group.

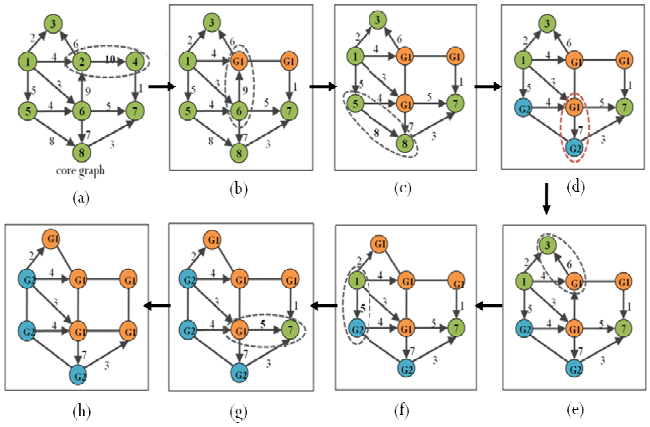

Figure 2: Grouping phase example

Figure 2 shows an example of how the grouping algorithm works for an *FCG* with 8 cores, σ_{SW} constraint of 5, and target group size of 2 (i.e., $m=2$). The opportunistic grouping proceeds from the *FCG* in (a) by selecting the pair of cores v_2 and v_4 that have the maximum edge weight of 10, and combining them into a new group (*G1* in (b)). The grouping process continues till we attempt to combine v_6 and v_8 in (d). This grouping is disallowed as it would lead to a prohibitive σ_{SW} size of 6 for the resulting group. Ultimately, we exit after obtaining the two groups, as shown in (h).

4.3 Switch Placement

In this section, we describe the N-TA and TA approaches for switch placement. Both techniques assume that switches can only be placed at the intersection of cores in the floorplan.

4.3.1 N-TA Switch Placement

In this phase, our goal is to find switch positions that minimize core-to-switch and switch-to-switch link length, to reduce packet latency and power consumption. Algorithm 2 describes switch placement in our framework. First, all groups are sorted in a non-ascending order of total communication passing through their

assigned switch (line 1). Then groups are iteratively removed from the top of this sorted list (lines 2-3) and processed to determine their switch location on the floorplan. For a group under consideration (subsequently referred to as the *home group*), first the accumulated distance for each intersection (stored in $inter\_list_i()$ for group G_i) is calculated (lines 4-7). The accumulated distance of an intersection in the home group is the sum of the Manhattan distances from the center of the cores in the home group to the intersection (intra-group distances) and from the center of the cores in other groups that communicate with cores in the home group to the intersection (inter-group distances). If the switch location for a group that communicates with the home group has already been determined in a previous iteration, we make use of distances from that switch to the intersection, instead of from the center of cores in the group. Then the intersection with the minimum accumulated distance in the home group is selected as the location of the switch for the home group (line 8). The whole process repeats for other groups in the list until switch locations for all groups have been fixed. Sorting the groups based on their communication volume allows more critical groups to fix their switch locations first, improving result quality based on our analysis.

Algorithm 2. Non-thermal-aware SW placement

input : m groups, floorplan
output : m switch positions
1: create list L of groups sorted in non-ascending order of total comm.
2: **while** (list $L \neq \{\emptyset\}$) **do**
3 : remove G_i from top of list L
4: create candidate list $inter\_list_i()$ containing the intersections of G_i
5: **for** each candidate $inter_k$ in $inter\_list_i()$
6: calculate accumulated distance of $inter_k$
7: **end for**
8: place $switch_i$ at $inter_k$ with min accumulated distance
9: **end while**

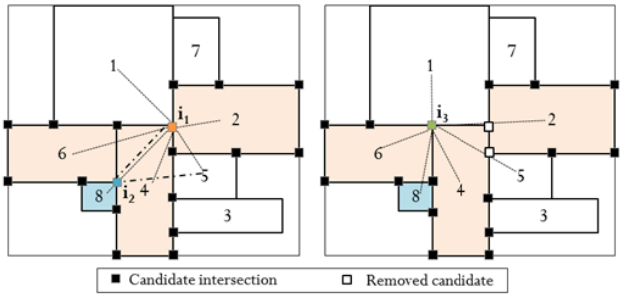

Figure 3: Switch placement (a) N-TA, (b) TA

Figure 3 (a) shows an example of how Algorithm 2 works. Let group *G1* consist of cores 2, 4, and 6 and group *G2* consist of core 8. As group *G1*'s communication volume is higher than that of group *G2*, we first consider placing the switch for *G1*. We initially create an intersection list $inter\_list_{G1}()$ consisting of the 18 core-to-core intersections in *G1*. Assuming that cores in *G1* communicate with cores 1, 5, and 8, distances from cores 1, 5, and 8 as well as from cores 2, 4, and 6 to intersection i_1 are added together to obtain the accumulated distance for i_1. We calculate accumulated distances for every intersection in $inter\_list_{G1}()$ and then place the switch at intersection i_1, which has the least accumulated distance. Next, we attempt to find the switch location for group *G2*. The intersection list $inter\_list_{G2}()$ has 3 core-to-core intersections in *G2*. Assuming that core 8 communicates with group *G1* and core 5, distances from switch *S1* and core 8 to intersection i_2 are added together to obtain the accumulated distance for i_2. After calculating accumulated distances for all 3 intersections in *G2*, we find intersection i_2 as the one with the minimum accumulated distance, and therefore i_2 becomes the location of switch *S2* for group *G2*.

4.3.2 TA Switch Placement

In this phase, our goal is to find switch positions that not only minimize (5), but also to avoid placing switches at intersections that may be located close to hotspots on the die. A thermal profiler is used to obtain peak temperatures for the cores on the die. Cores with power densities exceeding a threshold ϕ are designated as hot cores (C_{Hot}). If the number of such hot cores adjacent to an intersection exceeds a threshold σ_{Hot}, it indicates that the intersection is located in a potential hotspot. Algorithm 3 describes the thermal-aware switch allocation. The procedure in lines 1-6 is similar to Algorithm 2. Subsequently, we now determine the number of C_{Hot} adjacent to the intersection under consideration (line 7) and compare it with σ_{Hot} (line 8). If the σ_{Hot} threshold is exceeded, the intersection is removed from the list of candidates in $inter\_list_i()$ for the switch location. After this pruning step, and calculating accumulated distances for valid intersections, we place the switch at the intersection with the minimum accumulated distance. Figure 3(b) shows an example of TA switch placement, for the same application and set of assumptions as described for figure 3 (a). Cores 2 and 4 are designated as C_{Hot}, and the value of σ_{Hot} is set to 1. After analyzing the intersections in $inter\_list_i()$ *for G1,* we remove two of the candidate intersection locations (white boxes) and place the switch at location i_3 which has the minimum accumulated distance out of the remaining 16 candidates.

Algorithm 3. Thermal-aware SW placement

input : m groups, floorplan, thermal profile, σ_{Hot}
output : m switches positions
1: create list L of groups sorted in non-ascending order of total comm.
2: **while** (list $L \neq \{\emptyset\}$) **do**
3 : remove G_i from top of list L
4: create candidate list $inter\_list_i()$ containing the intersections of G_i
5: **for** each candidate $inter_k$ in $inter\_list_i()$
6: calculate accumulated distance of $inter_k$
7: count the number of adjacent C_{HOT} of $inter_k$
8: **if** (the number of adjacent C_{HOT} for $inter_k > \sigma_{Hot}$)
9: remove $inter_k$ from $inter\_list_i()$
10: **end if**
11: **end for**
12: place $switch_i$ at $inter_k$ with min accumulated distance
13: **end while**

4.4. NI placement

Once all the switch locations have been fixed, we focus on placing the NIs for every core. The NI is allowed to be placed anywhere along the periphery of a core in our approach. For a core c_i with width w and height h, let $(x0_i, y0_i)$ represent the bottom left coordinate, (xs_i, ys_i) be the location of the *switch* in the group to which core c_i belongs, we would like to find the location of c_i's NI (xn_i, yn_i), subject to the following:

$$\text{Minimize } (|xs_i - xn_i| + |ys_i - yn_i|) \quad (11)$$

$$\text{subject to } x0_i \le xn_i \;\&\&\; xn_i \le (x0_i + w),$$
$$y0_i \le yn_i \;\&\&\; yn_i \le (y0_i + h) \quad (12)$$

To find the solution to this optimization problem, we make use of integer linear programming [23], which allows us to determine the NI location for each core in the application. Note that while NI placement can also performed in a thermal aware manner, we choose not to because of the severe performance overheads and insignificant improvements in the chip thermal profile (switches in contrast have much higher power density and thus more suitable for thermal aware placement optimization). Knowing the switch and NI locations on the floorplan, the next section describes how to

perform link placement to connect these components and support communication between them.

4.5. Link Placement

In this phase we aim to find a path between cores through the switches and NIs that satisfies latency constraints and avoids deadlock. For this purpose, we adopt a Rectilinear Minimum Weight Spanning Tree (RMWST) approach [24] for cycle-free link placement that is deadlock-free. First we create a link weight graph (*LWG*) as described in Section 3, and assign larger values for L_P than D_P to ensure shorter paths between cores with tighter latency constraints. Algorithm 4 describes the *LWG* construction (lines 1-2) using *m* switches as vertices and edges with weights calculated from *(3)*. Once the *LWG* has been created, the shortest path between switches is found using Prim's algorithm [25] (line 3). Figure 4 (a) shows an example of an *LWG* with 7 switches, while figure 4 (b) shows the result of using RMWST on the graph.

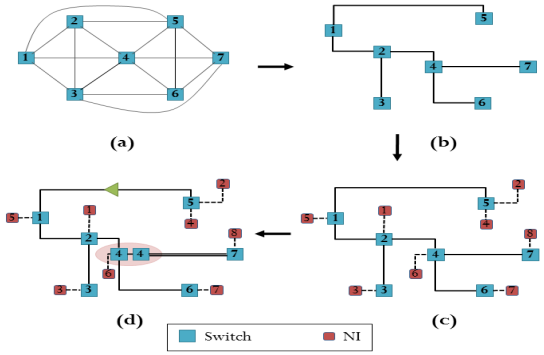

Figure 4: Link placement and optimization

Algorithm 4. Link Placement and Optimization

input : m switch positions, n NI positions, link_freq, σ_{SC}, σ_{SW}
output: optimized application-specific NoC topology
1: establish vertices in *LWG* with *m* switches
2: establish edges in *LWG* and calculate weight using *(3)*
3: find *RMWST* of *LWG* using Prim's algorithm
4: add *n* NI positions as vertices to *LWG*
5: annotate edges with the rectilinear distance of edge as weight
6: **for** all edges l_{ij} in *LWG* **do**
7: **if**(*supp_* bw_{ij} of l_{ij} < bw_{ij})
8: open new port/link and update *supp_*bw_{ij}
9: **end if**
10: **if**(length of l_{ij} > σ_{SC})
11: add buffer to l_{ij}
12: **end if**
13: **end for**
14: **for** all vertex s_i in *LWG* **do**
15: **if**(size of s_i > σ_{SW})
16: split s_i
17: **end if**
18: **end for**

4.6. Post Processing Optimization

During the previous phases we gave higher priority to latency constraints over bandwidth. During post processing optimization, we address this problem by adjusting the *LWG*. Algorithm 4 describes the optimization procedure in more detail. After creating the RMWST to connect vertices in the *LWG*, we enhance the *LWG* further by adding *n* NI positions as new vertices (line 4), and assigning the rectilinear distance between the vertices as the edge weight (line 5). For every edge l_{ij} in the *LWG*, if the maximum supportable bandwidth *supp_*bw_{ij} is smaller than the bandwidth requirement bw_{ij}, a new port is opened and a new link is added between vertices *i* and *j* (lines 7-9). Next, the length of the link is checked against the single cycle distance threshold σ_{SC} (signal propagation distance in *1/link_freq* time). If the link length is longer than σ_{SC}, a pipeline buffer is added to the middle of the link (lines 10-12). Finally, we check if these modifications have increased any of the switch sizes beyond σ_{SW}, in which case we split the switch into two smaller switches (lines 15-17).

Figure 4 (c)-(d) shows the changes to the *LWG* from figure 4 (b) made by the optimization steps. Figure 4 (c) adds NIs and updates edge weights for the *LWG* in Figure 4 (b). Figure 4(d) shows the result after the optimization steps, with double links added to meet bandwidth requirements between switches 4 and 7, a pipeline buffer inserted to meet the single cycle constraint between switches 5 and 1, and splitting switch 4 to meet the switch size constraint.

5. Results

5.1. Experimental Setup

We parallelized five SPLASH-2 benchmarks (*ocean, fft, radix, cholesky, fmm*) [26] on multiple cores, implementing *ocean* with 13 cores, *fft* with 22 cores, *radix* with 48 cores, *cholesky* with 68 cores and *fmm* with 104 cores, to span the spectrum of low to high complexity CMPs for analyzing the effectiveness of our synthesis framework. We implemented the models in SystemC, using a modified version of the open-source Nirgam [27] NoC simulator. The Hotfloorplan [20] tool was used to obtain the thermal profile of the die, and a modified variant of Parquet [28] to perform detailed floorplanning. NoC power consumption estimates were derived using the Orion 2.0 tool [29]. We set the NoC frequency to 2 GHz and used 45 nm as the implementation technology. Finally, for the thresholds/parameters used in our proposed framework, we assume the following values: σ_{SW} = 6, σ_{Hot} = 2, σ_{SC} = 8 mm; L_P = 0.7, B_P = 0.3 (*CG*); L_P = 0.5, B_P = 0.25, D_P = 0.25 (*FCG*); and L_P = 0.7, D_P = 0.3 (*LWG*). All simulations were run on a Linux workstation with a 2.8GHz Intel Core Duo CPU and 6GB memory.

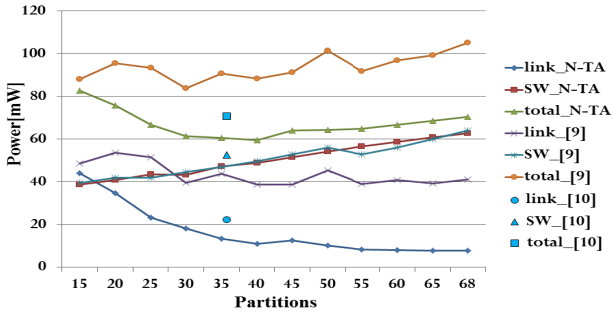

Figure 5: Power consumption of *cholesky*

5.2. Comparison of N-TA approach with previous work

Our first set of experimental results compare the quality of results generated by our N-TA synthesis framework with existing work on application-specific NoC synthesis by Murali et al. [9] and Srinivasan et al. [10]. We used the same benchmarks and power/ performance models on our implementations of [9] and [10] as we did with our N-TA approach, for a fair comparison. Figure 5 shows a set of Pareto design points of the power consumption for solutions generated by [9], [10], and our N-TA approach for the *cholesky* benchmark. Both N-TA and [9] generate a set of solutions depending on the number of allowed partitions (groups) and then select the best solution, while the approach in [10] only generates a single solution. It can be seen that the total power consumptions for the solutions generated by our approach (total_N-TA) are significantly lower than for solutions generated by [9] (total_[9]) or the solution generated by [10] (total_[10]). It is interesting to note that while the switch powers ('SW_') for our N-TA approach and [9] are comparable, the link power ('link_') for N-TA is much

lower than for [9]. This is due to our use of more effective floorplan-aware switch and NI placement techniques.

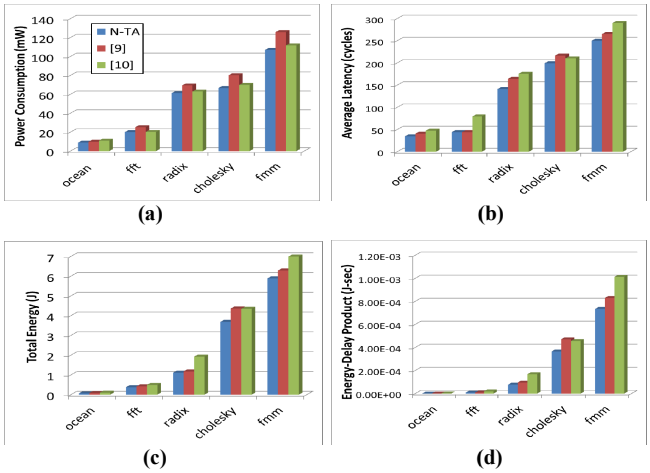

Figure 6: Comparison between N_TA, [9], [10] (a) power consumption, (b) average latency, (c) total energy, (d) energy-delay product

Figure 6 shows the result of NoC power consumption, average latency per packet, total energy, and energy-delay product for the three schemes being compared across the five benchmarks considered. For NoC power consumption, it can be seen that N-TA improves upon [9] by 15.7% and upon [10] by 6.30% on average, primarily as a result of lower link lengths and shorter routes (which translates into lower power), due to more efficient switch and NI placement techniques. In terms of average latency, N-TA improves upon [9] by 8.51% and upon [10] by 21.08% on average. For [9], the lacks of floorplan information during partitioning leads to link lengths that exceed the single cycle region threshold and require pipeline buffers that add to the latency. On the other hand, [10] makes use of a greater number of (simpler) switches, which increases the path latency for packets. In terms of total energy consumption, N-TA improves upon [9] by 12.5% and upon [10] by 27.05% on average, while in terms of energy-delay product, N-TA improves upon [9] by 19.8% and upon [10] by 42.7%. It can thus be seen that our proposed application-specific synthesis approach provides an improvement over state-of-the-art existing techniques on application-specific NoC synthesis.

Table I: Comparison of N-TA and TA approaches

Benchmark	Synthesis Approach	Peak Temp (°C)	Power(mW)			Avg. Latency
			Switch	Link	Total	
ocean	N-TA	79.1	6.29	2.67	8.95	34.6
	TA	75.9	5.87	4.53	10.40	37.74
fft	N-TA	86.8	12.07	6.17	18.24	44.08
	TA	83.2	11.08	8.29	19.37	47.60
radix	N-TA	90.8	32.4	24.89	57.29	141.14
	TA	87.2	30.40	33.43	63.83	152.29
cholesky	N-TA	87.0	45.35	18.10	63.45	199.17
	TA	83.6	41.57	26.04	67.61	219.08
fmm	N-TA	86.6	72.78	34.22	107.0	250.0
	TA	82.8	66.72	45.40	112.1	268.75

5.1.2 Comparison of N-TA with TA approach

Our proposed POSEIDON synthesis framework has the capability to generate solutions that can minimize peak temperature on the die, while satisfying application performance constraints. As existing approaches do not support thermal-aware application-specific NoC synthesis, we compare our thermal-aware (TA) synthesis approach with our non-thermal aware (N-TA) approach in this section, to understand the trade-offs involved with thermal-aware NoC synthesis. Table I shows a comparison of peak temperature, power consumption, and average latency for the N-TA and TA approaches. It can be seen that the TA approach consumes on average 10% more power and has 8.7% higher latency than the N-TA approach. This is because thermal aware floorplanning and switch placement leads to longer link lengths, which leads to greater dynamic power consumption and latencies. However, it is interesting to note that the static (leakage) power in the switches goes down for the TA approach, primarily due to lower on-die temperatures. On average the TA approach reduces the peak temperature by 4.2% compared to the N-TA approach. Figure 7 shows the thermal maps for the *cholesky* benchmark, for the best solutions generated by the TA and N-TA approaches. It can be clearly seen that the number of hotspots has been reduced significantly in the solution generated by the TA approach, due to thermal-aware floorplanning and switch placement. Given that leakage power and hotspots will only increase as technology scales further, the TA approach may be a viable application-specific NoC synthesis option for future technologies.

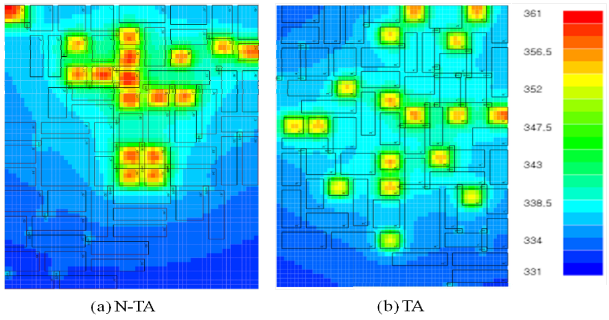

Figure 7: Thermal maps of *cholesky* of N-TA and TA

6. Conclusion

In this paper, we proposed the POSEIDON framework for the application-specific synthesis of NoCs in chip multiprocessor applications. The synthesis framework can be customized to generate either a non-thermal aware (N-TA) or thermal-aware (TA) solution that satisfies application performance constraints while minimizing power and peak temperatures, respectively. Our comparison with existing work on application-specific NoC synthesis showed that our N-TA approach provides a reduction of up to 15.7% in power consumption, 21.08% in average latency, 19.8% in total energy, and 42.7% in energy-delay product, compared to state-of-the-art approaches, as well as a 4.2% reduction in peak temperature when the framework is customized to generate a thermal-aware solution.

References

[1] L. Benini and G. De-Micheli, "Networks on Chip: A New SoC Paradigm," Proc. IEEE Computer, 49(1):70-71, Jan 2002.
[2] S. Vangal et al., "An 80-Tile 1.28 TFLOPS Network-on-Chip in 65 nm CMOS," Proc. IEEE Int'l Solid State Circuits Conf., 2007.
[3] Tilera Corporation. TILE64™ Processor. Product Brief. 2007.
[4] Plurality HAL-256, http://www.plurality.com/products.html, 2009.
[5] S. Murali, G. D. Micheli, "Bandwidth-Constrained Mapping of Cores onto NoC Architectures," Proc. DATE, 2004.
[6] S. Murali, G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs", Proc. DAC 2004.
[7] J. Hu, R. Marculescu, "Energy-Aware Mapping for Tile-based NoC Architectures under Performance Constraints," Proc. ASPDAC 2003.
[8] J. Hu et al. "Exploiting the routing flexibility for energy/ performance aware mapping of regular NoC architecture," Proc. DATE 2003
[9] S. Murali., et al., "Designing application-specific networks on chips with floorplan information", Proc. ICCAD 2006, pp. 355-362.
[10] K. Srinivasan, K. S. Chatha, "A Low Complexity Heuristic for Design of Custom Network-on-Chip Architectures," Proc. DATE 2006

[11] K. Srinivasan, et al., "Linear programming based techniques for synthesis of network-on-chip architectures," IEEE Trans. on VLSI, 2006.
[12] J.Chan, et al., "NoCOUT: NoC Topology Generation with Mixed Packet-switched and Point-to-Point Networks," ASPDAC, 2008.
[13] S. Xu et al., "Thermal Impacts on NoC Interconnects," NOCS 2007.
[14] A. Jalabert, et al., "xpipesCompiler: A tool for instantiating application-specific Networks on Chip," Proc. DATE, 2004.
[15] P. Lim, T. Kim, "Thermal-Aware High-level Synthesis based on Network Flow Method", Proc. CODES+ISSS 2006, pp.124-129
[16] A. Coskun et al., "Temperature Aware Task Scheduling in MPSoCs," Proc. DATE, 2007.
[17] G. M. Link, N. Vijaykrishnan, "Hotspot prevention through runtime reconfiguration in network-on-Chip," Proc. DATE, 2005.
[18] W. Liao, et al, "Microarchitecture level power and thermal simulation considering temperature dependent leakage model," Proc. ISLPED 2003.
[19] X. Hong, S. Dong, "Non-slicing floorplan and placement using corner block list topological representation," IEEE Trans. CAS, 51:228–233, 2004.
[20] Hotspot 5.0, http://lava.cs.virginia.edu/HotSpot/
[21] http://www.synopsys.com/dw/ipdir.php?ds=core_consultant
[22] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli and L. Benini, "Bringing NoCs to 65 nm," IEEE Micro 27(5): 75-85, 2007.
[23] lp_solve 5.5.2.0, http://lpsolve.sourceforge.net/
[24] F. K. Hwang, "On Steiner minimal trees with rectilinear distance", SIAM J. of Applied Mathematics, 30:104–114, 1976.
[25] T. Cormen, et al., Introduction to Algorithms, 3rd ed. MIT Press, 2009.
[26] SPLASH-2, http://kbarr.net/splash2
[27] Nirgam, http://nirgam.ecs.soton.ac.uk/
[28] ParquetFP, http://vlsicad.eecs.umich.edu/BK/PDtools/
[29] Orion 2.0, http://www.princeton.edu/~peh/orion.html