Reliability- and performance-driven mapping for regular 3D NoCs using a novel latency model and Simulated Allocation

Wei Gao\textsuperscript{a,c,d}, Zhiliang Qian\textsuperscript{b}, Pingqiang Zhou\textsuperscript{c,*}

\textsuperscript{a} Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, Peoples R China
\textsuperscript{b} Shanghai JiaoTong University, Shanghai, China
\textsuperscript{c} School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, Peoples R China
\textsuperscript{d} University of Chinese Academy of Sciences, Peoples R China

\textbf{ABSTRACT}

Network-on-Chip (NoC)-based communication architecture is promising in addressing the communication bottlenecks in current and future multicore processors. In this work, we consider the application-specific mapping problem and electromigration (EM)-induced through-silicon via (TSV) reliability issue in tile-based three-dimensional (3D) NoC architectures. In 3D NoCs, network contention may result in unacceptable communication delay among the processing cores and thus has significant effect on the system performance. So we propose a new latency model for the routers which characterizes the network contentions among different traffic flows from sharing of network resources. Then we solve the core mapping problem by a fast while efficient stochastic algorithm called Simulated Allocation (SAL), which integrates our new latency model and also aims to optimize the communication power, latency in the mapping procedure. After that, we use an incremental method to optimize the reliability of the TSVs. Experimental results show that, contention-aware model (CAM) has 25\% larger network latency than our latency model; compared with particle swarm optimization (PSO), our SAL algorithm can achieve 7\% less power with about 7.5\% run-time speedup; and for reliability, our method can achieve better results (up to 100\% increase in terms of the void nucleation time (VNT)) with 7.64\% increased latency.

1. Introduction

In recent years, the Network-on-Chip (NoC) has proven its success in providing a scalable communication scheme for System-on-Chips (SoCs) and general-purpose multicore processors [1]. One big advantage of NoC is that it can support concurrent communication among different source-destination pairs, thus it can improve the communication performance in terms of latency as well as reduce power consumption significantly [2], by avoiding large fan-outs and heavy contentions that happen in the traditional bus scheme.

Three-dimensional (3D) integrated circuits, in which multiple tiers are stacked above each other and vertically interconnected using through-silicon vias (TSVs), are emerging as a promising technology to reduce communication power (CP) and network latency (NL) for multicore chips [3]. However, the TSVs are susceptible to reliability issues, of which the electromigration (EM)-induced failure is a major concern [4].

Regular 3D tile-based NoC architectures such as mesh have been widely used in recent 3D IC design, especially in static mapping problem [5,6]. Due to their modular structure and high scalability, regular 3D tile-based NoCs are more attractive than inhomogeneous 3D NoCs [7].

When designing the regular 3D NoC architecture for a given multicore application, the challenges mainly come from three aspects. First, how to develop an efficient algorithm to map the application to the tile-based regular 3D NoC architecture. It is well known that the mapping problem is NP-Hard [8,9], and the designers need to tackle a very large solution space [10] in the application mapping, but the algorithm is required to converge to a high-quality solution in a fast way. In other words, although the global optimal solution is hard to reach in most problems, the final solution should avoid being trapped in bad local minimums. In addition, the algorithm is often required to incorporate multiple objectives and constraints (e.g., models of power, latency and bandwidth) in the optimization, which further complicates the search process [11].

Second, how to optimize the lifetime of the TSVs that are under the threat of the EM-induced failures. Reliability is an important issue in current integrated circuits [12]. In 3D ICs, the reliability of TSVs has become a new challenge because the vertical TSVs have different characteristics compared to the horizontal metal interconnects [13].

According to the EM models presented in [14], the lifetime of a TSV is affected by both its current density and temperature. The current density in a TSV is determined by the traffic through it, and therefore is affected by the core to tile mapping in the 3D NoCs. Different mappings also lead to different power distribution in the 3D chip and thus affect the TSVs’ temperature values. Therefore, we should consider the reliability issue in the 3D NoC mapping problem, and optimize the lifetime of the TSVs.

* Corresponding author.
E-mail addresses: gaowei@shanghaitech.edu.cn (W. Gao), qianzl@connect.ust.hk (Z. Qian), zhoupq@shanghaitech.edu.cn (P. Zhou).

https://doi.org/10.1016/j.vlsi.2018.04.012
Received 7 September 2017; Received in revised form 12 March 2018; Accepted 20 April 2018
Available online 9 May 2018
0167-9260/© 2018 Elsevier B.V. All rights reserved.
Finally, how to model the performance metrics in NoC. In addition to reliability, performance (latency or bandwidth) is the main concern in exploring the regular NoC architectures [11]. The mapping procedure needs a large number of iterations, thus in general, the latency model should be simple but still reflect certain non-linear effects (such as the impacts of traffic contentions on the NL). Therefore, a fast but accurate latency model is required in the mapping process [15].

In this work, we address the aforementioned challenges, and have made the following contributions:

- We develop a novel mapping algorithm based on the stochastic SAL approach, and it can easily incorporate optimization objectives such as CP and NL in the mapping procedure. Our experimental results show that the proposed algorithm achieves better mapping results as well as faster convergence speed compared to recent work [5].
- Accompanying SAL, we propose a new delay model for the routers, which is able to characterize the contention effects from resource sharing, and to reflect the effect of traffic amount on router and link latency. Our model uses two separate queues to model one router and it can be easily embedded into the objective or constraint functions when exploring the solution space in NoC-based multicore mapping. Compared to the simple hop-count model and contention-aware model, our model can achieve better NL (hop-count is 17% worse than SAL, CAM is 25% worse than SAL).
- After the SAL mapping procedure, we propose an incremental updating approach to tackle the EM-induced reliability issue in TSVs, and attempt to optimize the lifetime of the TSVs in the 3D NoC mapping process. Results show that the bottleneck TSVs’ lifetime can achieve up to $10 \times$ improvement.

The rest of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we provide a formal description of the mapping problem. We describe our reliability optimization method in Section 4, then present our new latency model in Section 5. Our SAL-based mapping algorithm is presented in Section 6. Finally we state our experimental results in Section 7.

2. Related work

The prior work on 2D and 3D NoC-based mapping can be classified into two categories—mathematical programming [16,17] and heuristic approaches [18]. The mathematical programming approaches try to formulate the mapping problem as either an Linear Programming [17] or an Integer Linear Programming (ILP) [16] problem. Although this kind of approaches can achieve a theoretical optimal solution to the formulated programming problem, they have two main limitations. First, it usually takes a long time to solve the mathematical programming problems, especially when the chip has a large number of cores/tiles. Second, in order to make the problem solvable, the programming formulation tends to adopt simple performance models. For instance, the hop-count-based latency model is widely used in the objective or the constraint functions in order to achieve a linear formulation [11].

However, the hop-count model fails to incorporate the sharing of network resources and network contention in NoC, thus the obtained mapping solution may suffer serious congestion under heavy workloads. Therefore, Chou et al. [16] propose a modified ILP model that considers the secondary contention effects. Their results show that contention-aware mapping can achieve better solutions than hop-count based models. Motivated by this observation, in this work, we develop a latency model to characterize the NL that takes both the sharing of network resources and network contention into account. Different from [16] where the contentsions are implicitly estimated and latency is only a function of travel distance, our work uses two queues for a router to explicitly model the contention effects from resource sharing, and to reflect the effect of the traffic amount on router and link latency. Compared to more complicated model [15] that further includes link dependency analysis, our model has less computational complexity and is thus more suitable to the NoC mapping procedure which involves a large number of iterations, each requires to evaluate the model once to obtain the latency result.

Due to the limitation of the mathematical methods, heuristic approaches such as genetic algorithm [19,20], simulated annealing [21], Branch-and-Bound methods [18] and particle swarm optimization [5], have been developed to explore the large solution space of a many core chip. For the same reason as in mathematical programming, most of these algorithms use simple hop-count model for performance evaluation. In this work, we propose to solve the mapping problem by the stochastic Simulation Allocation (SAL) algorithm [22], which has shown to be faster while more efficient than other stochastic algorithms such as simulated annealing in solving the multicommodity traffic flow problems [23] and 3D NoC topology synthesis problem [24].

The prior work [6] proposes a methodology for TSV placement (number and position) in 3D NoC mapping, and it uses commodity and hop-count in the objective function. There also exist several works in the literature that researched the reliability issue in 2D or 3D NoCs [13,25,26]. Wu et al. [25] proposes a mapping approach that considers the reliability issue in the links of 2D NoC architecture, and it uses a probability model to estimate the occurrence probability of fault links. Khayambashi et al. [13] proposes a method to estimate the probability of system failure. The results of [26] show that the square shaped topology of the network has better reliability because square shape has more candidate paths when adaptive routing is used. Our work differs from these prior works in that we use an accurate reliability model for TSVs (see Section 4) and try to improve their reliability in the 3D NoC mapping process.

3. Problem formulation

To formulate the mapping problem, we need the following definitions.

Definition 1. Core graph: A core graph is a directed graph $G(V, E)$, where each node $v_i \in V$ represents a core (either an IP block or a memory unit) and each directed edge $e_{ij}$ denotes the communication (i.e, traffic flow) from core $v_i$ to core $v_j$. The weight of edge $e_{ij}$, $w_{ij}$, gives the minimum traffic rate requirement (in MB/s) from core $v_i$ to core $v_j$.

Definition 2. Tile-based 3D NoC architecture: A tile-based architecture is a $M \times N \times Q$ regular grid structure with M rows, N columns and Q layers, where each grid at location $(m, n, q)$ (Here $m \in [1, 2, \ldots, M], n \in [1, 2, \ldots, N], q \in [1, 2, \ldots, Q]$) is a tile that can be mapped to a processing core or a memory core. Each tile consists of a core and a router, and all the routers in one layer are connected by links and routers in different layers are connected by TSV bundles, therefore form a 3D mesh NoC network. The number of TSVs in each TSV bundle between routers at two adjacent layers equals to the number of bits in one flit.

Fig. 1(a) and Fig. 1(b) respectively show an example of the core graph and a title-based 3D NoC regular architecture. Note that there are two TSV bundles between tile 2 and tile 8: One TSV bundle is from tile 2 to tile 8, another is from tile 8 to tile 2.

Simply speaking, given an application represented as a core graph and the corresponding tile-based 3D architecture, the mapping problem tries to find a one-to-one mapping between the cores in the core graph and the tiles in tile-based NoC architecture where $|V| \leq M \times N \times Q$, with the constraint that the traffic rate $\lambda_{xt}$ in the given core graph can be supported by the available bandwidth in the communication network after mapping. In our work, we assume Z-Y-X routing is adopted in the NoC network. Fig. 1 shows an example with 12 cores mapped to 12 tiles. According to [8], the mapping problem is NP-Hard.

In our work, we solve the 3D NoC mapping problem in two steps. First, we apply our SAL-based mapping methodology (see Section 6) to find the best mapping solution with CP and NL as the objectives. In our implementation, we choose to optimize a weighted sum of the CP and NL, while meeting the communication requirements $\lambda_{xt,s}$ on the NoC.
architecture. More specifically, our objective function is given by

\[ f = c \cdot \rho + \beta \cdot \exp(\frac{E_a}{k_i \cdot T_{tmp}}) \]  

where \( c \) represents the location in a metal conductor, \( k \) is the diffusivity, and \( D_0 \) is the effective diffusion coefficient:

\[ D_x = D_0 \exp\left(\frac{E_a}{k_i \cdot T_{tmp}}\right) \]  

The positive stress will be developed at cathode node of the metal conductor (x = 0) and negative stress will be developed at anode node (x = 1). If the largest stress at cathode node (x = 0) \( \sigma_{EM}^c \) exceeds critical stress \( \sigma_{crit} \), then the void will be created, and the void nucleation time (VNT), \( t_{nuc} \), can be calculated as [14]

\[ t_{nuc} = \frac{L^2 k_i T_{tmp}}{D_j \cdot \beta} \ln \left( \frac{\sigma_{EM}}{\sigma_{EM}^c - \sigma_{crit}} \right) \]  

where \( \sigma_{EM}^c = \frac{\rho \cdot j \cdot \zeta \cdot L}{\Omega} \)

In our work, we use the model (5) to compute the lifetime of a TSV. From equation (5), the VNT of one TSV bundle is affected by its TSV temperature (\( T_{tmp} \)) and current density (\( j \)).

4.2. Incremental mapping updating for reliability improvement

Fig. 2 shows the VNT distribution of the TSVs in benchmark b1 obtained by the reliability model presented in Section 4.1. Note that in our work, we assume that the TSVs in one bundle have the same probability of sending 0 and 1 signals, and therefore have the same current density and temperature at the cathode node of TSVs (x = 0). In other words, all the TSVs in one bundle have the same VNT value. From Fig. 2 we can see that the VNT distribution is not even among different TSV bundles. Some TSV bundles with large traffic amount and high temperature (such as the 8-th TSV) have small VNTs and they become the bottleneck of the whole 3D chip. Therefore, in our work, we propose an incremental mapping

### Table 1

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho )</td>
<td>Electrical resistivity</td>
<td>1.67e-8 ( \Omega \cdot m )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>Electric charge</td>
<td>1.6e-19 C</td>
</tr>
<tr>
<td>( \zeta )</td>
<td>Effective charge</td>
<td>4</td>
</tr>
<tr>
<td>( \Omega )</td>
<td>Atomic volume</td>
<td>1.6e-29 m³</td>
</tr>
<tr>
<td>( k )</td>
<td>Boltzmann constant</td>
<td>1.38e-23 J/K</td>
</tr>
<tr>
<td>( B )</td>
<td>Back flow stress modular</td>
<td>1e11 Pa</td>
</tr>
<tr>
<td>( D_0 )</td>
<td>Pre exponential factor of diffusion</td>
<td>1e-8 m²/s</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>Activation energy</td>
<td>0.8 eV</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Thermal stress</td>
<td>400 MPa</td>
</tr>
<tr>
<td>( j )</td>
<td>Average current density</td>
<td>( j )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Length of TSV</td>
<td>2e-5 m</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Absolute temperature</td>
<td>( \gamma )</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>Electromigration stress</td>
<td>( \gamma )</td>
</tr>
</tbody>
</table>

---

Fig. 2. VNT distribution of the TSV bundles in benchmark b1.
updating method to improve the reliability of such bottleneck TSV bundles.

Our idea is motivated by the following two observations:

- First, according to Equation (5), the VNT of a TSV is dependent on its temperature and current density, which are both affected by the mapping of the cores in the 3D tile architecture.
- Second, in a 3D chip as shown in Fig. 3, in the vertical direction, the whole chip temperature tends to be reduced if we move a core with high power density closer to the heat sink layer. Besides, horizontally, in the same layer, if we want to reduce the temperature at a certain spot, we can reduce the power density of its surrounding cores. Therefore, if we update the core mappings, we can swap the cores in the 3D tile architecture, and thus optimize the temperature profile of the 3D chip.

Therefore, as stated in Section 3, after the SAL-based core mapping step (see Section 4), we add another step to improve the VNT of the bottleneck TSV bundles in the 3D chip, by incrementally updating the mapping solutions obtained by SAL.

Algorithm 1

```
1 Input: a core graph \( G(V, E) \) and a tile-based architecture, the current mapping solution, \( H \);
2 Output: the mapping solution;
3 \( T_{min} \); T;
4 Find the TSV bundle with smallest VNT, \( T \);
5 \( T_{min} = T \);
6 Find the adjacent core set \( AC \) with the TSV;
7 for travel all feasible solution do
8 Initialize a sign vector \( S = \{0 \ldots 0 \} \) has same size as core set \( V \):
9 for \( g=1; g \leq \text{num of AC}; g++ \) do
10 for \( r=1; r \leq \text{num of } V; r++ \) do
11 if \( S[r] = 0 \) && \( g \neq r \) then
12 if JUDGE_FEASIBLE_SOLUTION(\( c_g \), \( r \)) then
13 Swap core \( c_g \) with core \( r \);
14 \( S[r] = 1 \);
15 break;
16 end
17 end
18 end
19 if latency of new solution \( X \) is smaller than 1.1 * \( H \) then
20 if least VNT of TSVs in new solution \( X \), \( T \), is bigger than \( T_{min} \) then
21 \( T_{min} = T \);
22 \( T_{min} = T \);
23 Save the new solution \( X \);
24 end
25 end
26 end
```

Procedure JUDGE_FEASIBLE_SOLUTION(core \( c_g \), core \( r \)):

\[
G = \text{layer of tile AC}[g];
R = \text{layer of tile which core } r \text{ mapped to};
\]

if power of core \( r \) is smaller than power of core \( c_g \) then
   Return true;
else
   if power of core \( r \) is larger than power of core \( c_g \) then
      Return true;

Fig. 3. A simple example of three layers 3D chip with heat sink.

Given the mapping solution (with cost \( H \)) obtained by the SAL step, we pick the TSV bundle with smallest VNT in the 3D chip, and then call Algorithm 1 to improve this bottleneck TSV bundle's VNT value. Since Algorithm 1 updates the core mappings, we will get another VNT distribution of TSV bundles after this step. Then we pick the TSV bundle with smallest VNT in the new VNT distribution and repeat the above procedure until no more improvement can be made on the VNT of the bottleneck TSV bundle in the latest mapping solution.

Algorithm 1 takes the core graph \( G(V, E) \), the 3D tile-based architecture, and the current core mapping solution as the inputs. In this algorithm, we first find the TSV bundle with smallest VNT (line 4) and assign \( T \) to the current min VNT \( T_{min} \) (line 5). Then we find all adjacent cores to this TSV bundle (line 6). For example, for the TSV bundle from router 13 to router 22 in Fig. 3, the cathode node of TSV bundle is connected to router 13 and the corresponding core is mapped to tile 13. The adjacent core set \( AC \) of this bundle includes the cores which are mapped to the tiles 12, 14, 16, 10. Then we travel all the feasible solutions that can reduce this TSV bundle's temperature after swapping each core mapped to tiles in set \( AC \) with other cores in the 3D chip (lines 7–26). For each feasible solution we initialize a sign vector \( S \) that has the same size as core set \( V \) (line 8). \( S[r] = 0 \) represents that core \( r \) is not swapped with any cores mapped to tiles in \( AC \). Each core \( c_g \) mapped to tile \( AC[g] \) (1, 2, ..., \( g \), ..., number of AC) in \( AC \) set. If core \( r \) in \( V \) is not swapped and core \( c_r \) is different from core \( r \) (line 11). Then we call the Judge Feasible Solution to determine which core can be swapped (lines 28–38). Then we swap core \( c_r \) with core \( r \) and set the sign \( S[r] \) to 1 (lines 13–14). If the new solution has better least VNT and latency increase less than 10%\( H \), we should save the new solution (lines 21–25). The 10% is the appropriate value verified by experiments.

To calculate every TSV bundle's VNT value in each new solution (line 21), we should update the temperature and current density of the TSV bundles. To calculate the TSV's temperature, we use the following equation

\[
\text{TMP} = \text{C} \times \text{P} 
\]

where \( \text{TMP} \) is the temperature vector of all the TSV bundles, \( \text{C} \) is the coefficient matrix which can be obtained by following the method proposed in [28], and \( \text{P} \) is the vector of the power consumption of the tiles in the 3D chip. To calculate the current density of each TSV bundle, we use the following formula
where $T_{TSV}$ is the time from the header flit leaves Router3 to the tail flit of the same packet leaves Router3. The formula to compute $T_{TSV}$ will be given in Section 5.2.

In our work, we use the queuing theory to develop the latency models for the routers and links respectively in Section 5.2 and Section 5.3. We define the following notations used in the rest of this section:

- $\lambda$ – the average arrival rate of the customers in one queue,
- $C$ – the service rate of a single server,
- $\eta$ – the traffic utilization of a single server,
- $L_q$ – the expected total number of customers in the queue,
- $p_K$ – the probability of having $K$ customers in the queuing system,
- $T_w$ – the waiting time of one customer,
- $T_s$ – the service time of one customer.

### 5.2. Latency model for single-channel routers

In our work, we consider a classic router architecture as shown in Fig. 5. When the header flit of a packet arriving at an input port of the router, it will go through the following processing steps [29]: buffered in the input buffer (BW), route computation (RC), arbitration for the input and output ports of the crossbar (SA), traveling through the crossbar and being placed on the output link connected to the next router node (ST). We can divide all the aforementioned processing steps into two sequential parts, as illustrated in Fig. 6: Input queue (IQ) and Output queue (OQ).

- **Input queue (IQ):** This includes the BW and RC steps. The delay in this queue comes from the sharing of the route computation and the finite buffer size that causes the waiting time before the flits entering the buffer. We assumed that every finite input buffer can hold $K$ flits and each packet has $B$ flits. Due to the finite buffer size ($K$ flits), we model the BW process of input port as a $M/M/1/K$ queue. Then according to [30], the average BW queue delay for one header flit is

$$T_{BW} = \frac{1}{C_{BW}} + \frac{L_q}{\lambda_{BW}(1 - p_K)}$$

where $\lambda_{BW}$ (flits/cycle) is the sum of all the flow’s arrival rate at the input port which measures the resource sharing from the traffic amount in input buffer, $C_{BW}$ is the service rate of an input buffer that equals to one flit per cycle [29],

$$\eta = \frac{\lambda_{BW}}{C_{BW}} \quad (\eta < 1),$$

$$L_q = \frac{\eta}{1 - \eta} - \frac{\eta(K\eta^K + 1)}{1 - \eta^{K+1}},$$

and

$$p_K = \frac{\eta^K(1 - \eta)}{1 - \eta^{K+1}}$$

In addition to the BW process, there is additional delay time of the RC process for the header flit, thus

$$T_{RC} = T_{BW} + t_{RC}$$

where $t_{RC}$ is the constant time for the RC step which equals to one cycle per header flit [29].

Our $M/M/1/K$ queues for the input buffers not only consider the service time in the input buffers but also the waiting time caused by the finite buffer sizes at the input ports.
\[ T_{SA} = \frac{1}{C_{SA}} \]  
(15)

As shown in Fig. 5, two traffic flows, one from Router 1 to Router 4 with rate of \( \lambda_{14} \) flits/cycle, another from Router 2 to Router 4 with rate of \( \lambda_{24} \) flits/cycle, enter two different input ports of Router 3, but come out from the same output port. Thus in order to characterize the degree of contention quantitatively, the total arrival rate of the SA queue at output port 1 of Router 3 from these two traffic flows is calculated as

\[ \lambda_{SA} = \lambda_{14} + \lambda_{24} \]  
(16)

Note that we need to consider one packet as a whole when computing the waiting time in this SA step: If two packets \( P \) and \( Q \) come from different inputs need to enter the same output simultaneously, their header flits will go through an arbitration process and the header flit of one packet (say \( P \)) will gain the access to switch crossbar. Then all the flits of packet \( Q \) have to wait for next SA process until all the flits of packet \( P \) have passed the ST queue.

According to [30], the average SA queue delay for one header flit is

\[ T_{SA}^{\lambda} = \frac{1}{2} \frac{\eta_{SA}}{C_{SA} - \lambda_{SA}} \]  
(17)

where \( \lambda_{SA} \) (flits/cycle) is the arrival rate, and \( C_{SA} \) is the service rate of the SA queue, which equals to one flit per cycle [29].

\[ \eta_{SA} = \frac{\lambda_{SA}}{C_{SA}} \quad (\eta_{SA} < 1), \]  
(18)

Then we have

\[ T_{SA} = T_{SA}^{\lambda} + T_{SA}^{\eta} \]  
(19)

The delay for the header flit in ST stage is one cycle per flit [29], i.e., \( t_{ST} = 1 \), thus the delay of the output queue OQ is

\[ T_{OQ} = T_{SA} + t_{ST} \]  
(20)

and

\[ T_{o} = (B - 1)t_{ST} \]  
(21)

In summary, the total delay of one header flit travelling through one router can be modeled as

\[ T_{router} = T_{OQ} + T_{OQ} \]  
(22)

5.3. Latency model for links

In NoC, each link may be shared by several flows in the core graph. Fig. 7 shows that link \( L_{1,2} \) is shared by flow1 and flow2 with traffic rates of \( \lambda_{12} \) and \( \lambda_{21} \). Then the total traffic rate through link \( L_{1,2} \) which models resource sharing by traffic amount is

\[ \lambda_{12} = \lambda_{12} + \lambda_{21} \]  
(23)

Let \( C_{12} \) be the bandwidth of the link \( L_{1,2} \), which is determined by the link width and clock frequency, then the link utilization is

\[ \eta_{link} = \frac{\lambda_{link}}{C_{12}} \quad (\eta_{link} < 1) \]  
(24)

We can model the link \( L_{1,2} \) as a single \( M/D/1 \) queue, and according to [30], the link delay can be expressed as

\[ T_{link}^{\lambda} = \frac{1}{2} \frac{\eta_{link}}{C_{12}} + \frac{1}{C_{12}} \]  
(25)

Model (25) characterizes the sharing of the physical link by different traffic flows.

6. Sal-based mapping methodology

In this work, we adapted the SAL algorithm [23] to solve the performance- and power-driven mapping problem formulated in Section 3. Simulation Allocation (SAL) is a stochastic approach that has shown to be faster and more efficient than other stochastic algorithms such as simulated annealing, in solving the multi-commodity traffic flow problems [23] and 3D NoC topology synthesis problem [24].

Algorithm 2 presents our SAL procedure. At the initial mapping step (line 6), we adapt our random mapping strategy (line 6), which is found, a sanity check step (line 16) is performed in order to guarantee that the total amount of traffic rates in each physical link of the tile-based NoC architecture does not exceed the capacity (i.e., total available bandwidth) of the link. Since we are adopting Z-Y-X routing in our work, once the core-to-tile mapping is done for each core in the given core graph \( G \), the total traffic rates in each link is easy to calculate. If the new solution has better cost, we should save the new solution to \( M_{best} \) (lines 17–19).
7. Experimental results

We implement our power- and latency-driven SAL-based mapping methodology (see Section 4), together with incremental mapping updating for reliability optimization (see Section 4.2) in C++. All the experiments are conducted on a computer with a 3.4 Ghz Intel i7 processor and 32 GB memory running Linux. We use TGFF [31] to generate four benchmarks with different characteristics (see Table 2). To enrich the diversity of benchmarks, we add some realistic benchmarks VOPD [32], IMP2 [24], the number of cores in these six benchmarks ranges from 12 to 108.

The parameters for the NoCs are set as: 500 Mhz clock frequency, 16-bit flip and 8-flit packet. In our work, CP measures the power consumed when data is transmitted through the routers and links by the Dsent [33] power model, and the NL is evaluated by our latency model presented in Section 5. Also, the reliability metric VNT is evaluated by reliability model presented in Section 4 and $j_0 = 10 mA/\mu m^2$ (see Table 1) is taken from [27].

7.1. Results of our latency-driven mapping

In this section, we present the results about the accuracy of our proposed latency model, and its performance in latency-driven mapping.

7.1.1. Accuracy of the latency model

In this section, we verify the accuracy of the latency model for links and routers by comparing our model with the widely used NoC simulator BookSim [34].

We first use a simple benchmark with $4 \times 4 \times 2$ mesh architecture as shown in Fig. 8 to verify the accuracy of our latency model. This benchmark has three traffic flows: tile 0 → tile 6, tile 0 → tile 4, tile 29 → tile 7. We gradually increase the injection rates of these three flows from 0 to 0.45 flits per cycle, as shown in Fig. 9.

We then respectively use our latency model and BookSim to evaluate the average NL of these three traffic flows. From Fig. 9 we can see that when the injection rate is lower than 0.3 packets/cycle, which implies that the contention of the network is in the low and medium range, our model can match the results of BookSim very well; when we further increase the injection rate, our model can still correctly predict the trend of the changes in NL.

7.1.2. Results of the latency-driven mapping

In this part, we use SAL to do latency-driven mapping (by setting $\alpha = 0$ in Equation (1)) using our latency models. We compare the results with contention-aware model (CAM) [16], hop-count model [5] and queuing model (QM) [15]. All the four latency models are integrated with the same SAL engine. After obtaining the mapped results, we use BookSim simulator to evaluate the latency results of four different models.

Table 2 shows the benchmarks we have tested together with their corresponding results. Note that $N$ represents the iteration number of SAL algorithm (see Algorithm 2). Not surprising, Table 2 shows the latency

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Parameters</th>
<th>Average NL (in cycles)</th>
<th>Runtime (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOPD</td>
<td>12</td>
<td>15</td>
<td>$10^5$</td>
</tr>
<tr>
<td>b1</td>
<td>18</td>
<td>36</td>
<td>$10^7$</td>
</tr>
<tr>
<td>b2</td>
<td>27</td>
<td>26</td>
<td>$10^7$</td>
</tr>
<tr>
<td>IMP2</td>
<td>56</td>
<td>196</td>
<td>$10^7$</td>
</tr>
<tr>
<td>b3</td>
<td>72</td>
<td>62</td>
<td>$10^7$</td>
</tr>
<tr>
<td>b4</td>
<td>108</td>
<td>144</td>
<td>$10^7$</td>
</tr>
<tr>
<td>Average</td>
<td>1</td>
<td>1.17</td>
<td>1.25</td>
</tr>
</tbody>
</table>
results of hop-count are 17% worse than our model. For more complex model CAM, it has 25% larger latency than our latency model. This is because our model measures the degree of network contention by traffic amount, but CAM estimates the network contention by just counting the number of shared links, which does not reflect the actual queuing delay. This can be further illustrated by the mapping results of the benchmark b1 (whose core graph is shown in Fig. 10) as presented in Table 3, where we list the complete routing path of each flow obtained by three different latency models: hop-count, CAM and our model.

Table 3

<table>
<thead>
<tr>
<th>Flow</th>
<th>Paths by hop-count</th>
<th>Paths by CAM</th>
<th>Paths by our model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 → 3</td>
<td>17 → 14 → 13</td>
<td>3 → 4 → 5</td>
<td>14 → 13</td>
</tr>
<tr>
<td>0 → 7</td>
<td>17 → 14 → 11 → 10</td>
<td>3 → 4</td>
<td>14 → 17 → 16</td>
</tr>
<tr>
<td>0 → 10</td>
<td>17 → 14</td>
<td>3 → 12</td>
<td>14 → 5</td>
</tr>
<tr>
<td>0 → 15</td>
<td>17 → 16 → 15</td>
<td>3 → 6</td>
<td>14 → 11</td>
</tr>
<tr>
<td>1 → 8</td>
<td>9 → 12</td>
<td>11 → 14</td>
<td>3 → 4</td>
</tr>
<tr>
<td>2 → 3</td>
<td>4 → 13</td>
<td>13 → 4 → 5</td>
<td>7 → 16 → 13</td>
</tr>
<tr>
<td>2 → 7</td>
<td>4 → 13 → 10</td>
<td>13 → 4</td>
<td>7 → 16</td>
</tr>
<tr>
<td>2 → 10</td>
<td>4 → 13 → 14</td>
<td>13 → 12</td>
<td>7 → 4 → 5</td>
</tr>
<tr>
<td>3 → 7</td>
<td>13 → 10</td>
<td>5 → 4</td>
<td>13 → 16</td>
</tr>
<tr>
<td>3 → 8</td>
<td>13 → 12</td>
<td>5 → 14</td>
<td>13 → 4</td>
</tr>
<tr>
<td>3 → 10</td>
<td>13 → 14</td>
<td>5 → 14 → 13 → 12</td>
<td>13 → 4 → 5</td>
</tr>
<tr>
<td>4 → 2</td>
<td>1 → 4</td>
<td>1 → 10 → 13</td>
<td>15 → 6 → 7</td>
</tr>
<tr>
<td>4 → 3</td>
<td>1 → 10 → 13</td>
<td>1 → 4 → 5</td>
<td>15 → 12 → 13</td>
</tr>
<tr>
<td>4 → 5</td>
<td>1 → 2</td>
<td>1 → 10</td>
<td>15 → 12</td>
</tr>
<tr>
<td>4 → 7</td>
<td>1 → 10</td>
<td>1 → 4</td>
<td>15 → 16</td>
</tr>
<tr>
<td>4 → 9</td>
<td>1 → 0</td>
<td>1 → 2</td>
<td>15 → 6</td>
</tr>
<tr>
<td>5 → 1</td>
<td>2 → 11 → 10 → 9</td>
<td>10 → 11</td>
<td>12 → 3</td>
</tr>
<tr>
<td>5 → 17</td>
<td>2 → 5</td>
<td>10 → 9</td>
<td>12 → 9</td>
</tr>
<tr>
<td>6 → 2</td>
<td>16 → 7 → 4</td>
<td>7 → 16 → 13</td>
<td>17 → 8 → 7</td>
</tr>
<tr>
<td>6 → 3</td>
<td>16 → 13</td>
<td>7 → 4 → 5</td>
<td>17 → 14 → 13</td>
</tr>
<tr>
<td>6 → 7</td>
<td>16 → 13 → 10</td>
<td>7 → 4</td>
<td>17 → 16</td>
</tr>
<tr>
<td>7 → 10</td>
<td>10 → 13 → 14</td>
<td>4 → 13 → 12</td>
<td>16 → 7 → 4 → 5</td>
</tr>
<tr>
<td>6 → 15</td>
<td>16 → 15</td>
<td>7 → 6</td>
<td>17 → 14 → 11</td>
</tr>
<tr>
<td>7 → 1</td>
<td>10 → 9</td>
<td>4 → 13 → 10 → 11</td>
<td>16 → 7 → 4 → 3</td>
</tr>
<tr>
<td>8 → 2</td>
<td>12 → 3 → 4</td>
<td>14 → 13</td>
<td>4 → 7</td>
</tr>
<tr>
<td>8 → 3</td>
<td>12 → 13</td>
<td>14 → 5</td>
<td>4 → 13</td>
</tr>
<tr>
<td>8 → 7</td>
<td>12 → 9 → 10</td>
<td>14 → 5 → 4</td>
<td>4 → 13 → 16</td>
</tr>
<tr>
<td>8 → 10</td>
<td>12 → 13 → 14</td>
<td>14 → 13 → 12</td>
<td>4 → 5</td>
</tr>
<tr>
<td>8 → 15</td>
<td>12 → 15</td>
<td>14 → 5 → 8 → 7 → 6</td>
<td>4 → 13 → 10 → 11</td>
</tr>
<tr>
<td>9 → 1</td>
<td>0 → 9</td>
<td>2 → 11</td>
<td>6 → 3</td>
</tr>
<tr>
<td>12 → 2</td>
<td>3 → 4</td>
<td>16 → 13</td>
<td>1 → 4 → 7</td>
</tr>
<tr>
<td>12 → 8</td>
<td>3 → 12</td>
<td>16 → 13 → 14</td>
<td>1 → 4</td>
</tr>
<tr>
<td>12 → 16</td>
<td>3 → 6</td>
<td>16 → 15</td>
<td>1 → 2</td>
</tr>
<tr>
<td>14 → 2</td>
<td>7 → 4</td>
<td>17 → 14 → 13</td>
<td>8 → 7</td>
</tr>
<tr>
<td>14 → 8</td>
<td>7 → 16 → 13 → 12</td>
<td>17 → 14</td>
<td>8 → 5 → 4</td>
</tr>
<tr>
<td>14 → 16</td>
<td>7 → 6</td>
<td>17 → 16 → 15</td>
<td>8 → 5 → 2</td>
</tr>
<tr>
<td>Average hop count</td>
<td>2.47</td>
<td>2.53</td>
<td>2.53</td>
</tr>
</tbody>
</table>

Fig. 8. A simple benchmark with three representative flows.

Fig. 9. Results of the benchmark shown in Fig. 8.

Fig. 10. Core graph of b1, two flows are not drawn because of limited space. One is from core 8 to core 3 with 0.313 flits/cycle, another is from core 12 to core 8 with 0.112 flits/cycle.

Table 2 also shows the comparison results between our model and the most complex delay model QM. Due to the large time cost of QM model, we failed to get the results for five out of the total six benchmarks within 15 days, which is the maximum allowable run-time we set for each run.
For the VOPD benchmark, our model found better solution than QM model with more than 1000× speedup.

7.2. Results of power-driven mapping

In this section, we present the results of our power-driven SAL mapping approach. To show the effectiveness of our SAL mapping algorithm, we compare it with recent work PSO [5] and simulated annealing (SA). Note that for fair comparison, PSO is implemented as multiple-particle swarm optimization with random initial population, and we also use the same bit-power model obtained by DSENT [33] and cost function in our SAL approach (by setting $\beta = 0$ in Equation (1)).

We test these three algorithms on 6 benchmarks. Table 4 list the benchmark parameters and the results. Note that $P$ and $NP$ respectively

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Parameters</th>
<th>Network Power (in W)</th>
<th>Runtime (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SAL</td>
<td>PSO</td>
</tr>
<tr>
<td>VOPD</td>
<td>12 15 3.9*10^5 1000 50</td>
<td>0.0703 0.0708 0.0779</td>
<td>289.7 378.1 665.7</td>
</tr>
<tr>
<td>b1</td>
<td>18 36 8*10^6 1000 50</td>
<td>0.1394 0.1447 0.1501</td>
<td>866.2 1072.2 4788.9</td>
</tr>
<tr>
<td>b2</td>
<td>27 26 2.0*10^7 5000 100</td>
<td>0.1458 0.1467 0.1482</td>
<td>1572.5 10756.1 4797.1</td>
</tr>
<tr>
<td>b3</td>
<td>56 196 10^7 5000 100</td>
<td>0.4932 0.5505 0.4311</td>
<td>7101.1 115102.0 24202.0</td>
</tr>
<tr>
<td>b4</td>
<td>72 62 3*10^7 5000 100</td>
<td>0.3967 0.4372 0.3969</td>
<td>8072.5 75243.6 36338.5</td>
</tr>
<tr>
<td>Average</td>
<td>1 1.07 0.9997</td>
<td>1 7.53 3.44</td>
<td></td>
</tr>
</tbody>
</table>
represents the iteration number of PSO algorithm and the number of particles. On average, our SAL-based mapping method can achieve 7% less power than PSO with about 7.5% speedup, and about 3.4% speedup compared to SA with nearly the same quality of results.

7.3. Results of reliability optimization

Table 5 shows the effectiveness of our reliability optimization step presented in Section 4.2. We can see that on average 2.47% improvement in VNT can be achieved on the five out of the six tested benchmarks, with slightly lower NL. Especially, the benchmark IMP2 achieves more than 10% improvement in VNT with 7.64% latency increase.

As one example, Fig. 13 shows the VNT distribution of the TSV bundles in benchmark b1 before and after the optimization. Before optimization, the VNT of the 8th TSV bundle becomes the bottleneck of the whole chip. After optimization, the VNT distribution of the TSV bundles is more even, and the smallest VNT of the TSV bundles can be increased by 2 ×. These results indicate that the VNT distribution is more balanced after the remapping optimization based on core power distribution.

7.4. Scalability of our SAL-Based mapping approach

The complexity of our SAL-based mapping algorithm (see Algorithm 2) is affected by the size of the core graph, more specifically, its numbers of cores and flows. In our work, we designed two sets of experiments to show how the run-time and NL vary as we increase the size of the task graph. First, we fixed the number of cores to be 32, but varied the flow number from 0, 10, 20, 30, 40 and up to 50 to show how the flow number influence the complexity of algorithm. For each particular flow number, we randomly generated the traffic flows (source-destination pair and traffic rate) among the 32 cores. Fig. 14 shows that both the NL and run-time increase approximate linearly with the number of traffic flows.

Then, we fixed the total number of traffic flows to be 40, and varied the number of cores from 16, 50, 72, 98, 128 and up to 162. Again, for each particular core number, we randomly generated the 40 traffic flows. Fig. 15 shows that NL decreases as we increase the number of cores because of the reducing network contention, but the run-time increases approximately linearly with the number of cores.

8. Conclusion

In this work, we solve the mapping problem in tile-based 3D NoC architectures with an SAL algorithm that can integrate our new queue-based latency models and network power as the optimization objective. We also propose an incremental mapping updating step to optimize the reliability of TSVs in the 3D chip. Results show that our SAL-based mapping algorithm can achieve better power and latency results than prior work, and our incremental mapping approach can improve the TSV reliability significantly.

References


