Delay Bound Optimization in NoC Using a Discrete Firefly Algorithm

Gaoming Du 1,*, Chao Tian 1, Zhenmin Li 1, Duoli Zhang 1, Chuan Zhang 2,3, Xiaolei Wang 1 and Yongsheng Yin 1

1 Institute of VLSI Design, Hefei University of Technology, Hefei 230601, China; 2014213304@hfut.edu.cn (C.T.); zhenmin.li@hfut.edu.cn (Z.L.); zhangduoli@hfut.edu.cn (D.Z.); wangxiaolei@hfut.edu.cn (X.W.); yinyongsheng@hfut.edu.cn (Y.Y.)
2 National Mobile Communications Research Laboratory, Southeast University, Nanjing 211189, China; chzhang@seu.edu.cn
3 Purple Mountain Laboratories, Nanjing 211111, China
* Correspondence: dugaoming@hfut.edu.cn

Received: 25 October 2019; Accepted: 29 November 2019; Published: 9 December 2019

Abstract: The delay bound in system on chips (SoC) represents the worst-case traverse time of on-chip communication. In network on chip (NoC)-based SoC, optimizing the delay bound is challenging due to two aspects: (1) the delay bound is hard to obtain by traditional methods such as simulation; (2) the delay bound changes with the different application mappings. In this paper, we propose a delay bound optimization method using discrete firefly optimization algorithms (DBFA). First, we present a formal analytical delay bound model based on network calculus for both unipath and multipath routing in NoCs. We then set every flow in the application as the target flow and calculate the delay bound using the proposed model. Finally, we adopt firefly algorithm (FA) as the optimization method for minimizing the delay bound. We used industry patterns (video object plane decoder (VOPD), multiwindow display (MWD), etc.) to verify the effectiveness of delay bound optimization method. Experiments show that the proposed method is both effective and reliable, with a maximum optimization of 42.86%.

Keywords: delay bound; network calculus; network on chip; firefly algorithm

1. Introduction

In modern system on chips (SoC) design, average [1–3] and worst-case [4–15] performance are two essential metrics for communication architecture. Quality-of-service (QoS) in network on chip (NoC) represents the worst-case traverse time of on-chip communications. In NoC based SoC, optimizing the delay bound is challenging. Formal approaches have been proposed for delay bound modeling [16–21]. Wang et al. [16] used signal flow chart and signal time measures to analyze the upper bound of transmission delay time. Ren et al. [17] analyzed the communication delay bound for individual flows in NoC using an improved asymmetric multichannel structure of a router. Lu et al. [19,20] modeled a classic input-queuing virtual channel router and a unified platform in Simulink based on xMAS to improve the accuracy of NoC performance analysis.

As an effective tool, network calculus has been applied in NoC performance analysis [21,22]. How to improve the tightness of a NoC performance model based on network calculus has become one of the important research directions. Saggio et al. [23] validated the effectiveness of network calculus for delay bound modeling. To improve the accuracy of a NoC delay bound, Zhao et al. [24] used simulated annealing (SA) to automatically calculate the simulation parameters. They also extended the method from delay bound to a backlog bound tightness study [25], where SA was replaced by an adaptive simulated annealing (ASA) algorithm, with higher efficiency and precision.
The works above show that heuristic approaches can be an effective tool for worst-case performance analysis and that motivated us to improve the delay bound tightness under different mappings. Other than optimizing specific configuration, we focused on optimizing the delay bound in specific application mapping. We observed that different mapping schemes have different delay bounds for NoC traffic flows. When the mapping scheme changes, the contention scenarios also change. However, delay bound optimization is challenging not only because such problems are hard, but also its model should be both accurate and fast enough for solution exploration.

The firefly algorithm (FA) was inspired by the behavior of fireflies flashing and was first put forward by Xin-She Yang [26]. In recent years, FA has been widely used to optimize NoC design in design spaces [27–29]. In this paper, we propose a delay bound optimization method using firefly algorithms (DBFA). Instead of improving the simulation parameter accuracy and analyzing network contention, DBFA attempts to find the optimal mapping scheme directly. The experimental results show that the mapping scheme determined by FA is closer to the optimal mapping scheme than discrete particle swarm optimization (DPSO) [30]. Our contribution is summarized as follows:

- We modeled the worst-case delay bound in network on chip using network calculus. It is suitable for both unipath and multipath routing NoCs.
- We adopted FA to optimize the end-to-end delay bound. FA shows high efficiency in many other fields, such as scheduling. To the best of our knowledge, this is the first work using FA for delay bound optimization.
- We performed extensive experiments using both synthetic and industry patterns. We also integrated our delay bound model into DPSO, a state-of-the-art algorithm, for performance comparison.

2. Delay Bound Analysis Using Network Calculus

Simulation experiments [31] can easily obtain latency and communication costs, but it is difficult to obtain a delay bound. The network calculus fills the gap. In Figure 1, \( R(t) \) and \( R^*(t) \) represent the actual arrival curve and service curve of NoC, respectively. A linear function \( \alpha(t) \) covers the maximum rate of \( R(t) \); \( \beta(t) \) represents the minimum service rate of \( R^*(t) \). Both are defined as follows:

\[
\begin{align*}
\alpha(t) &= \begin{cases} 
rt + b & t > 0 \\
0 & \text{otherwise}
\end{cases} \\
\beta(t) &= \begin{cases} 
R(t - T) & t > T \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

where \( r \) is the sustainable arrival rate, \( b \) is the burstiness, \( R \) is the minimum service rate, and \( T \) is the maximum processing latency. In particular, \( R \) is usually greater than \( r \) to ensure no packet dropping in NoC. Delay bound \( \bar{D}(t) \) of a flow \( f_i \) is calculated by finding the greatest horizontal distance \( h(\alpha, \beta) \) between its arrival curve \( \alpha(t) \) and the system equivalent service curve \( \beta(t) \). Hence the delay bound is derived as follows [9]:

\[
\bar{D}(t) = h(\alpha, \beta) = T + \frac{b}{R}.
\]  

To express more clearly, we show all the symbols in Table 1.
3. Delay Bound Optimization Using Discrete Firefly Algorithms

3.1. Problem Formulation

We first present a general mapping process through the following definitions to optimize the total communication cost.

To map an application characteristic graph \( G(V, E) \) to a NoC topology graph \( P(U, F) \), the mapping function \( \text{map}() \) should satisfy the following constraints:

\[
\begin{align*}
    v_i \in V & \Rightarrow \text{map}(v_i) \in U; \\
    v_i \neq v_j & \Leftrightarrow \text{map}(v_i) \neq \text{map}(v_j); \\
    \text{size}(V) & \leq \text{size}(U); \\
    b_{ij} & \leq B_{ij}.
\end{align*}
\]

The smaller the delay bound, the better the mapping results. In this paper, we assume the NoC topology is a mesh. Therefore, the delay bound is usually calculated by the following equations.

\[
\text{DelayBound} = \max(D_1, D_2, ..., D_n),
\]

where \( D_n \) is delay bound of every target flow, whose calculation is shown in the rest of this section. Thus, the optimization target is \( \min \text{DelayBound} \).

3.2. Delay Bound Model

We present the delay bound calculation model in Algorithm 1. The key idea of the delay bound analysis is obtaining the end to end equivalent service curve (ESC) by considering all kinds of resource sharing, shown in line 3–13, and summarized in four steps.
**Step 1:** Use the function `ClassifyConFlow()` to classify the flow contentions into a unified representation as \( f_{[a,b]} \), where \( a \) and \( b \), respectively, are the traffic injection and output router node. These flows could be a single flow or an aggregate flow consisting of several contention flows; i.e., both unipath and multipath routing are supported in this model.

**Step 2:** Calculate the arrive curve of \( f_{[a,b]} \) at node \( a \).

**Step 3:** Calculate the equivalent service curve for the target flow and obtain the delay bound.

**Step 4:** Repeat Steps 1–3 to find the delay bound of every single flow and select the max value as the delay bound in the current mapping.

**Algorithm 1** Delay bound calculation

```
1 Input \( b_i(t) = R_i(t - T_i)^+, \ 1 \leq i \leq X \) \( \times Y, a_j = r_jt + b_j, \ 1 \leq j \leq FlowNum \)
2 Output \( D_{\text{max}} \)
3 for the flow \( f_k \in \{TagFlowSet\} \)
4   | ESC(f_k) \hspace{1cm} //To calculate the Equivalent Service Curve of flow \( f_k \).
5     NodeSet() \hspace{1cm} //find all nodes which \( f_k \) has passed.
6     CClassSet = ClassifyConFlow() \hspace{1cm} //Classify all the contention flows.
7     for \( f_{[a,b]} \in \text{CClassSet} \)
8         | ACSet = \text{AC}(f_{[a,b]})) \hspace{1cm} //Calculate the arrive curve of cut-flows and combine the same type flows.
9         if CrossContention \hspace{1cm} //The treatments of cross contention situation.
10            Cut all cross-contention flows \hspace{1cm} //The treatments of cross contention situation.
11                ACSet = \text{AC}_{\text{CrossCon}}(f_{[a,b]}, ACSet) \hspace{1cm} //Calculate the arrive curve of cut-flows and combine the same type flows.
12         else \hspace{1cm} //The treatments of other contention situations.
13                Delay(f_k, ESC) \hspace{1cm} //obtain it delay bound.
14         | D_{\text{max}} = \text{max}(Delay(f_k, ESC)) \hspace{1cm} //Obtain the worst-delay-bound for one scheme.
```

### 3.3. Distance Calculation

The calculation of distance is as follows.

\[
s_{mn} = \sum_{i=0}^{T_{num}} (|x^m_i - x^n_i| + |y^m_i - y^n_i|),
\]

where \( s_{mn} \) is the Manhattan distance between firefly \( X_m \) and \( X_n, (x^m_i, y^m_i) \) represents the coordinates of the mapping scheme of firefly \( m \) in task \( i \). Accordingly, \( (x^n_i, y^n_i) \) represents the coordinates of the mapping scheme of firefly \( n \) in task \( i \). And \( T_{num} \) is the total number of tasks.

In order to make the distance and the absorption coefficient in the same magnitude, we perform min-max operation to normalize the distance between two fireflies.

\[
s_{mn} = \frac{s_{mn} - s_{\text{min}}}{s_{\text{max}} - s_{\text{min}}}.
\]

where \( s_{\text{min}} \) is the minimum distance between two fireflies and \( s_{\text{max}} \) is the maximum distance for the target firefly. To make it more clear, take Figure 2 as an example. There are two fireflies \( X_m \) and \( X_n \) in Figure 2a,b, respectively.
For firefly \( X_m \), task numbers 1, 3, and 5 are mapped to \((4, 4)\)(node 0), \((3, 4)\)(node 5), and \((2, 4)\)(node 7), respectively. For \( X_n \), the positions of the above three tasks are \((2, 1)\)(node 4), \((2, 2)\)(node 5), and \((2, 3)\)(node 6), respectively. According to the Equation (7), we can calculate \( s_1 = |4 - 2| + |4 - 1| = 5 \), \( s_3 = |3 - 2| + |4 - 2| = 3 \), and \( s_5 = |2 - 2| + |4 - 3| = 1 \). The positions of task numbers 0, 2, and 4 have not changed, so \( s_0, s_2 \) and \( s_4 \) are all 0. At last, we can calculate the distance between fireflies \( X_m \) and \( X_n \) as \( s_{mn} = 0 + 5 + 0 + 3 + 0 + 1 = 9 \).

Figure 2c shows the theoretical max distance for firefly \( X_m \), which does not exist actually. The max distance is a ideal up-bound value, which is defined as mapping the current task to the theoretical farthest corner node; e.g., task 0 and 2 are mapping to node 15. In this example, \( s_{max} = 6 + 5 + 5 + 5 + 5 + 6 = 32 \). As a result, the distance between firefly \( X_m \) and \( X_n \) is \( S_{mn} = \frac{9}{32} = 0.28125 \).

3.4. Refreshing of Firefly Locations

The original firefly location refreshing formula [26] is as the following:

\[
X_i(t + 1) = X_i(t) + \beta(r) \cdot (X_i(t) - X_i(t)) + \alpha \cdot \text{Random}().
\]  

In our approach, we rewrite the firefly refreshing formula as follows.

\[
X_i(t + 1) = (1 - \beta(r)) \cdot X_i(t) + \beta(r) \cdot X_i(t) + \alpha \cdot \text{Random}() 
\]  

Thus, this formula consists of the following two parts:

1. \( \beta \) movement. Fireflies refresh because of the attractiveness between any two fireflies; it is related to attractiveness \( \beta(r) \), so we call it \( \beta \) movement.
2. \( \alpha \) movement. Fireflies refresh because of the random movement; it is related to the maximal random step \( \alpha \), so we call it \( \alpha \) movement.

Each firefly moves towards the brighter fireflies through \( \beta \) movement (the mapping scheme least delay bound). Later, each firefly moves randomly though \( \alpha \) movement to find a better mapping scheme. The \( \alpha \) movement rule is different between the non optimal firefly and the optimal firefly, so we call the former \( \alpha 1 \) movement and the latter \( \alpha 2 \) movement.

1. \( \alpha 1 \) movement

In order to learn the \( \alpha 1 \) movement of firefly \( X_i(t_f) \) after \( \beta \) movement, we defined a set \( w \) to record the positions of elements which occupy different positions between \( X_i(t_f) \) and \( X_i(t_f) \). Then we chose \( \alpha \) number of positions from \( w \) as exchange positions, and exchanged each two elements with probability \( p \) to finish the \( \alpha 1 \) movement.

2. \( \alpha 2 \) movement

The \( \alpha 2 \) movement is used for preventing an optimal firefly from falling into a local optimum. It increases the exploring capability of DBFA. We chose \( \alpha \) number of positions in local optimum mapping scheme and exchanged each two elements with probability \( q \) to finish the \( \alpha 2 \) movement. The probability \( q \) was randomly generated by a uniform distribution function.
3.5. Pseudo Code of Firefly Algorithm

The pseudo code of DBFA is shown in Algorithm 2. All the steps described above are covered in this algorithm; i.e., defining firefly, calculating distance, refreshing locations, and optimizing the delay bounds.

For algorithm complexity, if there are \( n \) fireflies in the colony, we obtain a local optimum firefly in every generation when executing Algorithm 2. For other fireflies, the calculation process of each firefly from line 5 – 14 would be carried out \( (n - 1) \) times. Thus the iterations of all fireflies is \( (n - 1)^2 \). After all, the complexity of the FA is \( O((n - 1)^2) \). Therefore, the complexity of whole program of DBFA is \( o(m(n - 1)^2(\frac{1}{2} + \frac{1}{4})) \).

Algorithm 2 DBFA

```plaintext
1 Input  The application characteristic graph \( G_f(PE, E_f) \), NoC topology graph \( D_f(R, P_f) \), fireflies colony number \( Fnum \), maximum iterations \( Gmax \), absorption coefficient \( \gamma \), maximum attractiveness \( \beta_0 \), and maximal random step \( a \).
2 Output Global optimum firefly \( X_{\text{global}} \)
3 Set parameters to initialize fireflies
4 for all \( G < GMax \) 
5 | 
6 FindGlobal() = min \( M(D_{\text{max}}) \) // Search for the global optimum firefly \( X_{\text{global}} \) with a minimized delay bound.
7 for all \( i < Fnum \) 
8 | 
9 for all \( i < Fnum \) 
10 | 
11 S = Compute Distance(\( X_i \)) // Compute the distance between \( X_i(t_1) \) and \( X_j(t_1) \).
12 if \( \frac{1}{\gamma S_{ij}} < \frac{1}{\gamma S_{ji}} \times \exp(-\gamma S_{ij}) \) // Satisfy move condition that \( X_i(t_1) \) moves towards \( X_j(t_1) \).
13 | 
14 Compute the attractiveness between \( X_i(t_1) \) and \( X_j(t_1) \).
15 \( X_i(t_1) \) // execute \( \beta \) movement.
16 \( X_j(t_1) \) // execute \( a1 \) movement.
17 | 
18 | 
19 | 
20 \( X_{\text{global}} \) // execute \( a2 \) movement.
21 |
```

3.6. Example of the Delay Bound Optimizing Process

To further understand the movement procedure, still take Figure 2 as an example. We calculated the delay bound of firefly \( X_m \) in Section 3 as \( \bar{D}_m = 1139.833 \). Using the same method, we can also obtain the delay bound of firefly \( X_n \) \( \bar{D}_n = 366 \). The distance between \( X_m \) and \( X_n \) is \( S_{mn} = 0.3125 \), so we assign \( \gamma = 0.3 \), which satisfies the moving condition \( \frac{1}{1139.833} < \frac{1}{0.3125} \times \exp(-0.3 \times 0.3125) \).

Therefore firefly \( X_m \) moves towards \( X_n \).

The attractiveness between fireflies can be calculated with the following formula.

\[
\beta(r) = \beta_0 e^{-\gamma r^2}.
\] (11)

We define \( \beta_0 = 1 \), so the attractiveness between \( X_m \) and \( X_n \) is \( \beta = 1 \times e^{(-0.3 \times 0.3125^2)} = 0.97 \).

As Figure 3 shows, compared with the mapping scheme of \( X_m \), there are three different positions in \( X_m \) (regardless the position with the value –1). We can see that the first different value is 1, so we look for the location where 1 is in \( X_m \). Use the probability \( \beta \) to change the value in the last one with value in the fifth number. Repeat this step until all the different positions are changed to the same. So far we have already finished the \( \beta \) movement.
The firefly $X_m$ which has gone through the $\beta$ movement continues to complete the $\alpha$ movement. This step makes sure the firefly will move towards a lighter firefly exactly (with a lower delay bound). We supposed firefly $X_n$ was the local optimum firefly, so we made it move according to the $\alpha$ movement rules. As Figure 4 shows, we randomly chose $\alpha$ positions in local optimum firefly and randomly produced a change probability $q$. In this case, we supposed that for $q = 0.57$ and $\alpha = 3$, the three positions would be <1,5,8>, changing the values with probability 0.57 in turn. So far, we can finish the $\alpha$ movement.

Mapping scheme of $X_n$

Randomly choose three positions, such as <1,5,8>

Change the value in 1 and 5 with probability $\alpha$

Mapping scheme after $\alpha$ movement

4. Experiments and Results

We performed experiments for the following three purposes: (1) proving the effectiveness of DBFA in delay bound optimization, (2) comparing the results with state-of-the-art work DPSO, and (3) verifying the tightness of DBFA compared to a simulation.

4.1. Setting Up

We mapped some applications to a mesh-NoC to test the reliability of our method. The characteristic graphs of industry patterns [30] such as picture-in-picture (PIP) [32], multiwindow display (MWD) [33], MP3DEC [34], MP3ENC [34], VOPD [35], and DVOPD [36] are as shown in Figures 5, 6a, and 7. The mesh-NoC scale was $4 \times 4$ and the whole progress was simulated by $C++$ and run in the platform of Ubuntu12.04. Experimental parameter settings are shown in Figure 2. The experimental parameters are shown in Table 2. $Fnum$ represents the total number of fireflies in fireflies group and $GMax$ represents the maximum number of iterations. $\gamma$ and $\alpha$ represent the absorption coefficient and the maximal random step, respectively.
Figure 5. The characteristic graph of industry patterns.

Table 2. DBFA parameter settings.

| Applications | Vertices | Edges | Fnum | GMax | γ  | α  | NoC Mesh |
|--------------|----------|-------|------|------|----|----|---------|
| PIP          | 8        | 8     | 10   | 300  | 0.29 | 3  | 4 × 2   |
| VOPD         | 16       | 21    | 20   | 400  | 0.3  | 4  | 4 × 4   |
| MWD          | 12       | 12    | 30   | 600  | 0.3  | 5  | 4 × 4   |
| 263ENC MP3DEC| 12       | 12    | 25   | 800  | 0.2  | 3  | 4 × 4   |
| MP3ENC MP3DEC| 13      | 13    | 25   | 800  | 0.18 | 3  | 4 × 4   |
| 263DEC MP3DEC| 14      | 15    | 25   | 800  | 0.25 | 3  | 4 × 4   |
| DVOPD        | 32       | 44    | 20   | 6000 | 0.28 | 6  | 8 × 4   |

Figure 6. Application mapping example of VOPD using RAND, DPSO, and DBFA.

Figure 7. The characteristic graph of DVOPD.

4.2. Experiment Results

VOPD has 21 flows, which is the largest and most complex characteristic graph in this paper, and we take it as an example to prove that optimization performance of DBFA is more convincing. The mapping scheme using the RAND, DPSO, and DBFA methods is shown in Figure 6b–d, where circles represent tasks and rectangles represent network nodes. The mapping scheme obtained by the three methods has corresponding delays of 55, 46, and 42, respectively.

The results of minimum worst-case delay bound in every generation are shown in Figure 8. At first, the beginning the worst delay bound was 55 cycles; after 400-times optimization, the delay bound was 42, which reduced by 23.64%. Compared with DPSO, DBFA can avoid the situation of the algorithm falling into a local optimum. DBFA is designed for NoC, by introducing α movement and β
movement to successfully identify and jump out of local optimal traps. This is an important reason why DBFA is more efficient than DPSO. Its delay bound for each flow is shown in Figure 9.

In order to enhance the comparison of the results, we have added DVOPD to the original six industry patterns. The scale of DVOPD is much larger than any other pattern. The experimental results are shown in Figure 10. Although the scale of NoC has been greatly expanded, DBFA still has stronger performance than DPSO.

For other applications, the optimized results of every flow are shown in Figures 11–15 and the biggest delay is the delay bound. What needs special explanation is in application PIP: DBFA is almost unoptimized because the PIP contains eight tasks and eight cores, seven of which are the same. The application is so simple that there is little room for optimization.

![Figure 8](image-url) **Figure 8.** The worst-case delay bound of local optimum and global optimum.

![Figure 9](image-url) **Figure 9.** The delay bound for each flow of VOPD.

![Figure 10](image-url) **Figure 10.** The delay bound for each flow of DVOPD.

![Figure 11](image-url) **Figure 11.** 263DEC MP3DEC.
4.3. Scalability Analysis

The seven industry patterns in the experiments can be divided into small-scale, medium-scale and large-scale applications. Specifically, the scale of NoC in PIP is only $4 \times 2$, which is the smallest among all patterns, and the scale of NoC in DVOPD is $8 \times 4$, which is the largest among all patterns. We calculated the delay bound of PIP in NoCs of different scales. The experimental results are shown in Figure 16. Experimental data shows that, although the scale of NoC varies greatly, the delay bound is almost unchanged. This shows that DBFA and DPSO have strong stability in delay bound optimization.
4.4. Running Time of CPU

We also studied the feasibility of the DBFA. We mapped these applications, varying the scales of NoC and the whole CPU (Intel i5-8400) running time from initialization, to get optimal mapping schemes, which are shown in Table 3, where with the increasing of flows, the running time of CPU would increase by a slight to moderate amount. DBFA optimization time reduced compared to DPSO, which shows that DBFA is more efficient at optimizing delay bounds.

Table 3. CPU running times.

| Applications | IJCA’15 [30] | DBFA | Applications | IJCA’15 [30] | DBFA |
|--------------|--------------|------|--------------|--------------|------|
| VOPD         | 116.45       | 97.20| 263ENC MP3DEC| 24.75        | 17.70|
| PIP          | 8.92         | 6.65 | MP3ENC MP3DEC| 41.28        | 32.98|
| MWD          | 21.06        | 13.39| 263DEC MP3DEC| 53.16        | 40.27|

*In order to compare the results under the same environment, we implemented the DPSO algorithm proposed in [30].

4.5. The Comparison of Optimized and Simulation Results

In order to verify the validity of DBFA, we performed simulations and compared the optimized analytical results with the simulation results, the using application MWD. In simulations, we used Verilog to design a 4 × 4 NoC; the global clock network was 50 MHz and each router node handled one flit with two cycles, so the maximum delay of data in the network was four cycles. The experimental results are shown in Figure 17. This figure proves the validity of DBFA for optimizing the delay bound. For several flows (flow 8 and flow 11), the difference between theoretical results and stimulation results was minor, proving the tightness of the analytical results too. It is also important to point out that for some flows, such as flow 5 and flow 10, there existed a big gap. This is partly because the simulation time and flow contention were not well explored during the simulation.

5. Conclusions

Optimizing a delay bound in NoC is both important and hard. When the application mapping changes, the contentions between flows also change, which result in a different delay bound. In this paper, we first derived an analytical model for end-to-end flows in NoC, which can automatically compute delay bound for the target flow, when given the specified mapping. Then, we proposed a
firefly algorithm for application mapping, with the delay bound minimization as the optimization objection. We called this framework as DBFA. Experiments showed that the proposed DBFA can not only optimize the delay bound for a specified application, with an optimization rate up to 42.86%, but also has a fast running time and tight accuracy.

**Author Contributions:** Data curation, C.T.; Formal analysis, D.Z.; Funding acquisition, G.D.; Methodology, C.T. and Z.L.; Project administration, G.D.; Resources, D.Z.; Software, Z.L.; Supervision, G.D.; Validation, C.T.; Visualization, C.Z.; Writing—original draft, C.T.; Writing—review & editing, G.D., Z.L., D.Z., C.Z., X.W. and Y.Y.

**Funding:** This work was supported in part by the National Key Research and Development Program under grant 2018YFB2202604, the University Synergy Innovation Program of Anhui Province under grant GXXT-2019-030, NSFC under grants 61871115 and 61501116, and Jiangsu Provincial NSF for Excellent Young Scholars under grant BK20180059.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**

1. Ogras, U.Y.; Bogdan, P.; Marculescu, R. An analytical approach for network-on-chip performance analysis. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2010, 29, 2001–2013. [CrossRef]
2. Bogdan, P. Workload characterization and its impact on multicore platform design. In Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS 2010), Scottsdale, AZ, USA, 24–29 October 2010; pp. 231–240. [CrossRef]
3. Wu, Y.; Min, G.; Ould-Khaoua, M.; Yin, H.; Wang, L. Analytical modelling of networks in multicore systems under bursty and batch arrival traffic. *J. Supercomput.* 2010, 51, 115–130. [CrossRef]
4. Azarnova, T.V.; Barkalov, S.A.; Ukhllova, V.V. Estimation of time characteristics of systems with network topology and stochastic processes of functioning. In Proceedings of the International Conference “Applied Mathematics, Computational Science and Mechanics: Current Problems” (AMCSM 2019), Athens, Greece, 28–30 December 2019; p. 1203. [CrossRef]
5. Kiasari, A.E.; Hessabi, S.; Sarbazi-Azad, H. PERMAP: A performance-aware mapping for application-specific SoCs. In Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2008), Leuven, Belgium, 2–4 July 2008; pp. 73–78. [CrossRef]
6. Qian, Z.; Juan, D.; Bogdan, P.; Tsui, C.Y.; Marculescu, D.; Marculescu, R. A Support Vector Regression (SVR)-Based Latency Model for Network-on-Chip (NoC) Architectures (TCAD). *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2015, 35, 471–484. [CrossRef]
7. Thiele, L.; Chakraborty, S.; Naedele, M. Real-time calculus for scheduling hard real-time systems. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000), Geneva, Switzerland, 28–31 May 2000; pp. IV-101–IV-104. [CrossRef]
8. Leontyev, H.; Chakraborty, S.; Anderson, J.H. Multiprocessor extensions to real-time calculus. In Proceedings of the Real-Time Systems Symposium (RTSS 2009), Washington, DC, USA, 1–4 December 2009; pp. 410–421. [CrossRef]
9. Cruz, R.L. A calculus for network delay. I. Network elements in isolation. *IEEE Trans. Inform. Theory* 1991, 37, 114–131. [CrossRef]
10. Qian, Y.; Lu, Z.; Dou, W. Analysis of communication delay bounds for network on chips. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2009), Yokohama, Japan, 19–22 January 2009; pp. 7–12. [CrossRef]
11. Du, G.; Zhang, C.; Lu, Z.; Saggio, A.; Gao, M. Worst-case performance analysis of 2-D mesh NoCs using multi-path minimal routing. In Proceedings of the ACM International Conference on Hardware/Software-Codeign and System Synthesis (CODES + ISSS 2012), Tampere, Finland, 7–12 October 2012; pp. 123–132. [CrossRef]
12. Giroudot, F.; Mifdaoui, A. Buffer-aware worst-case timing analysis of wormhole NoCs using network calculus. In Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2018), Porto, Portugal, 11–13 April 2018; pp. 37–48. [CrossRef]
13. Jafari, F.; Lu, Z.; Jantsch, A. Least Upper Delay Bound for VBR Flows in Networks-on-Chip with Virtual Channels. *ACM Trans. Des. Autom. Electron. Syst.* 2015, 20. [CrossRef]
14. Jiang, Y.; Liu, Y. *Stochastic Network Calculus*; Springer: London, UK, 2009.
15. Du, G.; Liu, G.; Zhang, Y. On the accuracy of stochastic delay bound for network on chip. In Proceedings of the IEEE/ACM International Symposium on Networks-on-Chip (NOCS 2017), Seoul, Korea, 19–20 October 2017. [CrossRef]

16. Wang, J.; Zhang, Y.; Huang, H. Analysis and compute of real-time signal flow delay for network on-chip. In Proceedings of the International Conference on Innovative Computing and Cloud Computing (ICCC 2011), Wuhan, China, 13–14 August 2011; pp. 107–112. [CrossRef]

17. Ren, X.; Gao, D.; Fan, X.; An, J. Analysis of delay bounds for NoC based on improved asymmetric multi-channel router. J. Jilin Univ. 2014, 44, 782–787. [CrossRef]

18. Ayed, H.; Ermont, J.; Scharbarg, J.; Fraboul, C. Towards a unified approach for worst-case analysis of Tilera-like and KalRay-like NoC architectures. In Proceedings of the IEEE International Workshop on Factory Communication Systems Proceedings (WFCS 2016), Aveiro, Portugal, 3–6 May 2016. [CrossRef]

19. Lu, Z.; Zhao, X. xMAS-based qos analysis methodology. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2018, 37, 364–377. [CrossRef]

20. Zhao, X.; Lu, Z. A Tool for xMAS-Based modeling and analysis of communication fabrics in Simulink. ACM Trans. Model. Comput. Simul. 2017, 27. [CrossRef]

21. Qian, Y.; Lu, Z.; Dou, W. Analysis of worst-case delay bounds for on-chip packet-switching networks. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2010, 29, 802–815. [CrossRef]

22. Long, Y.; Lu, Z.; Shen, H. Composable Worst-Case Delay Bound Analysis Using Network Calculus. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2018, 37, 705–709. [CrossRef]

23. Saggio, A.; Du, G.; Zhao, X.; Lu, Z. Validating Delay Bounds in Networks on Chip: Tightness and Pitfalls. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 8–10 July 2015; pp. 404–409. [CrossRef]

24. Zhao, X.; Lu, Z. Empowering study of delay bound tightness with simulated annealing. In Proceedings of the Design, Automation and Test in Europe (DATE 2014), Dresden, Germany, 24–28 March 2014. [CrossRef]

25. Zhao, X.; Lu, Z. Heuristics-aided tightness evaluation of analytical bounds in networks-on-chip. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2015, 34, 986–999. [CrossRef]

26. Yang, X. Firefly Algorithms for Multimodal Optimization. Lect. Notes Comput. Sci. 2009, 5729, 169–178. [CrossRef]

27. Umamaheswari, S.; Kirthiga, K.I.; Abinaya, B.S.; Ashwin, D. Cost aware task scheduling and core mapping on Network-on-Chip topology using Firefly algorithm. In Proceedings of the International Conference on Recent Trends in Information Technology (ICRIT 2013), Chennai, India, 25–27 July 2013; pp. 657–662. [CrossRef]

28. Gandomi, A.H.; Yang, X.; Alavi, A. HosseinMixed variable structural optimization using Firefly Algorithm. Comput. Struct. 2011, 89, 2325–2336. [CrossRef]

29. Ilamathi, K.; Rangarajan, P. Determining Effective Shortest Path in Asynchronous Network-on-Chip through Bio-Inspired Optimization Techniques. Wirel. Pers. Commun. 2018, 102, 3375–3392. [CrossRef]

30. Sahu, P.K.; Manna, K.; Chattopadhyay, S. Application Mapping onto Butterfly-Fat-Tree based Network-on-Chip using Discrete Particle Swarm Optimization. Int. J. Comput. Sci. Appl. 2015, 115, 13–22.

31. Garcia, M.G.; Aedo, C.E.; Bagherzadeh, N. A new approach to the Population-Based Incremental Learning algorithm using virtual regions for task mapping on NoCs. J. Syst. Architect. 2019, 97, 443–454. [CrossRef]

32. Bertozzi, D.; Jalabert, A.; Murali, S.; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Trans. Parallel Distrib. Syst. 2009, 16, 113–129. [CrossRef]

33. Chang, K.-C.; Chen, T.-F. Low-power algorithm for automatic topology generation for application-specific networks on chips. IET Comput. Digit. Techn. 2008, 2, 239–249. 20070049. [CrossRef]

34. Krishnan, S.; Karam, S.C.; Goran, K. Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2006, 14, 407–420. [CrossRef]
35. Murali, S.; De Micheli, G. Bandwidth-constrained mapping of cores onto NoC architectures. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2004), Paris, France, 16–20 February 2004; pp. 896–901. [CrossRef]

36. Concer, N.; Bononi, L.; Soulie, M. The connection-then-credit flow control protocol for heterogeneous multicore systems-on-chip. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2010, 869–882. [CrossRef]