J Supercomput (2015) 71:4056–4071
DOI 10.1007/s11227-015-1504-y

KL_GA: an application mapping algorithm for mesh-of-tree (MoT) architecture in network-on-chip design

Juan Fang[1] · Lu Yu[1] · Sitong Liu[1] · Jiajia Lu[1] · Tan Chen[1]

Published online: 21 August 2015

Abstract As the very large-scale integrated circuit designs enter the deep sub-micron era, many-core processors are regarded as promising architectures to keep up with the Moore's law. To provide effective communications between the on-chip components, network-on-chip was proposed as a new paradigm that exhibits better scalability than the traditional buses. There have been previous researches on application mappings to reduce the power consumption, the network latency and the network area overhead. However, some of the previous proposed algorithms such as the Kernighan–Lin algorithm (KL) and some genetic algorithms (GA) have the problem of finding the local best result instead of a global optimal solution. In this paper, we propose a novel application mapping algorithm for the mesh-of-tree network topology, called KL_GA algorithm. Our proposed algorithm takes the advantage of both the Kernighan–Lin algorithm and genetic algorithms to reduce the overall communication cost. Our KL_GA algorithm first generates a mapping solution using a KL-based method. In order to avoid the appearance of premature phenomena, we next apply a GA-based algorithm to get rid of the population trapped in the local optimum and re-generate a new population.

✉ Juan Fang
fangjuan@bjut.edu.cn

Lu Yu
yulu080513@163.com

Sitong Liu
sitong0289@163.com

Jiajia Lu
lu_longjia@163.com

Tan Chen
chentan@bjut.edu.cn

[1] College of Computer Science, Beijing University of Technology, Beijing 100124, China

Our evaluations show that, compared to the random mapping algorithm, our KL_GA algorithm saves the power by 21.6 % and reduces the network latency by 16.3 % on the average.

Keywords NoC · Kernighan–Lin algorithm · Genetic algorithm · MoT topology

1 Introduction

As the number of cores integrated into a single chip keeps increasing, traditional buses are no longer able to provide satisfying communication service due to their lack of scalability. Network-on-chip (NoC), which scales well with the number of cores, emerged as a promising solution for on-chip communication [1,2].

There have been NoC-based many-core processor prototypes developed such as the MIT RAW chip and Terascale chip [3,4]. The power consumed by NoC with respect to the whole chip power on these two types of many-core processors is significant, about 40 % for RAW and 30 % for Terascale.

Generally, factors affecting the NoC performance and power consumption include the network topology, the routing algorithm, application mappings, etc. In this work, we investigate schemes to effectively map applications to cores to reduce the NoC power consumption and achieve optimal network performance. There have been previous proposed application mapping schemes [5–9]. For example, Tosun [8] proposed an algorithm that maps frequently communicating tasks closer to each other to reduce the distance traversed by the data packets. However, there is a problem with their algorithm because it can get stuck at local minima and thus may not generate the optimal solution. Zhu et al. [9] proposed a turn reduction scheme for application mapping which is based on the Kernighan–Lin (KL) algorithm.

Their scheme takes advantage of express NoC channels and maps tasks communicating with each other to the same row (or column) to reduce the number of turns. Their scheme may not find the optimal result when the possible mappings that can be selected are limited.

In this paper, we propose a mapping heuristic algorithm (KL_GA algorithm) that is based on the KL algorithm and genetic algorithm (GA) to reduce the overall communication cost. KL algorithm is a bi-partitioning strategy that is motivated by the fact that communication cost can be reduced by placing frequently communicating cores closely. GA, on the other hand, is a class of algorithms for solving optimization and searching problems that is inspired by the biological evolution. The GA algorithms mimic the process of natural selection and genetic mechanism to find a fittest survivor.

However, a typical GA algorithm is prone to get biased by the appearance of the premature phenomenon during the process of simulated evolution. In our proposed KL_GA method, we first apply the KL algorithm to generate the current optimal mapping. Then we employ a modified GA algorithm to find the final optimal mapping. If a stagnation phenomenon is observed during this searching process, we establish a new population by restarting the population initialization procedure because the best solution that could be found cannot be better than the current optimal mapping. As a result, the premature phenomenon can be effectively avoided in our algorithm. Our evaluations show that, comparing to the random mapping algorithm, KL_GA reduces

the network latency by an average of 16.3 % and the power consumption gets decreased by an average of 21.6 %.

The rest of this paper is organized as follows: Sect. 2 discusses the related work and in Sect. 3, we give the problem definitions; Sect. 4 describes in detail about our proposed mapping algorithm; Sects. 5 and 6 introduce our experimental environment and experimental results; and finally we conclude the paper in Sect. 7.

2 Related work

There have been several previous works proposed that employ specially designed application mapping algorithms to improve the NoC performance or reduce the communication cost.

Sahu et al. [10] proposed a partition strategy that extends the Kernighan–Lin bi-partitioning algorithm to improve the static and dynamic performances of NoC. Shah et al. [11] proposed a mapping algorithm (called KLMAP) based on the Kernighan–Lin strategy that employs butterfly-fat-tree structure to identify the closeness of cores by analyzing their bandwidth requirements. KLMAP is designed to minimize the network latency with satisfying the requirement of application's bandwidth first. Tosun [12] proposed a clustering scheme that utilizes integer linear programming (ILP) to effectively map applications to cores. In his scheme, the task graph and the mesh network are first partitioned into sub-graphs and sub-meshes. Then an ILP-based algorithm is employed to find an optimal mapping that maps each sub-graph onto a sub-mesh to improve the network performance.

However, the effectiveness of the KL algorithm has its limitation and the resulted mappings generated by the KL algorithm may not be optimal. In comparison, the GA has been observed to perform better than the KL algorithm in application mappings [8,13–15]. The issue with the genetic algorithm is that it needs to be designed carefully to avoid premature convergence to local optima; otherwise, we cannot get optimal results from GA either.

Tosun et al. [8,13] presented a mapping algorithm based on genetic algorithm to solve the energy and communication-aware mapping problems. Wang et al. [14] proposed a novel logistic scheme employing adaptive genetic algorithm for homogeneous 3D NoCs. Zang and You [15] designed an application mapping algorithm based on simulated annealing. Their algorithm combines the genetic algorithm and the simulated annealing algorithm together and searches for a low-power mapping solution through adaptive crossover and mutation. A reliability-aware mapping scheme for multi applications was proposed by Khalili et al. [16]. The proposed scheme utilizes the smallest rectangular region based on a given application core graph to place the given application and uses a heuristic algorithm to select a region which results minimum overall performance and communication energy.

Besides those works, researchers have also done many other work for application mapping. Some researchers dealed application mapping and dynamic scheduling with respect to the mapped communication [17,19]. Zhang et al. proposed overlay mapping technique by selecting a subset of nodes and links from the substrate-hosting network [20]. Researchers also combined the software and hardware together to solve mapping

issue [18,21]. Sahu [22] proposed a new mapping technique based on discrete Particle Swarm Optimization.

Most of these proposed GA-based algorithms have the issue that their initial solutions are generated randomly. Since the genetic algorithm takes the initial solutions as parents to generate offspring, stepping by selection, crossover and mutation, random initial solutions may not lead to the optimal final result.

The offspring of the initial solution cannot be good if the initial solution is not selected carefully. Because the GA algorithm repeats the procedure to generate offspring from parents, more offspring that does not have good fitness will be generated and the final mapping solution given by the GA algorithm will drift further away from optimal. To solve this problem, we propose a KL_GA algorithm for application mappings.

The novelty of our proposed algorithm is that we check for the occurrence of stagnation phenomenon during the search process. If the stagnation phenomenon is detected which means the optimal solution to be generated will be worse than the current optimal, we restart the population initialization procedure to establish a new population. As a result, the premature convergence of local optima can be avoided.

3 Problem definition

3.1 Energy model

In this paper, we focus on minimizing the NoC communication cost. The network power consumption is closely related to the energy consumed by data communication between tiles. Several NoC energy models have been proposed by previous research, for example, Hu and Marculescu [23] proposed a well-accepted energy model given as follows:

$$E_{bit} = E_{Sbit} + E_{Bbit} + E_{Wbit} + E_{Lbit}.$$

In this model, E_{bit} represents the energy consumed by transferring one bit of data by routers. E_{Sbit}, E_{Bbit}, E_{Wbit} and E_{Lbit} represent the energy consumed by the crossbar, the buffers, the wires and the links, respectively. When the network is of a large scale, the energy consumed by the buffers and the wires can be negligible. So the model can be rewritten as follows:

$$E_{bit}^{i,j} = n_{i,j} \times E_{Sbit} + (n_{i,j} - 1) \times E_{Lbit},$$

where $E_{bit}^{S,D}$ represents the energy consumed by transferring one bit of data from tile T_i to tile T_j, $n_{i,j}$ represents the number of routers the one bit of data travels through, generally measured by hops, namely the Manhattan distance. So the communication cost between two tiles can be calculated by

$$E_{i,j} = v_{i,j} \times E_{bit}^{i,j} = v_{i,j} \times n_{i,j} \times E_{Sbit} + v_{i,j} \times (n_{i,j} - 1) \times E_{Lbit}.$$

In this formula, $v_{S,D}$ is the network traffic between tile T_i and tile T_j.

So the total energy of the network is

$$E_{total} = \sum_{i,j}^{R} E_{i,j}.$$

3.2 Description of application mapping

Application mapping algorithms aim to assign a given task to a specific core in the NoC to meet certain constraints such as minimizing the network power consumption. First, we give the following definitions:

Definition 1 Task communication graph (TCG), a graph $G(T, E)$ in which $t_i \in T$ represents a task in the application and $e_{i,j} \in E$ represents the traffic between task t_i and task t_j.

Definition 2 Topology graph (TG) is a graph $T(P, L)$. $p_i \in P$ represents a processing core in the network topology and $l_{i,j} \in L$ represents a link between core p_i and core p_j.

The mapping algorithm can be formulated as follows:

Mapping algorithm: given an $G(T, E)$ and $T(P, L)$, find a one-to-one function map: $G->T$ which meets

$$\begin{aligned} &\forall t_i \in T \; map(t_i) \in P \\ &\forall t_i \neq t_j \; map(t_i) \neq map(t_j) \\ Size(T) &\leq Size(P). \\ &min\{E_{total}\}. \end{aligned}$$

4 Our proposed mapping algorithm

In this section, we describe our proposed heuristic-based mapping algorithm, named KL_GA, which combines the KL algorithm and genetic algorithm to reduce the overall communication cost. KL algorithm is often used for solving graph partitioning problem. The algorithm attempts to find a partition of a graph by dividing the graph into two subsets recursively such that the sum of the costs of edges between nodes in the two subsets is minimized. Since tasks with a significant amount of traffic need to be placed close to each other to reduce their Manhattan distance, we apply the KL algorithm to generate the initial optimal mapping solution. GA is a searching heuristic that mimics the process of natural selection to generate optimal solutions through inheritance, mutation, selection and crossover. We also employ GA in our proposed algorithm to search for optimal mapping solutions. Even though the randomly generated initial solution could enlarge the searching scope, the convergence rate of GA is greatly compromised by such initial solutions.

In addition, the low quality of initial population can easily make the solution to be trapped in local optimum. To solve this problem, we designed a jumping strategy in our GA method. Whenever the offspring are evaluated to be not superior to their parents, the jumping strategy is adopted to break away from local optimal solution and regenerate the initial population to keep searching for an optimal solution.

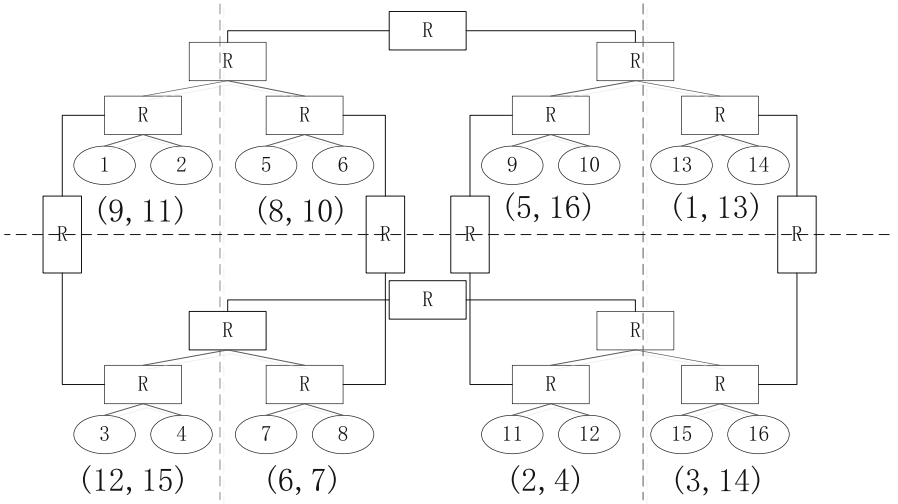

Fig. 1 An example of a chromosome with 16 genes (**a**) and the mapping of the chromosome onto a 4 × 4 MoT NoC (**b**)

In our GA method, we first create a chromosome structure that represents a valid mapping, where a gene represents a tile in NoC topology. Figure 1 shows the mapping of a chromosome onto a mesh-of-tree (MoT) topology.

GA algorithms use fitness function to evaluate the chances of an individual's survival in a group. The larger an individual's fitness value is, the more likely its genes will survive in the future generations. In our method, we use the reciprocal of energy function as the fitness function.

$$FitFun() = \frac{1}{E_{total}} = \frac{1}{\sum_{i,j}^{R} (v_{i,j} \times n_{i,j} \times E_{Sbit} + v_{i,j} \times (n_{i,j} - 1) \times E_{Lbit})}.$$

The main steps of our proposed mapping algorithm are described as follows (Fig. 4):
Step1 Generate the mapping solution p by the KL algorithm. We assume the current optimal solution $s = p$ and set he total number of iterations to be N and the current level of GA to be n, respectively. We assume that the number of nodes in TCG is equal to the number of tiles in the TG; otherwise we will add some dummy nodes.

Initially, all tasks are defined as in one partition. Because the result of the KL algorithm is relevant to the initial partition, we run the KL for K times by generating the initial partition randomly each time.

First the graph $G(T, E)$ is partitioned into two subsets, A and B of equal size. Then we calculate the internal cost $I_a(I_b)$ and external cost $E_a(E_b)$ of each node in A and B, and the external and internal costs of a vertex in A are calculated by Eqs. 1 and 2. Then we exchange nodes in A and B to find an optimal partition of the graph $G(T, E)$ using Eq. 4, which maximizes the difference between the internal cost and external cost. The partition continues recursively until there are two nodes in each subset.

$$E_i = \sum_{e=\{v_i, v_j\} \in E, v_j \in B} w(v_i, v_j) \tag{1}$$

$$I_i = \sum_{e=\{v_i,v_j\}\in E, v_j\in A} w(v_i, v_j) \tag{2}$$

$$D_a = E_a - I_a \tag{3}$$

$$T_{old} - T_{new} = D_a + D_b - 2c_{a,b}, \tag{4}$$

where $c_{a,b}$ represents the cost of the possible edges between a and b.

The algorithm is described in the following algorithm:

Algorithm: Kernighan-Lin-partitioning

Data: G, level

Result: *p*

```
begin
    p = Random_Initial_Mapping(G)
    for (i = 1 to K)
       Generate the initial partition t
       (p1, p2) = random partitioning of T
       KL(G, p1, p2)
       Kernighan-Lin-partitioning(G, level + 1, p1)
       Kernighan-Lin-partitioning(G, level + 1, p2)
       if FitFun(t) > FitFun(p) then
          p=t
    Return p
end

KL(G,p1,p2)
Data: G,p1,p2
Result: updated p1,p2

begin
    Current_partition = best_partition = (p1, p2)
    Compute D values for a in p1 and b in p2
    for (i = 1 to n)
        Locked[i] = false
    for (s = 1 to n/2)
        find unlocked a[i] from p1 and b[j] from p2, such that g[s] = D[a[i]] +
D[b[i]] – 2*c[a[i]b[j]] is maximal
        move a[i] to p2 and b[j] to p1
        current_partition = (p1, p2)
        Locked[a[i]] = true
       Locked[b[j]] = true
       best_partition = Get_better_partition(best_partition, current_partition)
       Update D values for p1 and p2
   Return best partition
end
```

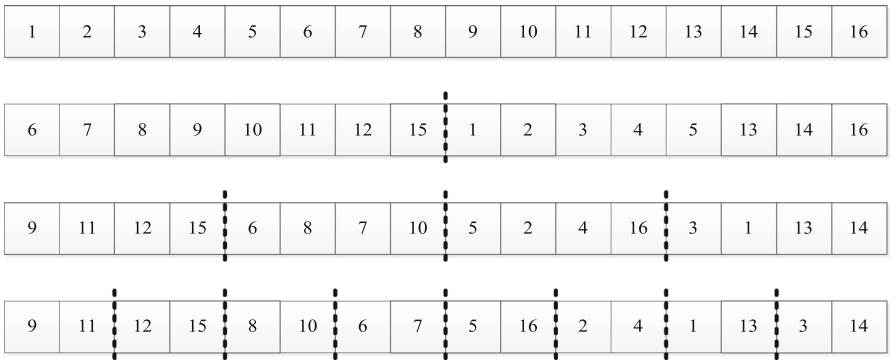

Fig. 2 The partition by the KL algorithm

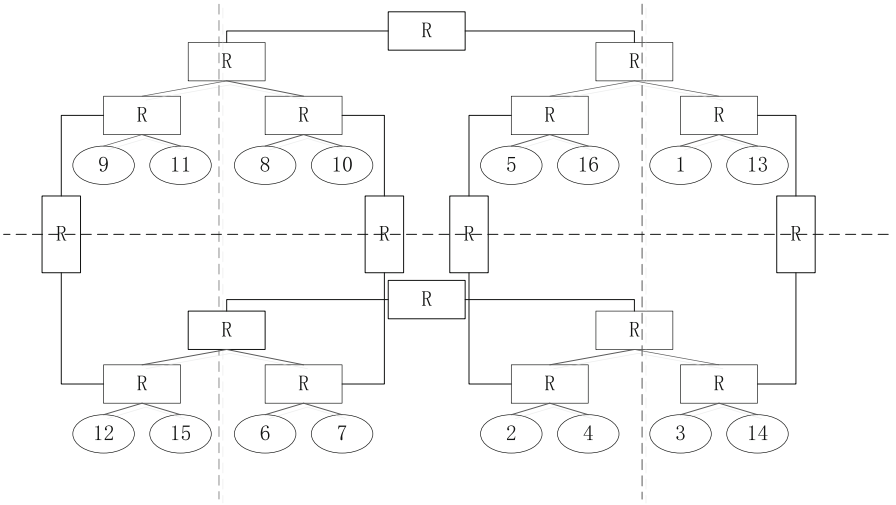

Fig. 3 Mapping onto a 4 × 4 MoT NoC

Figures 2 and 3 illustrate how the final partition generated from the KL algorithm can be mapped to the topology graph. We set this mapping solution as the current optimal one and then calculate the fitness function of this solution and store it.

Step2 Generate m initial mapping solutions as the current population. Here, we set m to be 20.

Step3 Get an optimal mapping solution q using the GA method and calculate the fitness function of q.

Step4 Determine whether the fitness value of q is larger than the current optimal solution s which is calculated in step1. If so, go to the next step; otherwise go to step6.

Step5 Set the current optimal solution $s = q$ and go to step3.

Step6 If the fitness value of q is smaller than the current optimal solution s, meaning the current population does not generate more optimal offspring, then we abandon this population, set n to $n + 1$ and go to step7.

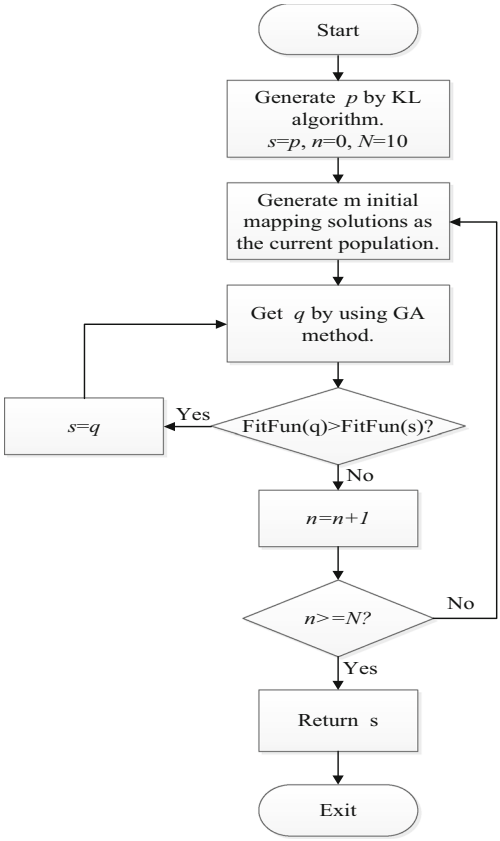

Fig. 4 The flow chart of our KL_GA algorithm

Step7 Compare n with is not larger than the total number of iterations N. If n is not larger than N, go to step2; otherwise go to the next step.
Step8 The optimal solution s is the final result of our algorithm.

Our GA method can be described as follows:
Step1 Initialize the current iteration count r to be 0 and the total number of iterations r_max to be 100.
Step2 Calculate the fitness value of each individual in the current population, sort them by their fitness values and record the current optimal solution q and two worst individuals $w1$ and $w2$.
Step3 Perform the selection procedure of GA. We define the top 50 % of the current population as the high-quality population. Then we select two individuals x and y from the high-quality population randomly.
Step4 Perform the crossover procedure of GA. We generate the number i randomly, which is no larger than the number of tasks. Starting from the gene i of x and y, we exchange all of the subsequent genes following it and generate the new individuals x' and y'. If the exchanged gene and the previous gene sequence are repeated, then we give up exchange in this position.

Step5 Perform the mutation procedure of GA. Generate the number j and k (no larger than the number of tasks) randomly, and then exchange the gene j and k of x' and y' with the probability of p to generate the new individuals x'' and y''. Here, we set $p = 0.2$.
Step6 Determine whether the fitness value of the new individuals x'' and y'' are larger than the individuals $w1$ or $w2$. if so, go to the next step; otherwise go to step8.
Step7 Replace $w1$ or $w2$ with x'' or y''.
Step8 Determine whether the value of r is greater than or equal to the value of *r_max.* If this is the case, go to the next step; otherwise go to step2.
Step9 Return the current optimal solution q as the final result.

The flow chat of these steps can be seen at Fig. 4. The complexity of genetic algorithm is $O(p^*c^*g)$, where p, c and g represent the population size, chromosome length and total number of iterations, respectively. Because our proposed algorithm is based on genetic algorithm, the complexity of our KL_GA is $O(p^*c^*n^*r)$, where p and c are same with the genetic algorithm. Since we add a jumping strategy to jump from local optimal solution, we break the total number of iterations into two parts. Let n and r represent the total number of external iterations and evolutionary generations of internal genetic algorithm. As $n^*r = g$, our KL_GA algorithm is in the same order of magnitude with the genetic algorithm.

5 Experimental setup

5.1 The evaluation model

We use the power consumption, network latency and instructions per cycle (IPC) to evaluate our proposed algorithm. The energy model is given in Sect. 3, which is an important indicator to evaluate the cost of the algorithm. The mapping algorithm includes optimization for power consumption as well as the network latency. Lower latency indicates higher network efficiency. The formula of network latency is described as

$$L_{avg} = \frac{\sum_{i=1}^{M} L_i}{M},$$

where M represents the number of received packets in certain cycles and L_i represents the network latency of the ith packet.

IPC represents the average number of instructions executed for every clock cycle, which can be used to evaluate the network performance. The formula of IPC is shown as follows:

$$IPC = \sum_{i=0}^{n-1} instruction_i / cycles.$$

5.2 The simulator and benchmarks

In our experiments, we used Gem5 as our simulator, which is an extensively configurable architecture simulator for computer system architecture-related research. The

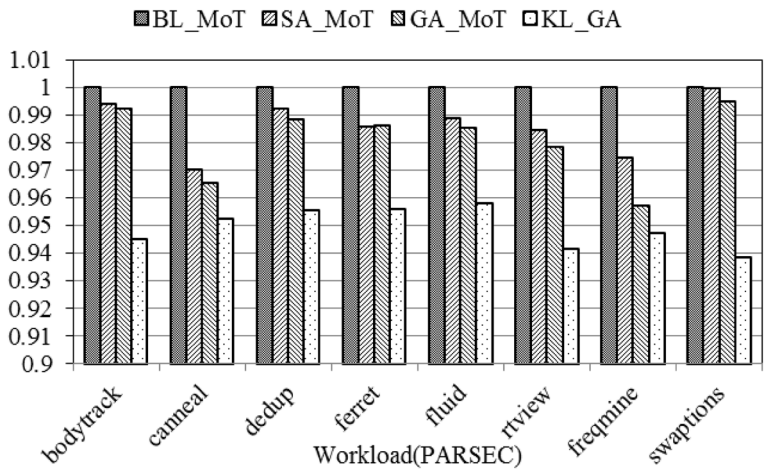

Fig. 5 Power consumption of 16 cores normalized to random mapping

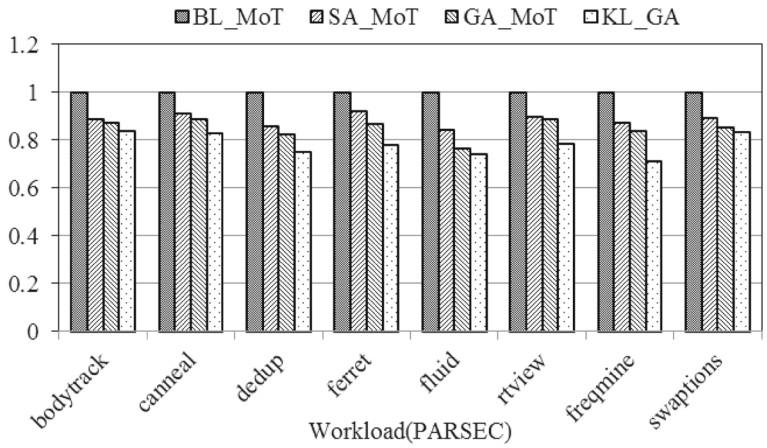

Fig. 6 Power consumption of 64 cores normalized to random mapping

Orion [24] model in Gem5 is used to evaluate the power consumption of the NoC. We used PARSEC [25] benchmarks in our experiments. The NoC topology we used is MoT topology, which is an irregular topology with better performance characteristics such as smaller diameter and lower node degree than the general mesh topology. We compare our KL_GA algorithm with several other algorithms on the MoT NoC architectures: (1) BL_MoT (the baseline): which maps the tasks onto the topology randomly; (2) SA_MoT: simulated annealing algorithm on the MoT structure; (3) GA_MoT: which is the conventional genetic algorithm on MoT structure; (4) KL_GA: our proposed algorithm on MoT structure. We set the total number of iterations N to 10 and set K to 5.

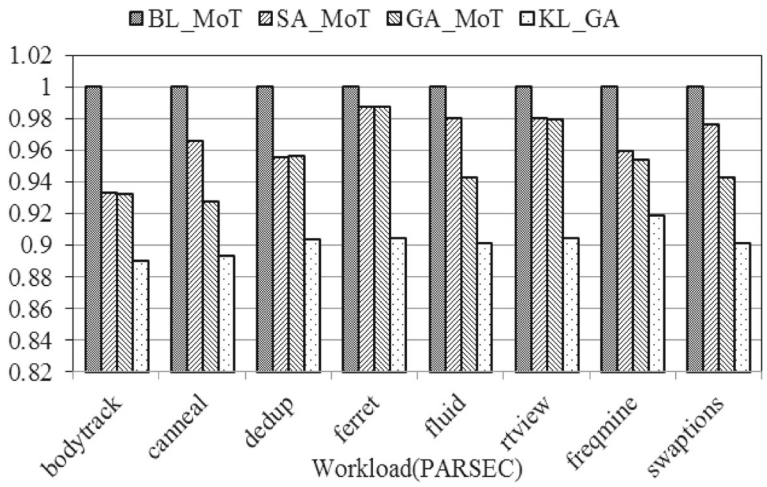

Fig. 7 Network latency of 16 cores normalized to random mapping

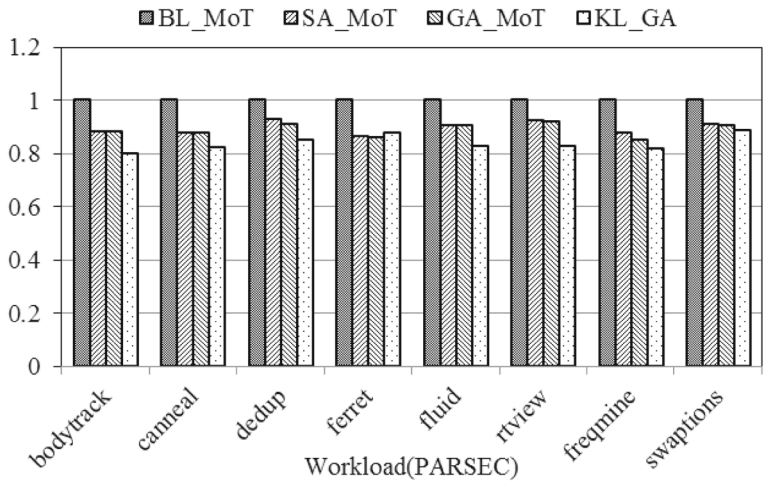

Fig. 8 Network latency of 64 cores normalized to random mapping

6 Experimental results

Figure 5 shows the experimental results of power consumption with 16 cores.

The power consumption is normalized to the random mapping algorithm. As shown in this figure, random mapping algorithm has the highest power consumption while the KL_GA consumes the least power, with an average of 5.1 % than the random mapping. Figure 6 shows the power consumption with 64 cores. In this case, the power savings of KL_GA is more significant than in the 16 core NoC shown in Fig. 5. Overall, KL_GA decreases power consumption by an average of 21.6 % compared to the baseline and achieves better performance compared to SA_MoT and GA_MoT.

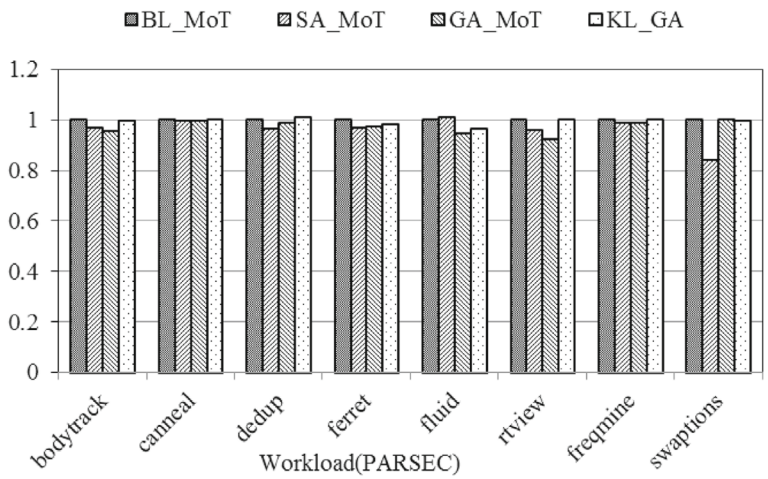

Fig. 9 IPC of 16 cores normalized to random mapping

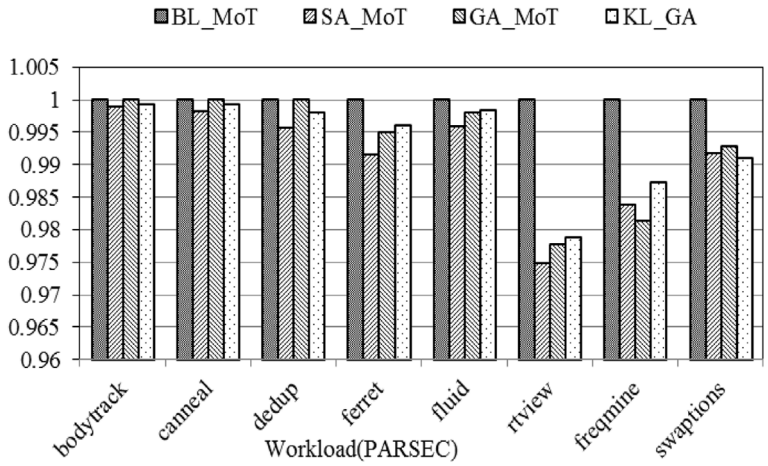

Fig. 10 IPC of 64 cores normalized to random mapping

The reason lies in the fact that KL_GA has a much lower chance to get trapped in local optimum because the jumping strategy was added in our algorithm. When the offspring are not better than the parents, the initial population is abandoned to stop from future iterative and evolutionary process which leads to worthless results. As a result, the solution generated by our KL_GA has less communication cost and lower power consumption.

The network latency normalized to the baseline case is shown in Figs. 7 and 8. The network latency shown here varies significantly, due to the varying traffic characteristics of the applications. For 16 cores, SA_MoT, GA_MoT and KL_GA decrease the network latency by average 3.3, 4.8 and 9.9 %, respectively, compared to the baseline. In the case of 64 cores, the differences between the mapping results of the algorithms

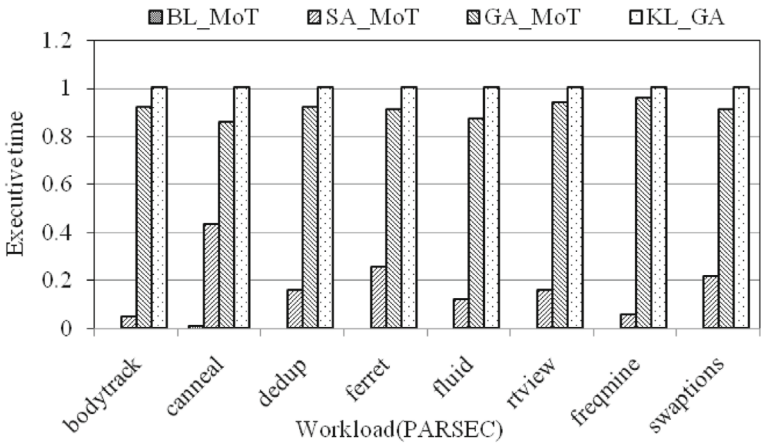

Fig. 11 Executive time of four mapping methods normalized to KL_GA for 16 cores

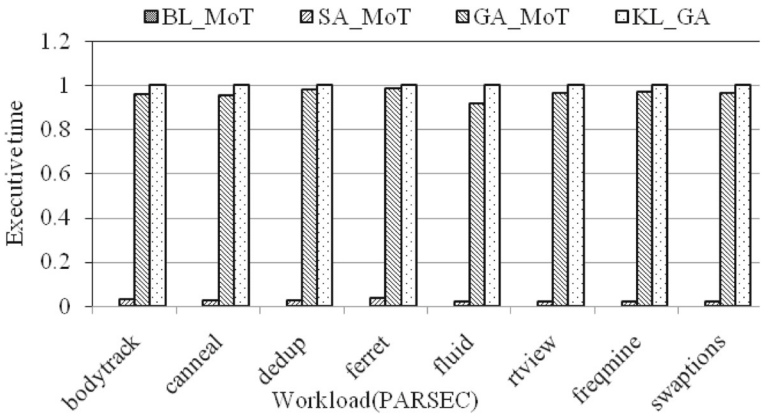

Fig. 12 Executive time of four mapping methods normalized to KL_GA for 64 cores

are more significant because the traffic loads between cores are greatly increased. Our proposed KL_GA algorithm decreases the network latency by an average of 16.3 % compared to the baseline as shown in Fig. 8.

It can be observed from Figs. 9 and 10 that for 16 cores and 64 cores, the difference between each algorithm is not obvious. For most cases, the decrease of the network performance is less than 2 %, which is negligible. This indicates that our proposed algorithm achieves almost the same network performance as other algorithms.

The comparison in execution time of the random, SA and GA mapping algorithms against KL_GA mapping is shown in Figs. 11 and 12. The size of the task graph we experimented is 16 and 64, respectively. Among the four mapping methods, KL_GA algorithm obtains the best result in reducing the communication cost and power consumption. As has been discussed above, the time complexity of KL_GA is the same with GA. When the size of the task graph is 16, it only takes minutes to determine the solutions, while it takes longer time to obtain the optimal solution with 64 nodes.

In summary, our proposed algorithm decreases the power consumption and network latency significantly compared to the random mapping algorithm. Comparing with the simulated annealing algorithm and conventional genetic algorithm, our KL_GA algorithm achieves better performance as well.

7 Conclusion

The design of the application mapping algorithms has become a hot topic in NoC related research. Several mapping algorithms have been proposed to lower power consumption, reduce network latency, achieve load balancing and minimize chip area and so on. By combining the KL algorithm and the genetic algorithm, we propose an algorithm, called KL_GA, to generate efficient mapping solutions. Our experimental results show significant reduction in the power consumption and the network latency.

Acknowledgments This work was supported by the National Natural Science Foundation of China (Grant Nos. 61202076, 60202062), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. They would also like to thank all the teachers and students in their laboratory for helpful discussions.

References

1. Dally WJ, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: Proceedings of design automation conference, Las Vegas, pp 684–689
2. Nurmi J (2005) Network-on-chip: a new paradigm for system-on-chip design. In: Proceedings of international symposium on system-on-chip, pp 2–6
3. Taylor MB, Lee W, Miller J et al (2004) Evaluation for the Raw microprocessor: an exposed-wire-delay architecture for ILP and streams. ACM SIGARCH Comput Archit News 32(2):2–13
4. Hoskote Y, Vangal S, Singh A et al (2007) A 5-GHz mesh interconnect for a teraflop processor. IEEE Micro 27(5):51–61
5. Shao J, Tian-Zhou C, Liu L (2012) Incremental run-time application mapping for heterogeneous network on chip. In: Proceedings of 14th IEEE international conference on high performance computing and communications (HPCC)
6. Wang J, Li Y, Chai S, Peng Q (2011) Bandwidth-aware application mapping for NoC-based MPSoCs. J Comput Inf Syst 7(1):152–159
7. Çelik C, Bazlamaçci CF (2013) Energy and buffer aware application mapping for networks-on-chip with self similar traffic. J Syst Archit 59(10):1364–1374
8. Tosun S (2011) New heuristic algorithms for energy aware application mapping and routing on mesh-based NoCs. J Syst Archit 57(1):69–78
9. Zhu D, Chen L, Yue S, Pedram M (2014) Application mapping for express channel-based networks-on-chip. In: Proceedings of design, automation and test in Europe (DATE)
10. Sahu PK, Manna K et al (2014) Extending Kernighan-Lin partitioning heuristic for application mapping onto network-on-chip. J Syst Archit 60(7):562–578
11. Shah N, Manna K, Chattopadhyay S (2011) An application mapping technique for butterfly-fat-tree network-on-chip. In: Proceedings of 2nd international conference on emerging applications of information technology (EAIT), pp 383–386

12. Tosun S (2011) Cluster-based application mapping method for network-on-chip. Adv Eng Softw 42(10):868–874
13. Coskun Ç, Cüneyt FB (2013) Energy and buffer aware application mapping for networks-on-chip with self similar traffic. J Syst Archit 59(10):1364–1374
14. Wang J, Li L, Wang Z et al (2014) Energy-efficient mapping for 3D NoC using logistic function based adaptive genetic algorithms. Chin J Electron 23(2):254–262
15. Zang M, You H (2012) Low power NOC process element mapping using genetic algorithm. J Inf Comput Sci 9(3):557–563
16. Khalili F, Zarandi HR (2013) A reliability-aware multi-application mapping technique in networks-onchip. In: Proceedings of 21st Euromicro conference on parallel distributed and network-basedprocessing. IEEE, pp 478–485
17. Jing N, He W, Mao Z (2011) A general statistical estimation for application mapping in network-on-chip. In: Proceedings of IEEE/IFIP 19th international conference on VLSI and system-on-chip (VLSI–SoC), pp 172–175
18. Yu H, Ha Y, Veeravalli B (2010) Communication-aware application mapping and scheduling for NoC-based MPSoCs. In: Proceedings of IEEE international symposium on circuits and systems: nano-bio circuit fabrics and systems, pp 3232–3235
19. Zhang X, Chen X, Phillips C (2012) Achieving effective resilience for QoS-aware application mapping. Comput Netw 56(14):3179–3191
20. Beltrame G, Sciuto D, Silvano C, Paulin P, Bensoudane E (2006) An application mapping methodology and case study for multi-processor on-chip architectures. In: Proceedings of IFIP WG 10.5 international conference on very large scale integration and system-on-chip, pp 146–151
21. Ghosh A, Paul S, Bhunia S (2012) Energy-efficient application mapping in FPGA through computation in embedded memory blocks. In: Proceedings of the IEEE international conference on VLSI design, pp 424–429
22. Sahu PK, Sharma A, Chattopadhyay S (2012) Application mapping onto mesh-of-tree based network-on-chip using discrete particle swarm optimization. In: Proceedings of international symposium on electronic system design, pp 172–176
23. Hu J, Marculescu R (2005) Communication and task scheduling of application-specific networks-on-chip. IEEE Proc Comput Digit Tech 152(5):643–651
24. Kahng AB, Li B, Peh LS et al (2012) ORION 2.0: a power-area simulator for interconnection networks. IEEE Trans Very Large Scale Integr Syst 20(l):191–196
25. Bienia C, Li K (2009) Parsec 2.0: a new benchmark suite for chip-multiprocessors. In: Proceedings of the 5th annual workshop on modelling, benchmarking and simulation