PSO is widely used in various fields of social life and scientific research. Guangqing Bao in Lanzhou University of Technology applied PSO algorithm in wind power system [23]. In addition, PSO has also been applied to the optimize scheduling of the reservoir [24] and vehicle routing problem [25]. Li Jiang in Anhui University applied PSO and SA to study of BP network learning method [26]. In this paper, we first apply the classical PSO together with SA to 3D NoC floorplanning algorithm and open a new field of PSO and SA application.

The rest of the paper is organized as follows. In Section 2, we specify the design and realization of PSO-SA-NoC algorithm. In Section 3, we conduct simulation experiments using VNOC3 Simulator to test the degree of improvement of the proposed algorithm on CPU processing time, average flit latency and network throughput. Finally, Section 4 is the conclusion of this paper and the future work we can do.

2. FLOORPLANNING ALGORITHM BASED ON PARTICLE SWARM OPTIMIZATION ALGORITHM NESTING SIMULATED ANNEALING ALGORITHM TO OPTIMIZE THE FLOORPLANS

2.1. PSO-SA-NoC Algorithm Design

In this paper, an improved floorplanning algorithm named the floorplanning algorithm based on particle swarm optimization algorithm nesting simulated annealing to optimize the floorplans (PSO-SA-NoC) has been proposed. The algorithm is an improvement of the original floorplanning algorithm based on Simulated Annealing (SA) algorithm to optimize the floorplans, using both the parallel computing features of PSO and the global optimization features of SA to mainly optimize average flit latency and network throughput.

The cost function of PSO-SA-NoC algorithm used is shown in Formula (1),

$$Costfunction = \alpha \cdot Area + (1-\alpha)WireLength \tag{1}$$

In (1), *Costfunction* represents the cost of completing the layout needs. *Area* represents the layout area used by IP cores layout using. *Length* represents length of the connection of generating layout uses. α is a parameter specified by the user which used to balance the area of tiles and the length of layout uses and valued from 0 to 1. In our algorithm, the value of α is 0.25. The update formulae of velocity and position in our algorithm are shown as Formula (2) and Formula (3),

$$v_{i+1} = \omega v_i + c_1 \cdot d_1(pre\_cost_i - x_i) + c_2 \cdot d_2(best_i - x_i) \tag{2}$$

$$x_{i+1} = x_i + v_i \tag{3}$$

In Formula (2), inertia weight ω uses variable weight and calculated formula is shown as Formula (4). The initial value of ω_{min} and ω_{max} are 0.5 and 3. The parameter *count* represents the current number of iterations and *N* represents the maximum number of iterations. Learning factor c_1 and c_2 use synchronous time-varying way and the calculated formula is shown as Formula (5). The initial value of c_{min} and c_{max} are 0.25 and 3. The parameter *count* represents the current number of iterations and *N* represents the maximum number of iterations. Random parameters d_1 and d_2 are numbers that program randomly generated from 0 to 1. The parameter pre_cost_i represents the best position of the particle itself and $best_i$ represents the optimum position of particle swarm, wherein the velocity is in the range [3, 3]. If the value exceeds the determined range, then revise it to equal to the maximum. The position sequence of particles satisfies with binary code. It also needs to be revised if out of the determined range. If the location information obtained is less than 0.5, then record it as 0, else record it as 1.

$$\omega = \omega_{\max} - \frac{(\omega_{\max} - \omega_{\min}) * count}{N} \tag{4}$$

$$c_1 = c_2 = c_{\max} - \frac{(c_{\max} - c_{\min}) * count}{N} \tag{5}$$

The simulated annealing parameters used in this algorithm include annealing probability *p* and the calculated formula is shown as Formula (6), in which *pre*[*i*] represents the fitness value of each particle and *T* represents the initial temperature.

$$p = \exp(\frac{pre[i] - pre\_cost}{T}) \qquad i \in [0, shu] \tag{6}$$

The given initial value of final temperature *term_temp* is 0.1. The annealing formula is shown as Formula (7), in which *r_t* represents temperature control parameter, λ is specified by users and the default value is 1.3, and *sv* represents the length of number of particles which have calculate fitness value.

$$T = r\_t * T, \quad r\_t = \exp(\frac{\lambda * T}{sv}) \tag{7}$$

2.2. PSO-SA-NoC Algorithm Realization

In this part, we show the realization of PSO-SA-NoC algorithm. The steps are as follows.

Step 1: Parameters initialization. Define the number of particles as *shu* and the maximum number of iterations as *N*, where *N=times*fp_p->*size(). Randomly generate *shu* initial particles X_0 and velocity V_0 which are from -3 to 3. The initial temperature is T. The temperature control parameter is *r_t* and annealing probability *p*.

Step 2: Assume the best position of the particle itself is *pre_cost* and the global best position of particle swarm is *best*.

Step 3: Decide whether the current temperature reaches the final temperature *term_temp*. If achieved, then jump to Step 6, else jump to step 4.

Step 4: Do the following for all particles:

⊖ Calculate the fitness of each particle according to Formula (1). If the particle adapt better than *pre_cost* or the random number is less than annealing probability *P* which can be calculated by Formula (4), Formula (5) and Formula (6), then the value is assigned to *pre_cost* and update the best position of the individual particle.

Table 1. Characteristics of the used testcases.

Testcases	Number of IP/cores	Core avg. W/H	Core std. dev. of W/H	Direct Topology R×R
apte	8	4324/2499	27/4	3×3
xerox	10	2114/2872	335/1290	4×4
hp	11	4533/924	2498/386	4×4
ami25	25	1770/1408	1201/896	5×5
ami33	33	1581/1573	830/865	6×6
ami49	49	1089/1123	768/651	7×7

⊖ If the fitness of *pre_cost* is better than *best*, then update the value of *best* and the value of *goodnum* adds to 1, else the value of *badmum* adds to 1.

⊛ Update the position and velocity of particles according to Formula (2) and Formula (3).

④ Correct the obtained position information of particles according to the determined range of value of position and velocity so that it would not exceed the available space.

Step 5: Update temperature value according to Formulate (7) and jump to Step 3.

Step 6: Output *best* and end the algorithm.

3. SIMULATION RESULTS AND DISCUSSION

3.1. Introduction of the Simulator

In order to test the degree of improvement of the proposed algorithm on CPU processing time, average flit latency and network throughput, the algorithm needs to be run on an actual system and compared with the original algorithm.

In this paper, we use the VNOC3 simulation platform developed by Cristinel Ababei in North Dakota State University using C++ in the Linux system. The simulator is actually a platform used to study 3D NoC architecture with two or three layers. The platform is built on a previous version of VNOC (a simulator for 2D NoC) and a B* tree layout. For three-layer architecture, the framework uses hMetis division. In addition, the simulator also includes a hidden option which allows users to generate new testcases and a GUI drawing tools. The drawing tools can be used to generate a three-layer network architecture but with only 2D display.

3.2. Testcases

In our experiments, we used six testcases whose characteristics are shown in Table **1**. In this table, we also present the size of the direct topologies for all the 3d NoC testcases with heterogeneous network architecture. We constructed these testcases from the classic MCNC testcases, whose area was scaled to achieve an average size of about 1cm×1 cm, which is a typical area for NoCs reported in the literature [27-29].

Based on these six testcases we have done a lot of experiments and calculated the average as the experimental results. But due to the limitation of paper space, we show only three testcases of the six to illustrate trends and the improvement of the performance. They are *apte*,*hp* and *ami49*.

We choose *apte* and *ami49* here since they respectively are the smallest and largest testcases in these six testcases and more persuasive. The testcase *hp* is special, because the number of its IP core is relatively small (only 11). But its aspect ratio is 2 to 3 times the other testcases. So in this paper, we show comparison of 3 testcases.

3.3. Simulation Results and Analysis

With the simulator introduced above we implement PSO-SA-NoC and compare with the original floorplanning algorithm based on simulated annealing algorithm to optimize the floorplans (hereinafter referred to as SA-NoC). In the simulation, the performance of the algorithm is mainly measured by three parameters, *i.e.* CPU processor time, throughput and average flit latency. In the simulation experiments, in order to ensure a fair comparison, the value of command line parameters (for example, operating cycle, preheat cycle, input and output buffer size, *etc*) of the two algorithm is always the same. In certain load conditions or certain Mesh size, the lower the average latency, higher the throughput, less the CPU processing time, the better performance of the algorithm.

In order to comprehensively study the impact of each indicator of 3D NoC on latency and throughput, in the simulation experiment, we compare the throughout and average latency from four aspects while other parameter values unchanged. (1) Change the number of virtual channels (from 2 to 6) and analyze the impact of virtual channels on average flit latency, throughput and CPU processing time. (2) Change the buffer size (from 1x to 5x and the data indicates 1 to 5 times the default cache) and analyze the impact of buffer size on average flit latency and throughput. (3) Change the injection load (20%-100%) and analyze the changes of CPU processing time, average flit latency and throughput. (4) Change Mesh size of the architecture (number of IP cores and aspect ratio) and analyze their impact on CPU processing, average flit latency and throughput. Detailed comparison and analysis are seen in section A, B, C, D. In the experiment, a number of simulations have been

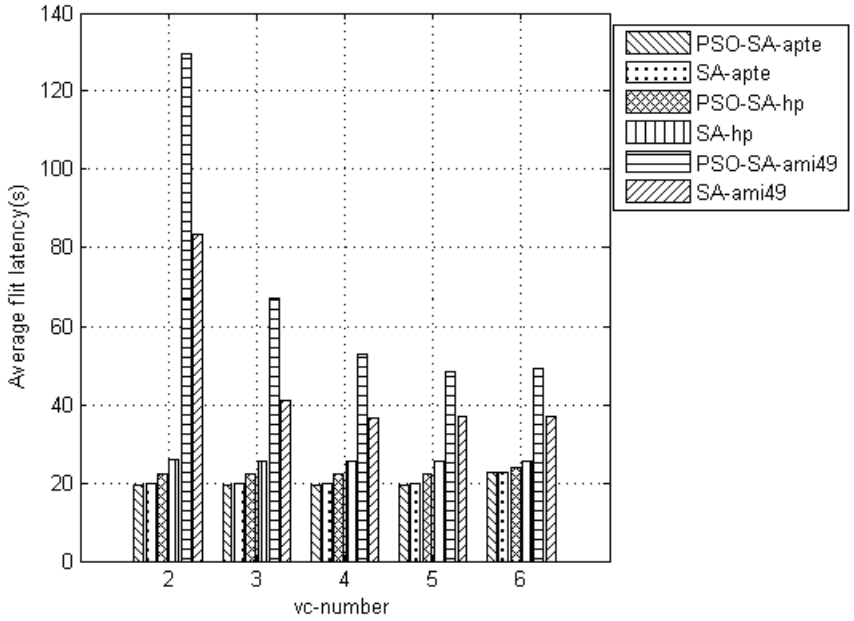

Fig. (1). Comparison of average flit latency for *apte*, *hp* and *ami49* while changing the number of virtual channels from 2 to 6.

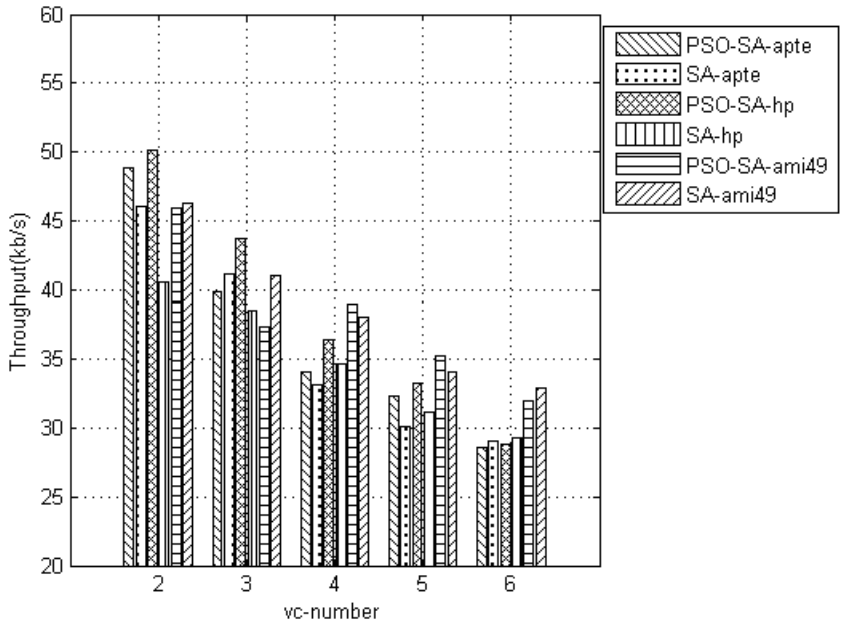

Fig. (2). Comparison of throughput for *apte*, *hp* and *ami49* while changing the number of virtual channels from 2 to 6.

conducted in each case and the final result is calculated from average. In each simulation, total run cycle is set to 60000cycles and preheat cycle is set to 1000cycles to ensure the obtained data values when the system reaches a steady state to reduce the error data obtained in the simulation.

3.3.1. The Impact of the Number of Virtual Channels on Performance

First, we observe the impact of increasing the number on virtual channels upon average flit latency, throughput and CPU processing time. In this scenario, we observe changes of average flit latency, throughput and CPU processing time while changing the number of virtual channels (VC) from 2 to 6 and compare the changes when using PSO-SA-NoC and SA-NoC.

For the testcases *apte*, *hp* and *ami49*, the comparison of average flit latency, throughput and CPU processing time of PSO-SA-NoC and SA-NoC are shown in Figs. (**1**-**3**). From the three figures, we can observe that for the testcase *apte*, compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm decreases by 2.44% in average and the network throughput improves by 2.90% in average. The most obvious is that the CPU processing time decreases by 20.98% in average which saves energy greatly.

For the testcase *hp*, with these three figures, we can conclude that compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm decreases by 13.41% in average and the network throughput improves by 9.57% in average. The most obvious is that the CPU processing time decreases by 75.81% in average which saves energy greatly.

For the testcase *ami49*, we can observe that compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm increases by 45.34% in average and the network throughput decreased by 1.31% in average. The CPU processing time decreases by 21.91% in average which saves energy greatly.

In summary, it can be seen that compared with the original SA-NoC, average flit latency and throughput of PSO-SA-NoC are much better when the size of mesh is not very large and do not improved when the size of mesh is large. However, CPU processing time decreases greatly in all cases, by an average of 39.57% and the maximum case decreases 75.81%.

3.3.2. The Impact of Buffer Size on Performance

In this scenario, instead of increasing the number of virtual channels, we study the impact of buffer size on performance by increasing the size of buffer. We assume that the areas of routers is about 20% of the total area, we can increase the areas of each router to reach to 5 times of the original value by increasing the size of input buffer and output buffer in each router port. We can observe the impact of

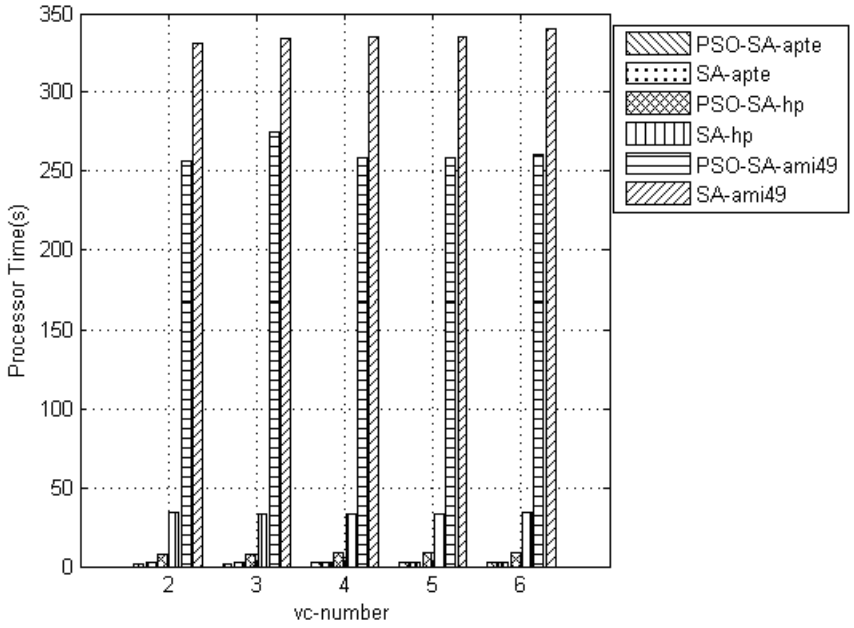

Fig. (3). Comparison of CPU processing time for *apte*, *hp* and *ami49* while changing the number of virtual channels from 2 to 6.

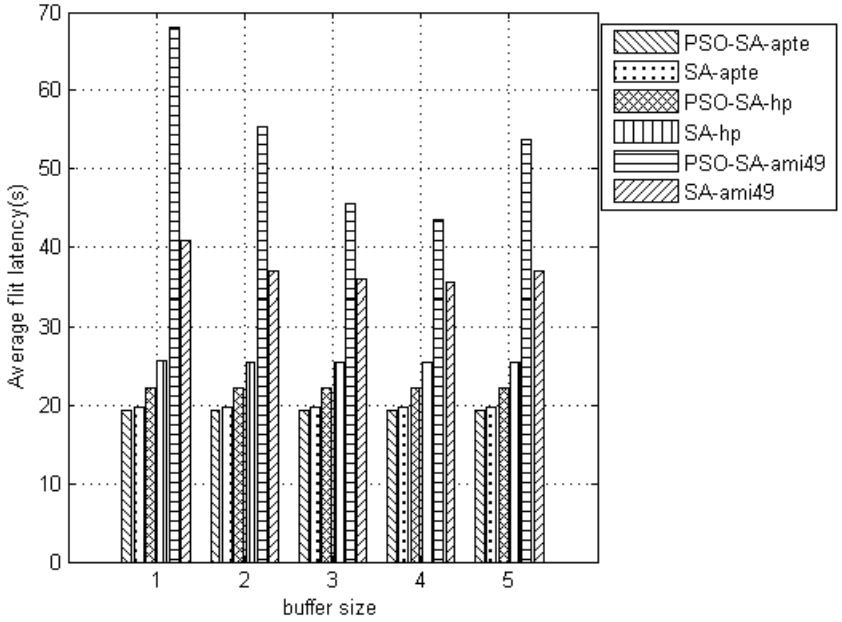

Fig. (4). Comparison of average flit latency for *apte*, *hp* and *ami49* while changing the size of buffers from 1x to 5x.

buffer size on average latency and throughput by changing the size of buffers.

For the testcases *apte*, *hp* and *ami49*, the comparison of average flit latency and throughput of PSO-SA-NoC and SA-NoC are shown in Figs. (**4** and **5**). From the two figures, we can observe that for the testcase *apte*, compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm decreases by 2.39% in average and network throughput improves by 4.36% in average.

For the testcase *hp*, with the two figures, we can conclude that compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm decreases by 13.07% in average and the network throughput improves by 50.64% in average. In this case, every performance of PSO-SA-NoC algorithm is much better than the original SA-NoC algorithm.

For the testcase *ami49*, we can observe that compared with SA-NoC algorithm, the average flit latency of PSO-SA-NoC algorithm increases by 41.98% in average and the network throughput decreased by 1.14% in average.

In summary, it can be seen that compared with the original SA-NoC, all performances of PSO-SA-NoC improve obviously in all cases when buffer size changes.

3.3.3. The Impact of Injection Load on Performance

Through a lot of experiments, we can conclude that the impact of changing the injection load on network latency and throughput is the largest. To compare the performance of the

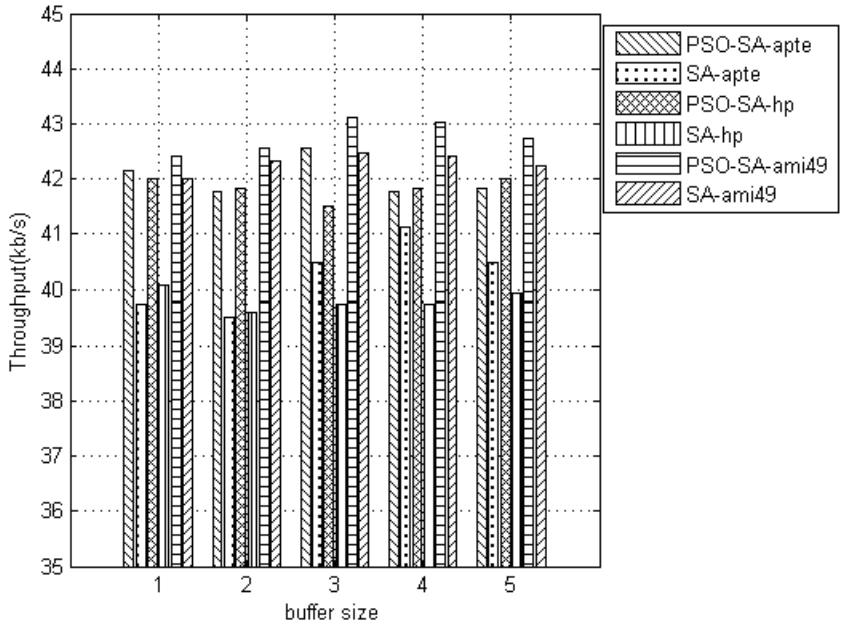

Fig. (5). Comparison of throughput for *apte*, *hp* and *ami49* while changing the size of buffers from 1x to 5x.

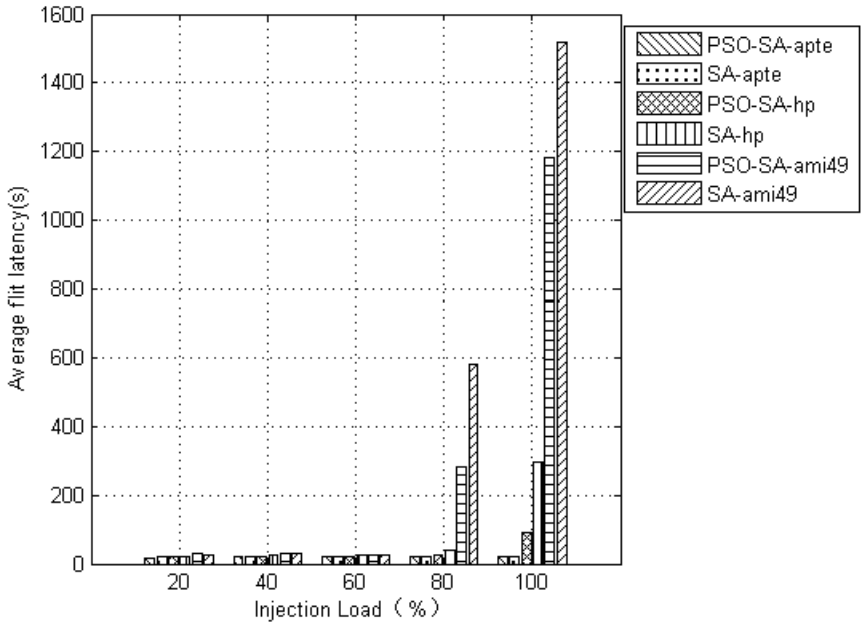

Fig. (6). Comparison of average flit latency for *apte*, *hp* and *ami49* while changing injection load.

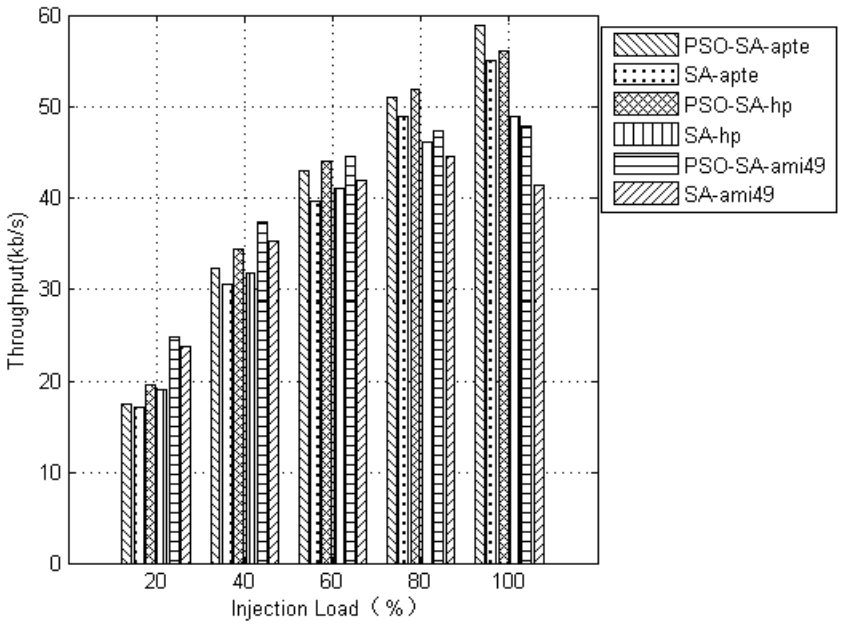

Fig. (7). Comparison of throughput for *apte*, *hp* and *ami49* while changing injection load.

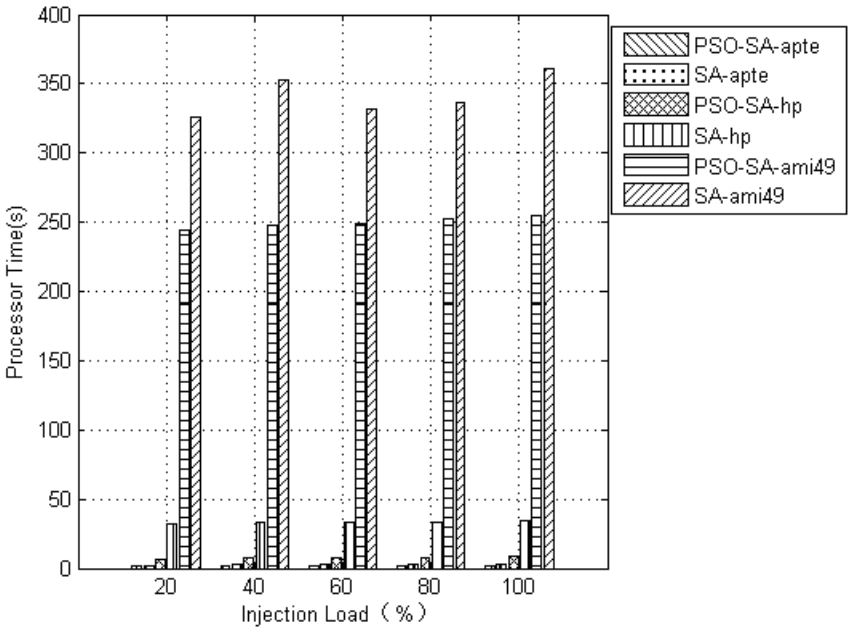

Fig. (8). Comparison of CPU processing time for *apte*, *hp* and *ami49* while changing injection load.

algorithm we proposed with the original algorithm, we need to explain that by changing injection load. In this paper, we conducted lots of experiments for the six testcases and eventually reach the following results by taking the average.

For the testcase *apte*, *hp* and *ami49*, the comparison of average flit latency, throughout and CPU processing time of the two algorithms are shown in Figs. (**6-8**). From the three figures, we can observe that for the testcase *apte*, compared with SA-NoC algorithm, CPU processing time of PSO-SA-NoC algorithm decreases by 25%, the average flit latency decreases by 2.75% network throughput improves by 6.28%.

For the testcase *hp*, before injection load reaches 70%, average flit latency has no change substantially. With the three figures, we can conclude that compared with SA-NoC algorithm, CPU processing time of PSO-SA-NoC algorithm decreases by 77.24%. Average flit latency increases by 28.33% and the network throughput improves by 9.77%. In summary, for network architecture with smaller scale or larger aspect radio, PSO-SA-NoC algorithm improves performance better.

For the testcase *ami49*, we can observe that compared with SA-NoC algorithm, CPU processing time of PSO-SA-NoC algorithm decreases by 25.09%. Average flit latency decreases by 18.89% and network throughput improves by 6.86%. In this case, every performance of PSO-SA-NoC algorithm is better than the original SA-NoC algorithm.

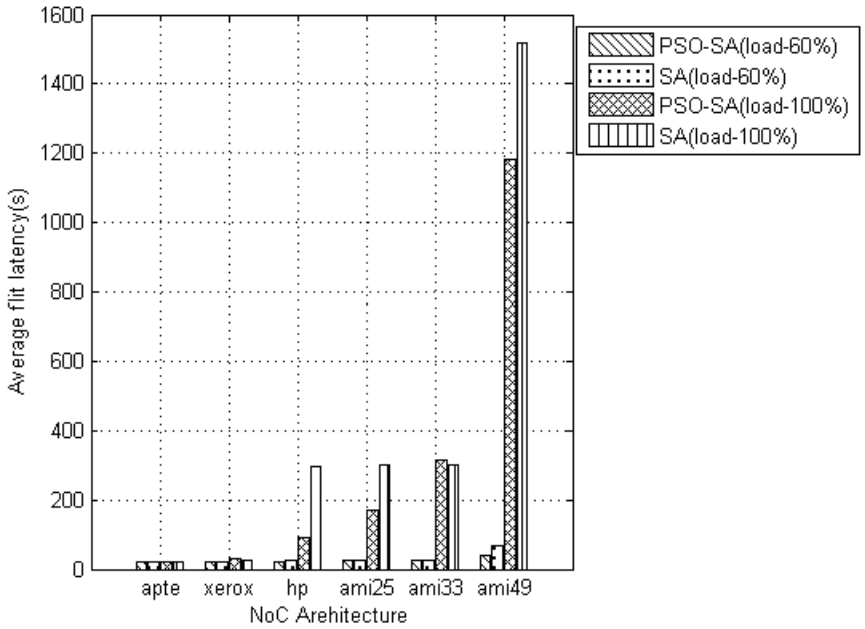

Fig. (9). Comparison of average flit latency of architectures with different Mesh size when injection load is 60% and 100%.

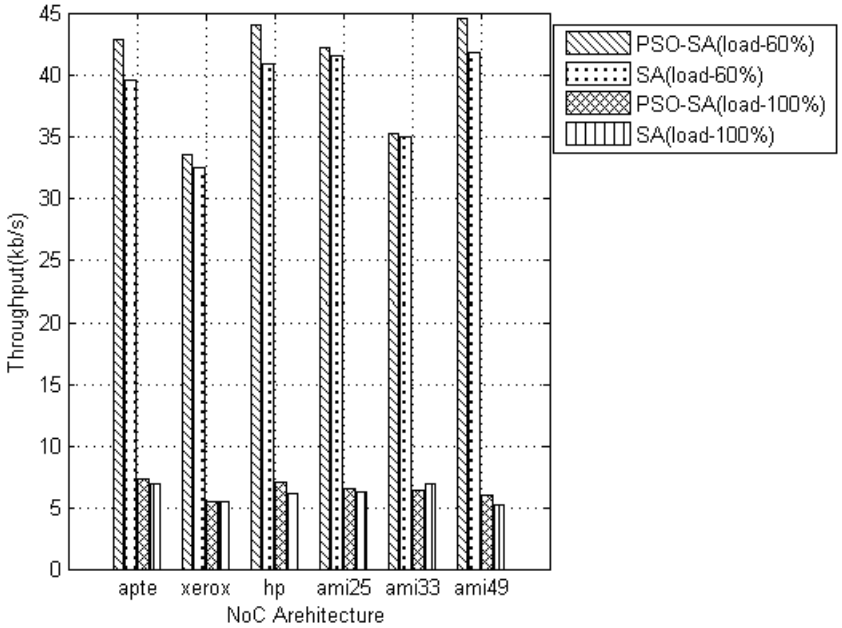

Fig. (10). Comparison of throughput of architectures with different Mesh size when injection load is 60% and 100%.

In summary, it can be seen that compared with the original SA-NoC, CPU processing time of PSO-SA-NoC algorithm decreases greatly, by an average of 35.39% and the maximum case decreases 78.59%. Average flit latency decreases 11.18% in average and 74.55% in the best case. Network throughput increases 6.17% in average and 24.02% in the best case.

3.3.4. The Impact of Different Testcases on Performance

Different testcases have different numbers of IP cores and different aspect ratio of the IP cores. Also, the Mesh size of their direct topologies is different. In this section, we study the change of average flit latency, throughput and CPU processing time of different testcases. In order to get more equitable results, in this paper, we do research and simulation of different injection loads and the results are as follows. When injection loads are different, the average flit latency increases as the number of IP cores increases, and the throughput is controlled by two parameters of Mesh size and aspect ratio. The variation trend of CPU processing time is substantially the same.

In this paper, we do simulation experiments of injection load of 20% to 100%, and then we get the line chart and analyze it. But due to the limitations of our paper space, we just show the comparison of average flit latency in the condition of 60% and saturation injection load as shown in Fig. (**9**). It can be observed in the figure that when the Mesh size is less

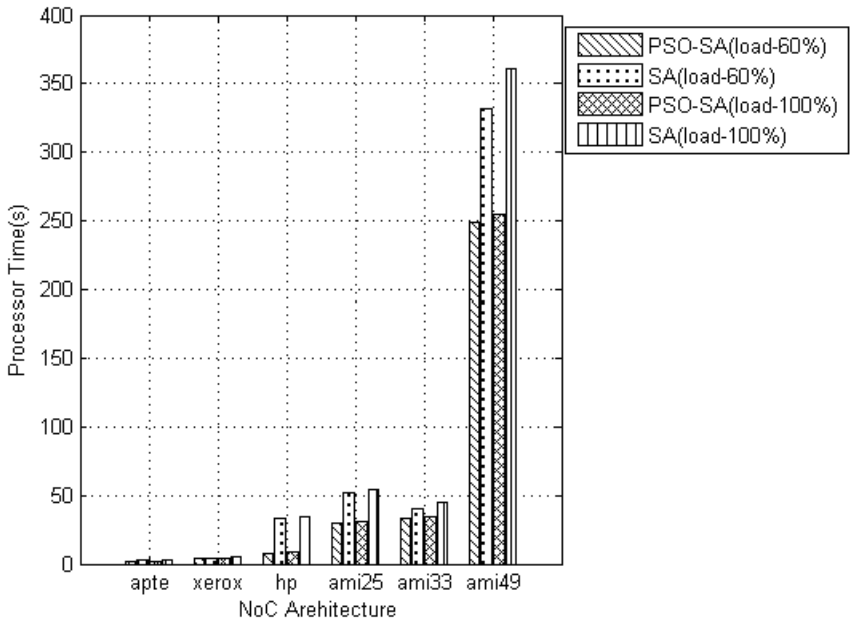

Fig. (11). Comparison of CPU processing time of architectures with different Mesh size when injection load is 60% and 100%.

than 6, average flit latency increases more slowly and the average flit latency of PSO-SA-NoC algorithm is better than that of SA-NoC algorithm. When Mesh size is over 6, average flit latency of SA-NoC algorithm increases greater while that of the proposed PSO-SA-NoC algorithm increases slowly.

To illustrate the impact of different testcases on network throughput, in this paper, we also show the comparison of the impact of different Mesh size and respect ratio on throughput in the condition of 60% and saturation injection load as shown in Fig. (**10**). It can be observed in the figure that as a whole the throughput does not just increase or decrease as the number of IP cores increase, but shows a growth trend like a wavy line. This also explains why the throughput is not only affected by the number of IP cores but also the aspect ratio. According to the configuration parameters of each testcase shown in Table **1**, we can draw a conclusion that the throughput is proportional to the number of IP cores and the aspect ratio. Therefore, although the number of IP cores of *xerox* increases, the aspect ratio decreases close to 1. So the throughout presents a certain downward trend. For comparison of the two algorithms, the throughput of PSO-SA-NoC algorithm is slightly higher than that of SA-NoC algorithm as a whole. In effect overall, PSO-SA-NoC algorithm is better.

For CPU processing time, in this paper, we just show the simulation results when the injection load is 60% and saturation injection load as shown in Fig. (**11**). It can be observed in the figure that when the Mesh size is less than 6, CPU processing time increases more slowly. When Mesh size is over 6, CPU processing time of SA-NoC algorithm increases at a speed of 7 times while that of PSO-SA-NoC algorithm increasing at a speed of 3 to 5 times. As a whole, compared with the original algorithm, CPU processing time of PSO-SA-NoC algorithm decreases especially for architectures with large scale.

In summary, compared with SA-NoC algorithm, CPU processing time of PSO-SA-NoC algorithm decreases by 35.39%. The network throughput increases and for architectures with Mesh size less than 6, the average flit latency decreases.

CONCLUSION

In this paper, we propose an improved floorplanning algorithm named the algorithm based on particle swarm optimization algorithm nesting simulated annealing algorithm to optimize the floorplans (PSO-SA-NoC). The algorithm is based on the advantages of parallel processing units of particle swarm optimization algorithm, and optimizes layout of the tiles to make the floorplanning path and the CPU processing time shorter and more efficient. Also, we adapt and build the original 3D NoC simulator to simulate the proposed algorithm and compare it with the original one. We add calculation method of throughput to make the experimental data more comprehensive and persuasive.

In the future, we need to design NoC architecture based on heterogeneous network architecture to replace original floorplanning algorithm. Secondly, it is helpful for a comprehensive study of architectures to write graphical interface that can display in three dimensions. In addition, 3D NoC architecture based on heterogeneous layout is from tree graph mapping. To ensure the quality of the mapping, researchers need to design better mapping algorithm to improve floorplanning performance of 3D NoC.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (NSFC) (61272006).

REFERENCES

[1] M. Sarrafzadeh, "Transforming an arbitrary floorplan into a sliceable one", In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, Santa Clara, CA, USA, 1993.

[2] Y. Jin-Tai, L. Kai-Ping, and H. Chun-Tsai, "Sliceable transformation of non-slicing floorplans based on vacant block insertion in LB-packing process", In: *IEEE International 48th Midwest Symposium on Circuits and Systems*, Covington, KY, USA, 2005.

[3] Y. Jin-Tai, C. Chih-Wei, Y. –F. Luo, C. Yi-Hsiang, "Packing-driven sliceable transformation for 3D floorplan designs", In: *Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*, Toulouse, France, 2008.

[4] O. Oluwaseun, "Parallel implementation of non-slicing floorplan with MPI and OpenMP", MS thesis, Ryerson University, 2012.

[5] Y. M. Li, "An non-slicing area prejudged algorithm for floorplanning without simulated annealing", In: *2nd International Conference on Computer and Automation Engineering*, Singapore, 2010.

[6] C. Yu-Ning, "Non-slicing floorplanning-based crosstalk reduction on gridless track assignment for a gridless routing system with fast pseudo-tile extraction", In: *ACM International Symposium on Physical Design*, New York, NY, USA, 2008, pp. 134-141.

[7] X. Ning, "Hybrid algorithm for non-slicing floorplans optimization", In: *9th International Conference on Solid-State and Integrated-Circuit Technology*, Beijing, 2008, pp. 2313-2316.

[8] H. –J. Bai, S. –Q. Dong, and X. –L. Hong, "Buffer insertion based on single-pair shortest-path algorithm for interconnect-centric floorplanning", In: *8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, Shanghai, 2006, pp. 1873-1875.

[9] X. Hong, L. Ma, Y. Cai, C. K. Cheng, and J. Gu, "Sequence cloth diagram of Angle module and boundary constraint layout planning algorithm based on angle module expressed in sequence", *Science China*, vol. 32, no. 3, pp. 409-418, 2002.

[10] W. Haiqi, and D. Sheqin, "Topology generation algorithm for application specific network on chip", *Journal of Computer Aided Design & Computer Graphics*, vol. 23, no. 9, pp. 1576-1584, 2011.

[11] H. Liang-li , W. Fa-yuan , and W. Feng-jun, "A method based on flock of birds with human-computer technology for packing layout", *Journal of China Academy of Engineering Physics*, vol. 3, pp. 29-31, 2009.

[12] E. F. Y. Young, and T. Ma, "TCG-based multi-bend bus driven floorplanning", In: *Proceedings of the Asia and South Pacific Design Automation Conference*, Seoul, Korea, 2008, pp. 192-197.

[13] E. F. Y. Young, and R. Wang, "3-D floorplanning using labeled tree and dual sequences", In: *Proceedings of the International Symposium on Physical Design*, Seoul, Korea, 2008, pp. 54-59.

[14] J. He, "*The Research and Development of Placement Algorithm for Network on Chip*", PhD thesis, Wuhan University of Technology, 2010.

[15] K. M. Reza, A. Federico, M. Srinivasan, P. Antonio, S. Ciprian, and B. Luca, "A floorplan-aware interactive tool flow for NoC design and synthesis", In: *Proceedings of IEEE International SOC Conference*, Belfast, Northern Ireland, UK, 2009, pp. 379-382.

[16] V. De Paulo, and C. Ababei, "A framework for 2.5D NoC exploration using homogeneous networks over heterogeneous floorplans", In: *International Conference on ReConFigurable Computing and FPGAs*, Cancun, Quintana Roo, Mexico, 2009, pp. 267-272.

[17] D. Wang, J. Wang, and H. Wang, '*Intelligent Optimization Algorithms'*. Higher Education Press, 2007, pp. 136-137.

[18] Y. Feng, "*Research and application of simulated annealing algorithm*", PhD thesis, Kunming University of Science and Technology, 2005, pp. 1-2.

[19] L. Wang, and Y. Qi, 'Application of simulated annealing algorithm to optimize the design of t-shaped micro-reactor', *Computer Application and Chemistry*, vol. 10, no. 30, pp. 1193-1196, 2013.

[20] Z. Liu, C. Liu, X. Kuang, and D. Zhou, "Application of simulated annealing algorithm to assessment of dynamic positioning capability", *Journal of Ship Mechines*, vol. 4, no. 17, pp. 375-381, 2013.

[21] J. Kennedy, and R. Eberhart, "Particle swarm optimization", In: *IEEE International Conference on Neural Networks Proceedings*, Perth, WA, Australia, 1995, pp. 1942-1948.

[22] R. Eberhart, and J. Kennedy, "A new optimizer using particle swarm theory", In: *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, Nagoya, 1995, pp. 39-43.

[23] G. Q. Bao, and K. Mao, "An improved PSO algorithm and its utilization in wind power generation system", *Control Engineering of China*, vol. 20, no. 2, pp. 262-271, 2013.

[24] G. Ding, and W. Cao, "Application of improved particle swarm optimization algorithm in optimal operation of reservoir", *South to North Water Transfers and Water Science & Technology*, vol. 12, no. 1, pp. 127-130, 2014.

[25] J. Zheng, "*Research oil vehicle routing problem based onimproved particle swarm optimization algorithm*", PhD thesis, *North China Electric Power University*, 2013, pp. 1-2.

[26] L. Jiang, "*Study of BP network learning method based on particle swarm optimization algorithm and simulated annealing algorithm*", PhD thesis, Anhui University, 2013, pp.1-2

[27] K. Kim, S. Lee, J. –Y. Kim, and M. Kim, "A 125GOPS 583mW network-on-chip based parallel processor with bio-inspired visual attention engine", In: *IEEE International Solid-State Circuits Conference*, San Francisco, CA, 2008.

[28] S. R. Vangal, J. Howard, G. Ruhl, and S. Dighe, "An 80-Tile Sub-100-W TeraFLOPS processor in 65-nm CMOS", *IEEE Journal Of Solid-State Circuits*, vol. 43, no. 1, pp. 29-40, 2008.

[29] C. Mineo, R. Jenkal, and S. Melamed, "Inter-Die signaling in three dimensional integrated circuits", In: *IEEE 2008 Custom Intergrated Circuits Conference*, San Jose, CA, 2008, pp. 655-658.

Received: March 16, 2015 Revised: July 23, 2015 Accepted: July 23, 2015

© Guozhi *et al.*; Licensee *Bentham Open.*

This is an open access article licensed under the terms of the (https://creativecommons.org/licenses/by/4.0/legalcode), which permits unrestricted, non-commercial use, distribution and reproduction in any medium, provided the work is properly cited.