A List Simulated Annealing Algorithm for Task Scheduling on Network-on-Chip

Song Chai
School of Communication and Information Engineering,
University of Electronic Science and Technology of China, Chengdu, China
Email: s.tschai@gmail.com

Yubai Li, Jian Wang and Chang Wu
School of Communication and Information Engineering,
University of Electronic Science and Technology of China, Chengdu, China
Email: {ybli, wangjian3630, changwu}@uestc.edu.cn

***Abstract*—In this paper, a List Simulated Anneal (LSA) algorithm is proposed for the DAG tasks scheduling on the Network-on-chip to simultaneously optimize makespan, load balance and average link load. A task list is first created for the DAG tasks, and the task-to-processor assignment is performed using Best Fit rule. Then the generated schedule is further optimized using LSA. In LSA, the task execution order is determined by the task list, and the task mapping solution is optimized using simulated annealing. By conducting series of simulation, the performance of our proposal is validated. Comparing to the list Best Fit and list random mapping algorithm, our LSA has 9% and 25.4% shorter makespan, 31.1% and 79.4% better load balance and 18% smaller average link load.**

***Index Terms*—DAG, task scheduling, Network-on-Chip, list schedule, best fit, simulated annealing**

I. Introduction

The task scheduling problem on Multiprocessor has proved to be NP-hard [1]. To solve this complex problem, many heuristics are proposed, such as list scheduling heuristics [2] and meta-heuristics [3]. List scheduling maintains a task list which is created based on the task priorities, and assigns task to processor using certain rules, such as First Fit, Best Fit and Next Fit [4]. In another way, meta-heuristics randomly search the solution space for the optimal schedule, such as Genetic Algorithm [5], Particle Swarm Optimization [6], and Simulated Annealing (SA) [7].

Simulated annealing is a generic probabilistic algorithm for global optimization which is inspired from the metallurgic process of heating and cooling down the material so that the atoms of material progress to the equilibrium state. SA simulates this process to explorer the search space.

In this paper, we combine the list schedule heuristics and SA meta-heuristics, and propose a List Simulated Anneal (LSA) algorithm for the DAG tasks scheduling on the Network-on-chip. Our proposal draws advantages from both heuristics, and performs optimization simultaneously on makespan, load balance and average link load.

The rest of this paper is organized as follow: Section II summarizes the related work; the problem is formulated in Section III; our proposal is elaborated in Section IV; Section V gives the comparative simulation results; and Section VI concludes the paper.

II. Related works

List heuristic and meta-heuristic for task scheduling for multicore system has been widely researched. For list scheduling, [8] proposes a communication-aware list scheduling algorithm for the NoC-based MPNoC. In [9], a contention-aware list scheduling algorithm is proposed for the dynamic reconfigurable NoC system. Reference [10] implements a Best Fit Decreasing heuristic for task scheduling in NoC. List scheduling is popular for its straightforward implementation and relatively low computational requirement. However in most situations, the result of list scheduling can be further optimized.

For the meta-heuristics methods, a genetic algorithm is proposed in [5] for the task scheduling in multiprocessor system. Reference [6] proposes a modified particle swarm optimization with load-balance to schedule heterogeneous tasks on to heterogeneous processors. In [7], a simulated annealing task scheduling algorithm is proposed for Voltage-Frequency islands applied NoC-based MPSoC. The meta-heuristics can effectively explore the solution space for the optima, however if combined with list schedule, the efficiency of searching as well as the probability of finding a better solution is greatly increased.

In this paper, we combine list schedule and simulated annealing, and propose a list simulated annealing scheduling algorithm to simultaneously optimize make-span, load balance and average link load.

doi:10.4304/jcp.9.1.176-182

III. Problem Formulation

A. Task Model

In this paper, tasks are modeled using Directed Acyclic Graphs (DAGs). A DAG $G=(V,E)$ is an acyclic graph where *V* is the set of nodes which represent the tasks and *E* is the set of edges in which an element e_{ij} denotes the communication from task *i* to task *j*. The edge reflects the precedent relation between two tasks. If there is an edge e_{ij} between task *i* and task *j*, then task *i* is called a *predecessor* of task *j*, and correspondingly, the task *j* is called a *successor* of task *i*. The node with no predecessor is the entry node, and the node with no successor is the exit node.

Each node v_i and edge e_{ij} is associated with a weight, denoted as C_i and T_{ij} respectively. Weight C_i is the computational load required by a Processing Element (PE) to execute task *i*; and T_{ij} is the data transmission load between task *i* and task *j*. In our work, both C_i and T_{ij} are converted to an amount of time (cycles).

The *Computation-Communication Ratio* (*CCR*) [11] of a DAG is then defined using C_i and T_{ij}. *CCR* is defined as the total computation time of a DAG divided by it total data transmission time, as given in (1).

$$CCR=\sum_{v_i\in V}C_i \Big/ \sum_{e_{ij}\in E}T_{ij} \qquad (1)$$

In DAG, a path is a sequence of nodes and edges lying in the route. The length of a path is calculated by adding up all the computation time of each node and the transmission time of each edge on the path. The top distance of node v_i is the length of the longest path from entry node to it.

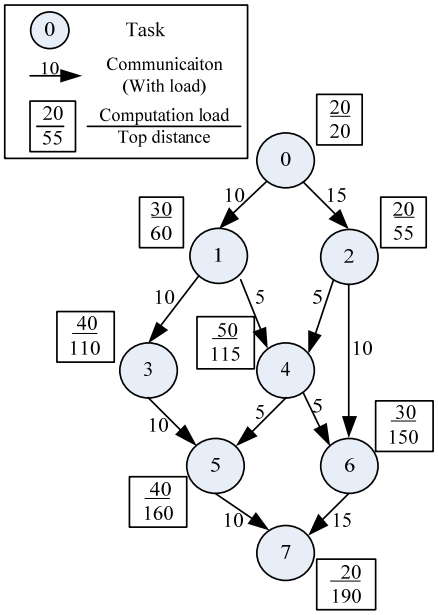


Figure 1. An example DAG tasks with 8 nodes.

Fig. 1 shows an example of a DAG with 8 tasks, and the top distance of the task is calculated and annotated in the figure.

B. Network-on-Chip Hardware

The target hardware is a 2D mesh NoC-based MPSoC, as illustrated in Fig. 2. Each PE is connected to a router, and routers are interconnected with each other through bi-direction links. Data is transferred through NoC in the form of packets.

PEs are homogenous processor cores with local data cache. If two consequential tasks are scheduled to the same PE, the successor task reads the predecessor's data directly from the data cache of the PE without routing in NoC.

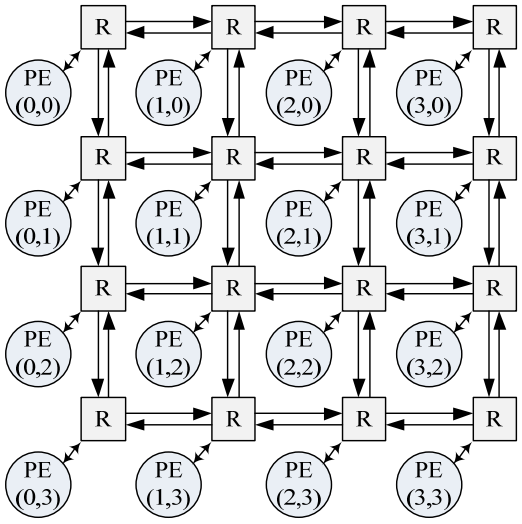


Figure 2. A 4×4 2D mesh NoC-based MPSoC

The microstructure of a NoC router is shown in Fig. 3. The router has five *Inports* and *Outports* corresponding to five directions of *East*, *West*, *North*, *South* and *Local*. The decoder in the *Inport* scans the first flit of the FIFO for any incoming packet. If decoder detects the head flit of a packet, it performs XY routing algorithm and send request signal to the arbiter of the corresponding *Outport*. If the arbiter receives multiple request signals, contention are solved using Round-Robin arbitration. The granted *Inport* then forwards the packet to the downstream router. Wormhole routing is adopted to minimize the buffer requirement as well as the packet latency [12]. The back pressure mechanism is also employed to further reduce end-to-end delay [13].

IV. Proposed Algorithm

In this section, a list simulated annealing algorithm is proposed for DAG tasks scheduling on the NoC. A task list is first created, and a schedule solution is obtained by applying Best Fit (BF) algorithm. The generated schedule is then employed as the initial solution of Simulated Annealing (SA), and optimized by the cooling down process.

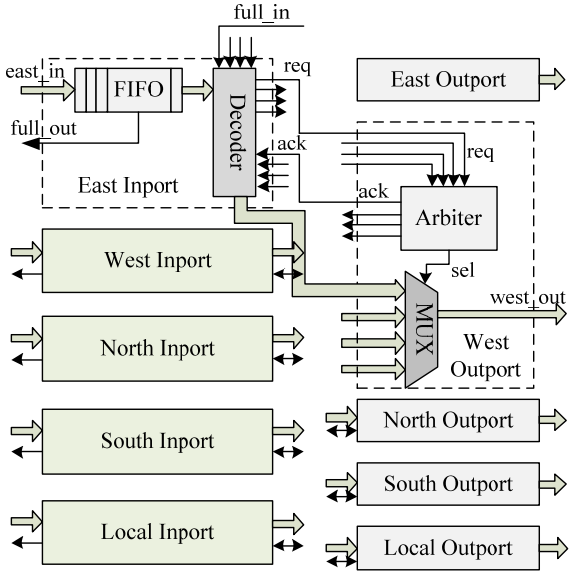


Figure 3. Microstructure of NoC node.

A. List schedule

The target DAG is processed by a list schedule algorithm to generate an initial schedule solution for the SA. The execution sequence of tasks is defined by a task list which is produced according to the top distance of tasks, and the task-to-processor assignment is performed using BF.

Task list generation: The top distance of each task in the target DAG is calculated using the method mentioned in Section III. Then the tasks are sorted based on their top distance values, and inserted to the task list. The task with the smallest top distance is placed at the head of the list. For the example DAG in Fig. 1, the task list is: task 0, task 2, task 1, task 3, task 4, task 6, task 5, and task 7.

Task list generated by top distance ensures that the precedent condition of tasks is met. For any task *i* with predecessor task *j* and successor task *k*, it is obviously that the top distance of task *i* is greater than that of task *j*, and smaller than task *k*'s top distance. In the generated task list, task *j* is scheduled prior to the task *i*, and only when task *i* finishes, task *k* is scheduled.

Best Fit scheduling: Tasks read from the list are then allocated to processor using the BF rule. For a to-be-scheduled task *i*, the BF algorithm calculates the Estimated Finish Time (EFT) of each PE, which is the time that each PE finishes current task running on it, and EFT is regarded as the earliest starting time that each PE can provide for the target task. BF chooses the PE with the best (earliest) EFT, and schedules task *i* to that PE.

Although BF scheduling is focusing solely on the makespan, its output schedule provides a good start point for the further optimization. The pseudocode of the list scheduling is shown in Fig.4.

B. List Simulated Annealing Scheduling

In this subsection, a list simulated annealing algorithm is proposed to further optimize the schedule generated by list best fit algorithm presented before. Like the algorithm

1. **FOR** *each task in DAG* **DO**
2. calculate *the top distance of each task*;
3. **END FOR**
4. sort *tasks into task list according to top distance*;
5. **FOR** *each task in task list* **DO**
6. **FOR** *each processor* **DO**
7. calculate *EFT*;
8. **END FOR**
9. schedule *current task to the processor with the smallest EFT*;
10. **END FOR**

Figure 4. Pseudo code of list scheduling.

in previous subsection, the execution order of tasks is determined by the task list which guarantees the precedent condition of tasks, and SA is used to find optimal task-to-processor allocation solution.

Solution representation: In the SA, schedule solution is represented by the symbol S. For a schedule problem with *n_tsk* tasks scheduling to *n_pe* PEs, a symbol is expressed as:

$$S = \{s_1, s_2, \ldots, s_{n\_pe}\}, \tag{2}$$

where an element $s_i = k$, $k \in \{1, \ldots, n\_pe\}$ denotes that the *i-th* task is scheduled to the *k-th* PE.

Evaluation of schedule solution: In our proposal, three metrics are monitored to evaluate the performance of a symbol (schedule), and they are: *Makespan* (*M*), *Load Balance* (*B*) [14] and *Average Link Load* (*L*) [15].

The *Makespan*, also called the schedule length, is the time span for the NoC to finish all the tasks in a DAG.

The *Load Balance* metric is defined as follow:

$$LB = \frac{ave\_load}{(\sum_{n=1}^{n\_pe}(load\_on\_pe(n) - ave\_load)^2)^{\frac{1}{2}}} \tag{3}$$

The *Load Balance* measures the inverse coefficient of variant of the total workload on each processor. The larger *B* value suggests better balanced schedule.

The last metric, *Average Link Load*, monitors the traffic load on each link, and is defined to be the average value of traffic loads on all links. A better schedule is supposed to minimize the *L* metric.

$$E = w_M(\frac{M_{ref} - M_i}{M_{ref}}) + w_B(\frac{B_i - B_{ref}}{B_{ref}}) + w_L(\frac{L_i - L_{ref}}{L_{ref}}) \tag{4}$$

The evaluation result of a symbol *i* is given in (4), which is based on the global weighted sum method in multi-objective optimization [16]. All three metric are normalized to a reference symbol which in our proposal is the schedule generated by list scheduling. The final evaluation result is the weight sum of the *improvement* of three metrics.

```
1. Initialize T, S, E, S_best and E_best;
2. WHILE TRUE DO
3.    T = Cooling(T);
4.    FOR n = 1 TO L DO
5.       S_new = NeighborMove(S);
6.       E_new = Evaluate(S_new, task_list);
7.       IF E_new > E_best THEN DO
8.          Update(S_best, E_best);
9.          Update(S, E);
10.      ELSE DO
11.         IF Accept_worse(E_best − E_new, T) THEN DO
12.            Update(S, E);
13.         ENDIF
14.      END IF
15.   END FOR
16.   IF Terminal_condition() THEN DO
17.      BREAK;
18.   END IF
19. END WHILE
```

Figure 5. Psedocode of SA

When evaluation is needed, the SA symbol along with the task list is send to the evaluation process for the assessment of the metrics, and then the process returns the evaluation score.

Cooling down process: SA starts with a high temperature level, then gradually cooling down to a lower temperature level. At each level, SA repeats certain number of neighbor searches, in which a new symbol is generated by randomly change the value of a random element from the original symbol.

Better moves of the neighbor searching are always accepted, and the original symbol is updated. A worse move is also accepted under a certain probability to prevent algorithm from local optima. The probability of accepting a worse move decreases as the temperature decreases.

The temperature cooling function is given in (5), where T_0 is the initial temperature, *i* is the ordinal number of current iteration, $L = n\_tsk \cdot (n\_pe - 1)$ [17] is the number of iteration per temperature level, and the cooling factor *q* is a constant. The cooling function forces the temperature to cooling from level to level.

$$Cooling(T) = T_0 \cdot q^{\left\lfloor \frac{i}{L} \right\rfloor} \tag{5}$$

The worse move acceptance function is defined as (6), where function *random()* returns a random number of interval (0,1), and E_0 is the performance score of the reference symbol S_0 .

Fig. 5 shows the pseudocode of the SA algorithm, and Line 4 to line 15 in the code shows the L iterations of the neighbor searching in the cooling down process.

$$Accept\_worse(E_{best} - E_{new}, T) = \begin{cases} 1, & random() < \dfrac{1}{1 + Exp(\dfrac{E_{best} - E_{new}}{E_0 \cdot T})} \\ 0, & others. \end{cases} \tag{6}$$

V. Simulation & Results

A. Simulation Setup

To evaluate the performance of our proposal, we implement the list SA (LSA) scheduling algorithm in C++, and simulate the produced schedules under a SystemC based cycle-accurate NoC simulator which is modified from the work in [18]. The list BF (LBF) as well as the list random mapping (LRM) scheduling is also implemented and simulated as reference.

The NoC is a 4×4 2D mesh structure with XY routing algorithm. Wormhole switching is adopted, and Round-Robin arbitration is enforced to solve contentions.

For the LSA, the algorithm terminates after 50 temperature level, and the cooling factor *q*=0.95. The weight coefficients in the evaluation are $w_M = 0.7$, $w_B = 0.15$ and $w_L = 0.15$.

B. Task Generation

In our simulation, both random DAG and the real-world application DAG are used for evaluation the performance of our schedule algorithm.

The 20 random DAGs are generated using TGFF 3.1 [19]. The detail information of each DAG is shown in Table I. The task number (*n_tsk*) varies from 52 to 101 which covers various of DAG sizes. Tasks' computation time is randomly generated from 40~160 cycles. *CCR* is set to 3, 2, 1.5 and 1.2 to simulate light, medium, heavy and extreme heavy communication load. The *series_w* and *series_l* are the parameters required by the new algorithm of TGFF 3.1. The *series_w* and *series_l* parameter are given as a tuple (*average*, *multiplier*), and set the width/length of series chains.

Two real world applications, solving Laplace Equation (LE) using Gauss-Seidel algorithm in [20] and Molecular Dynamics Code (MDC) in [21] are also simulated in our experiment.

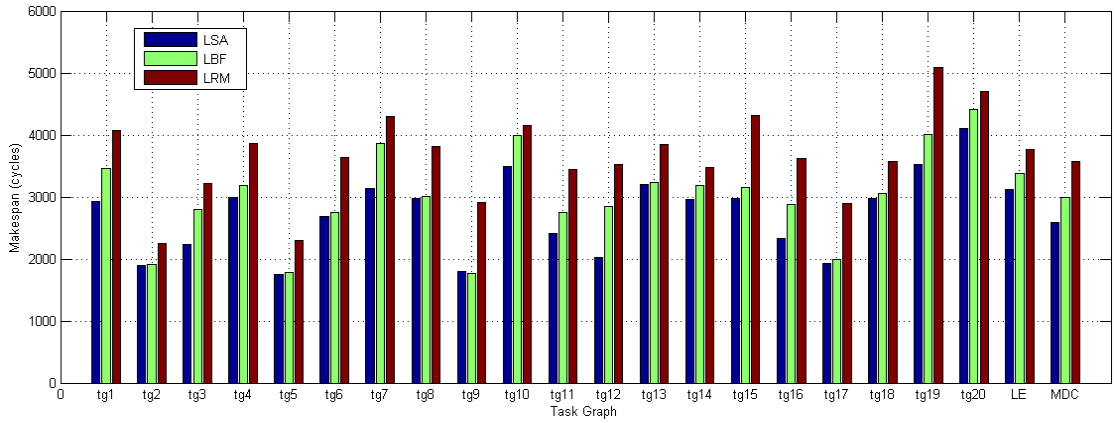
C. Simulation Results

The simulation results of List Simulated Annealing (LSA), List Best Fit (LBF) and List Random Mapping (LRM) scheduling algorithm under our NoC simulator is illustrated in Fig. 6.

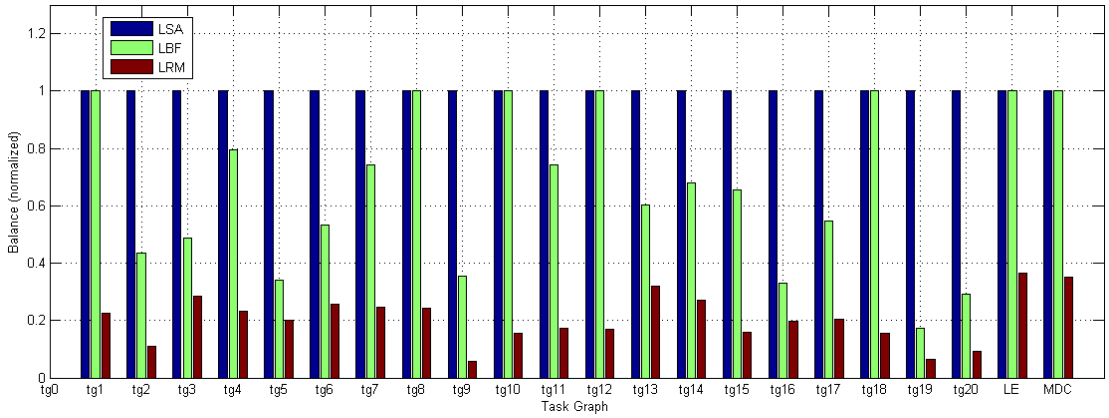
Fig. 6 (a) shows the results of makespan. The first thing to notice is that the makespan does not regularly expand as the number of tasks grows. This is because the makespan is more sensitive to the length of the critical path, which is the longest path exists in the DAG, than the number of tasks. For example, *tg1* has 63 tasks, and *tg17* has 101 tasks. However the makespan of *tg1* is 52%

TABLE I.
DETAILS OF RANDOM GENERATED TASK GRAPHS

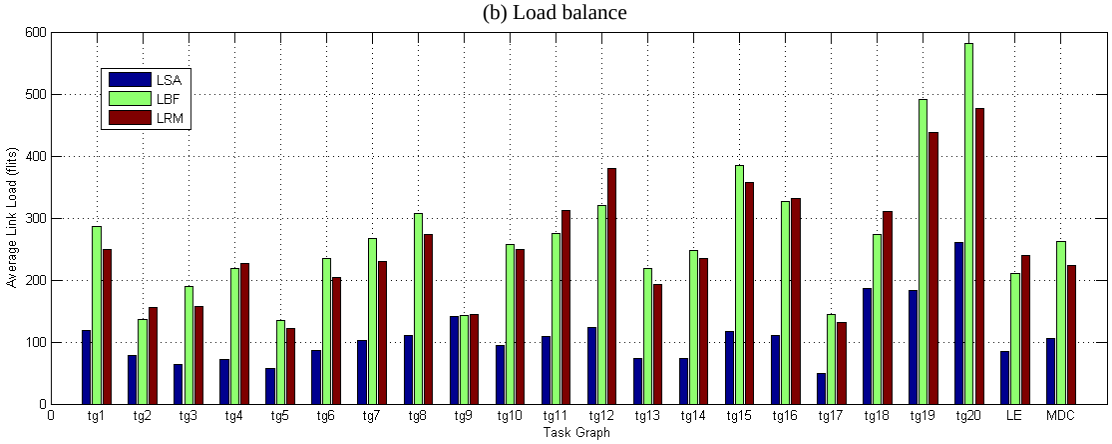
Task Graph	tg1	tg2	tg3	tg4	tg5	tg6	tg7	tg8	tg9	tg10
tsk_num	63	58	59	52	75	71	67	71	80	79
CCR	3	2	1.5	1.2	3	2	1.5	1.2	3	2
series_w	4,2	4,2	4,2	4,3	4,2	4,2	4,2	4,2	5,2	5,2
series_l	3,2	3,2	3,2	3,2	3,2	3,2	3,2	3,2	4,2	4,2
Task Graph	tg11	tg12	tg13	tg14	tg15	tg16	tg17	tg18	tg19	tg20
tsk_num	77	77	91	91	93	92	101	97	96	97
CCR	1.5	1.2	3	2	1.5	1.2	3	2	1.5	1.2
series_w	5,2	5,2	5,2	5,2	5,2	5,2	6,2	6,2	6,2	6,2
series_l	4,2	4,2	4,2	4,2	4,2	4,2	5,2	5,2	5,2	5,2



(a) Makespan



(b) Load balance



(c) Average link load

Figure 6. Simulation results of LSA, LBF and LRM.

larger than the makespan of *tg17*, for the critical path of *tg1* (1557 cycles) is larger than that of *tg17* (1226 cycles).

From Fig. 6 (a) we observe that the makespan of LSA is always smaller than the makespan of LBF and LRM. The average makespan of LSA is 91% of LBF and 74.6% of LRM. We also observe that in some situations, the makespan results of LSA is very close to that of LBF, that is because the schedule produced by LBF is already near-optima.

Fig. 6 (b) shows the load balance results, and all the results are normalized to the LSA equivalent. Notice that in some situations, the load balance of LSA is identical to that of LBF. The explanation to this phenomenon is that two or more processors swap all their tasks during the cooling down process of LSA. Besides these situations, LSA still outperforms LBF by 33.1% and LRM by 79.4%.

The average link load results are illustrated in Fig. 6 (c). Obviously, LSA remarkably reduces the average link load on NoC. The average link load of LSA is 57.9% and 56.3% smaller than that of LBF and LRM.

Moreover, we measure the end-to-end routing delay of each packet, and the overall average end-to-end delay of three schedule algorithm is shown in Fig. 7. Although it is not a optimization goal in our proposal, as presented in the figure, the average end-to-end delay of LBF and LRM are of the same level, and the routing delay of LSA is 18% shorter than that of LBF and LRM. This is because both makespan and average link load optimization favors the schedule with shorter routing delay.

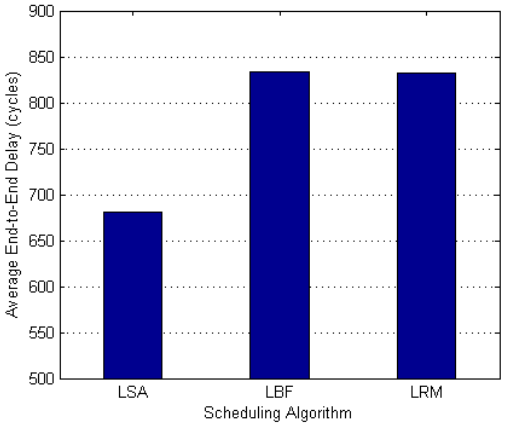


Figure 7. Average End-to-End Delay

VI. Conclusion

A list simulated annealing scheduling algorithm is proposed in this paper for the DAG tasks scheduling on the NoC. The proposal combines list schedule and simulated annealing to optimize makespan, load balance and average link load. Through series of simulations, the performance of our proposal is validated.

Acknowledgment

This work was supported by the National Natural Science Foundation of China [61201005], the Fundamental Research Funds for the Central Universities [No.ZYGX2011J006] and the National Major Projects [2011ZX03003-003-04].

References

[1] M. Bambagini, G. Buttazzo, and S. Hendseth, "Exploiting Uni-Processor Schedulabilty Analysis for Partitioned Task Allocation on Multi-Processors with Precedence Constraints," *RTSOPS 2012*, p. 17, 2012.

[2] H. Arabnejad and J. G. Barbosa, "Performance evaluation of list based scheduling on heterogeneous systems," in *Euro-Par 2011: Parallel Processing Workshops*, 2012, pp. 440-449.

[3] S. Mandloi and H. Gupta, "A Review of Resource Allocation and Task scheduling for Computational Grids based on Meta-heuristic Function," *IJRCCT*, vol. 2, pp. 099-102, 2013.

[4] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, 2010, pp. 1-8.

[5] S. Gupta, G. Agarwal, and V. Kumar, "Task scheduling in multiprocessor system using genetic algorithm," in *Machine Learning and Computing (ICMLC), 2010 Second International Conference on*, 2010, pp. 267-271.

[6] S. Sivanandam and P. Visalakshi, "Dynamic task scheduling with load balancing using parallel orthogonal particle swarm optimisation," *International Journal of Bio-Inspired Computation*, vol. 1, pp. 276-286, 2009.

[7] S. Ninomiya, K. Sakanushi, Y. Takeuchi, and M. Imai, "Task Allocation and Scheduling for Voltage-Frequency Islands Applied NoC-based MPSoC Considering Network Congestion," in *Embedded Multicore Socs (MCSoC), 2012 IEEE 6th International Symposium on*, 2012, pp. 107-112.

[8] H. Yu, Y. Ha, and B. Veeravalli, "Communication-aware application mapping and scheduling for NoC-based MPSoCs," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 3232-3235.

[9] M. Tagel, P. Ellervee, T. Hollstein, and G. Jervan, "Contention aware scheduling for NoC-based real-time systems," in *NORCHIP*, 2011, 2011, pp. 1-4.

[10] É. Cota, A. de Morais Amory, and M. S. Lubaszewski, "NoC Reuse for SoC Modular Testing," in *Reliability, Availability and Serviceability of Networks-on-Chip*, ed: Springer, 2012, pp. 59-83.

[11] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms," *Journal of Systems Architecture*, vol. 56, pp. 242-255, 2010.

[12] F. Samman, T. Hollstein, and M. Glesner, "New theory for deadlock-free multicast routing in wormhole-switched virtual-channelless networks-on-chip," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, pp. 544-557, 2011.

[13] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Systems Architecture*, vol. 50, pp. 105-128, 2004.

[14] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *Web Information Systems and Mining*, ed: Springer, 2010, pp. 271-277.

[15] N. Nikitin, S. Chatterjee, J. Cortadella, M. Kishinevsky, and U. Ogras, "Physical-aware link allocation and route assignment for chip multiprocessing," in *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, 2010, pp. 125-134.

[16] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, pp. 369-395, 2004.
[17] H. Orsila, E. Salminen, and T. D. Hämäläinen, "Best practices for simulated annealing in multiprocessor task distribution problems," *Simulated Annealing*, pp. 321-342, 2008.
[18] S. Chai, C. Wu, Y. Li, and Z. Yang, "A NoC simulation and verification platform based on systemC," in *Computer Science and Software Engineering*, 2008 International Conference on, 2008, pp. 423-426.
[19] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*, 1998, pp. 97-101.
[20] M.-Y. Wu and D. D. Gajski, "Hypertool: A programming aid for message-passing systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 1, pp. 330-343, 1990.
[21] S. Kim and J. Browne, "A general approach to mapping of parallel computation upon multiprocessor architectures," in *International Conference on Parallel Processing*, 1988, p. 8.