Low Latency Clustering & Mapping Algorithm with Task Duplication Technique on Cluster-Based NoC

Fangfa Fu, Yuxin Bai, Xin'an Hu, Jinxiang Wang, Mingyan Yu

Abstract—A Clustering & Mapping (CM) Algorithm, which can automatically divide task graphs into clusters and map tasks onto homogeneous cluster-based Network-on-Chip architectures, is presented in this paper. It performs task-duplication-based lineal clustering to group tasks into a series of clusters, and utilizes a heuristic task mapping process to allocate ready clusters onto the platform efficiently. Then a pipeline-based static task scheduling stage is used to enhance the throughput for streaming applications. CM can fully exploit the parallel characteristics within task graphs, minimize the inner-cluster communication delay upon the cluster-based NoC platform and utilize the individuality among generated clusters to further reduce the inter-cluster communication cost. The results generated by this algorithm are verified by a cycle-accurate simulator written in SystemC. Experiments show that significant communication time savings can be achieved by using the Clustering & Mapping algorithm when employed to the task graphs of 36, and 72 tasks. Compared to the results generated by PHTM and EACS, 48% and 56% time savings on average can be observed without obviously harming energy performance.

Index Terms—**Network-on-Chip, Mapping, Scheduling, Pipeline, Clustering.**

I. Introduction

The development of semiconductor technologies indicates that future SoC applications will require huge computation and communication capabilities and will inevitably consist of multiple processing cores integrated by on-chip communication architectures. This great number of processing cores has brought great challenges in many aspects, such as the scalability, communication performance and power efficiency. In this context, Network-on-Chip (NoC), which is proposed to replace the traditional bus-based on-chip communication architecture, provides a structured way of realizing inter-core communications on silicon.

With the increase of the network scale, however, the hardware cost of too many on-chip routers upon nodes, and NIs (Network Interface) whose area and power cost may be comparable with the IP cores, will be expensive. The bandwidth of router ports might not be sufficiently enjoyed with every router port of the same bandwidth either. To avoid these problems, an improved NoC architecture with cluster-based granularity [3, 4, 5, 12] is presented. In this architecture, a cluster is a set of IP cores that couple tightly with each other and is connected to a router node on the network. The inter-cluster communication is handled by NoC components such as the NIs and routers while for inner-cluster communication, on-chip bus is still utilized since it is more efficient than expensive routers and it can gather communication demands among a cluster of IP cores to fully exploit the bandwidth of routers ports. To achieve this NoC implementation, a target application composed of a set of existing tasks must be mapped onto a customized NoC platform, as is shown in Fig. 1-e. It is a significant step as well as an open problem for the NoC design.

Based on this promising cluster-based NoC architecture, in this paper, a Clustering & Mapping Algorithm is proposed to enhance the communication efficiency and homogenous IP cores are chosen to reduce the solution space. The Clustering & Mapping Algorithm (CM) first divides given task graphs into clusters, which reduces the size of original task graphs. A Tasks Duplication Technique (TDT) is used here to further reduce the communication overhead by redundantly allocating tasks onto multiple clusters. Ready clusters are then mapped onto cluster-based router nodes according to some kind of priority. As a result, utilizing the natural individuality of those task clusters, the communication delay among router nodes would be reduced on the network. Then we adopt a kind of heuristic pipelined-based approach for streaming applications to allocate and schedule tasks within cluster-based router nodes.

The remainder of this paper is organized as follow. The related work in this area is introduced in Sec.2, and then the definitions and problem formalization are presented in Sec.3. In Sec.4, the clustering algorithm description is proposed. Finally, the experimental results and conclusions are presented.

II. Related Work

Except for some special cases [6], the search for an optimal solution to the problem of multi-processor tasks mapping or scheduling has been proven to be NP-hard. Numerous approaches, which can be mainly classified into two categories [1]: non-deterministic approaches and deterministic approaches, have been developed to solve the problem.

Non-deterministic algorithms incorporate a combinatory process in the search for solutions. They typically require sufficient sampling of candidate solutions in the search space and have shown robust performance on a variety of

Manuscript received October 31, 2009.

F. Fu, Y. Bai, X. Hu, J. Wang, and M. Yu are with the Microelectronics Center, Harbin Institute of Technology, Harbin, Heilongjiang Province 150001, P.R China (fff1984292@163.com, abayax@gmail.com, xinanhu@hit.edu.cn, jxwang@hit.edu.cn, myyu@hit.edu.cn).

scheduling problems. Genetic algorithms [8, 13, 19, 18, 21], simulated annealing [4, 9, 15], tabu search [16], and artificial immune systems [14, 7, 5, 20] have been successfully applied to various scheduling problems. Non-deterministic algorithms, however, are less efficient and have much higher computational cost than deterministic algorithms.

Deterministic approaches attempt to utilize the heuristics from specific problems and try to guide the search for a solution. And many of them belong to list scheduling algorithms, which can be divided into two steps: in the first step, a priority value is given to each task in some criteria; in the second step, tasks are assigned to processors in some order of their priorities. ISH [11], DSH [11], MCP [22], and CPFD [2] are typical list scheduling approaches to homogeneous computing systems, while HEFT [17] and CPOP [17] are list scheduling algorithms designed for heterogeneous computing systems. However, the performance of these algorithms is heavily dependent on the effectiveness of the heuristics.

Another group of deterministic algorithms is clustering algorithms [10, 23]. These algorithms assume that there are an unlimited number of processors available for task execution. Clustering algorithms will use as many processors as possible in order to reduce the makespan (scheduling length or overall finishing time of a parallel application) of the schedule and could exploit the nature of individuality within clusters. Only if the number of processors used by a schedule is greater than the number actually available in a given problem, a mapping or merging process is required to merge the tasks in the proposed schedule onto the actual number of available processors. Therefore, by adding a tasks merging stage, clustering algorithms which would bring about great help to reduce communication time especially on our cluster-based NoC architecture are explored in this work.

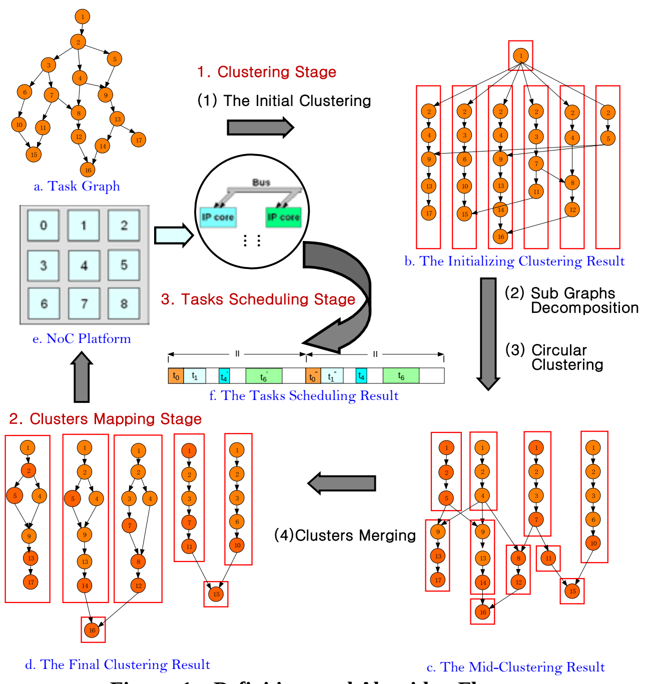
1. Clustering Stage
(1) The Initial Clustering
a. Task Graph
Bus
IP core
IP core
b. The Initializing Clustering Result
e. NoC Platform
3. Tasks Scheduling Stage
(2) Sub Graphs Decomposition
(3) Circular Clustering
f. The Tasks Scheduling Result
2. Clusters Mapping Stage
(4)Clusters Merging
d. The Final Clustering Result
c. The Mid-Clustering Result

Figure. 1 Definitions and Algorithm Flow

III. Definitions and the Problem Formulation

3.1 Definitions

(1) Task Graph

As is shown in Fig. 1-a, the given task graph is shown as a node-labeled and edge-labeled DAG (Directed Acyclic Graph), which is described as a four elements set $G = (V, E, T, C)$. The vertex set in the graph can be represented as $V = \{n_i \mid n_i$ is an ordered task, $i = 1, 2, 3 \dots v\}$. The edge set is described as $E = \{e_{ij} \mid e_{ij}$ is the edge from n_i to $n_j\}$. The task computation time set can be represented as $\Gamma = \{\tau_i \mid \tau_i$ is the computation time of $n_i, i = 1, 2, 3 \dots v\}$ and the communication time set is $Comm = \{c_{ij} \mid c_{ij}$ is the communication time from task n_i to n_j, n_i is the fork task of $n_j\}$. Task i is defined as $t_i \epsilon T$, and all tasks in a task graph compose a Task set T. If tasks are allocated onto the same node in one cluster, their communication cost will be zero.

(2) Predecessors $pred(i)$ and posterities $succ(i)$

All the predecessors (join tasks) of task i, are tasks who directly communicate with and are before task i. They compose the set $pred(i)$. All the posterities (fork tasks) of task i, are tasks who directly communicate with and are after task i. They compose the set $succ(i)$.

(3) Network Architecture Graph (NAG)

The NoC platform is described by a Network Architecture Graph (NAG) $N = N(R, L)$, which is a directed acyclic graph, where each node represents a router r_i; each $r_i \in R$ is able to connect with one PE through a NI. It is shown in Fig. 1-e.

(4) Clustering Stage

Cluster tasks $t_i \in T$ in given task graphs $G = (V, E, T, C)$into clusters, and guarantee the clusters satisfy some constraint requirements. We use function $\xi: t_i \rightarrow C(i)$ to represent "Clustering Stage".

(5) Clusters Mapping Stage

Allocate each cluster $c_i \in C$ onto the router node in the NAG. We use the function $\omega: C \rightarrow R$ to represent "Clusters Mapping Stage".

(6) Tasks Scheduling Stage

For the set of tasks allocated onto IP cores connected to the router node r_i, determine the execution sequence Seq of these tasks to meet with the real-time constraints. We use the function $\nu: U \rightarrow Seq$ to represent "Tasks Scheduling Stage".

(7) The earliest start time $est(i)$ and the earliest complete time $ect(i)$

$$est(i) = 0, \text{ if } pred(i) = \phi;$$

$$est(i) = min_{j\epsilon pred\ (i)}\ max_{k\epsilon pred\ (i), k\neq j}\left(ect(j), ect(k) + ck,j\right), \text{ if } predi\neq\phi;$$

It means tasks could only start after all of their fork tasks have been completed and their inter-communication has been finished.

$$ect(i) = est(i) + \tau_i;$$

(8) The value of tasks levels $level(i)$

It is the accumulation of execution time of tasks in a path and describes the executing order of all the tasks in some sense. It will be used as a parameter for the following process of the algorithm later.

$$level(i) = \tau_i, \text{ if } succ(i) = \phi;$$

$$level(i) = \tau_i + \max_{k\epsilon succ(i)}\left(level(k)\right), \text{if } succ(i) \neq \phi;$$

(9) The best predecessor $fpred(i)$

$$fpred(i) = j \mid \left(ect(j) + c_{i,j}\right) \geq \left(ect(k) + c_{k,i}\right),$$
$$\forall j \in pred(i); k \in pred(i), k \neq j;$$

3.2 Problem Formulation

Given:

The task graph: $G = (V, E, T, C)$;

The network architecture: $N = N(R, L)$, with each router node connecting to a set of IP cores;

For every $t_i \in T$ in G, choose a group of t_i;

To find functions:

$$\xi : t_i \to C(i);$$
$$\omega : C \to R;$$
$$\nu : U \to Seq.$$

IV. Clustering & Mapping Algorithm Description

The algorithm is described as three stages: clustering stage; clusters mapping stage; tasks scheduling stage, which is shown in Fig. 1.

4.1 Clustering Stage

In this stage, given task graphs are divided into clusters. The required property of each cluster is that it must only have one source node and one sink node. Thus the new clusters generated after this stage can be viewed as new "tasks" in a smaller size task graph. Since IP cores are interconnected with each other by bus, the communication delay inside clusters on router nodes is ignored here. Two clustering stage termination conditions are shown as below:

A. The overall communication volumes do not decrease anymore;

B. The number of clusters is less than that of the router nodes.

The process of this stage could be shown as four steps in Fig. 1: (1) The Initial Clustering; (2) The Decomposition of Sub Graphs; (3) Circular Clustering; and (4) Clusters Merging.

4.1.1 The Initial Clustering

There are two categories of clustering: lineal clustering and non-lineal clustering. Since the lineal clustering approach maps tasks in the same key path and exploits the parallel characteristics of the DAG well, lineal clustering is utilized to complete this stage. The first step of lineal clustering is to find tasks on all related path and to map them onto different clusters. A Task Duplication Technique is utilized to reduce scheduling length by increasing the individuality of clusters. According to level values of tasks, we then set a queue of tasks arranged from low to high level values. We call it a Level Queue (LQ). This is a down-to-top searching method from the sink task to source task. Fig. 2 shows the pseudo code of the initialization.

```
for every task from the first to last in LQ
   if(task i is not marked)
         put task i into a new cluster;
         cluste_number ++;
        mark task i;

        if (fred(i) is not marked)
           put fpred(i) into the same cluster;
           mark fpred(i);
        else
           copy fred(i) into the same cluster;

   else
         continue to look for unmarked tasks;
```

Figure. 2 Pseudo Code of Clustering Initialization

4.1.2 Sub Graphs Decomposition

All clusters generated in the previous step are called sub graphs. According to the constraints mentioned above, these clusters should be further regulated until each of the clusters only has one source task and one sink task in order to guarantee their individuality. Therefore, in this step, we would decompose the initial clustering results into new kind of clusters which have the property of a single task, such as owning predecessors and posterities.

Input: the initial clustering results C_1, C_2,…,C_n;

Output: new clusters satisfying the requirement that each has only one source task and one sink task.

Decomposition rules are shown as the pseudo code below in Fig. 3.

```
for every cluster generated from the initialization
  if (tasks number of Ci ==1)
     put this task into a new cluster;
  else
    for every task in Ci top-to-down according to Level values
       if (fork tasks number of ti >1)
          if (all fork tasks of ti are copied)
             put ti into the same cluster with its fpred(i);
          else
             put ti into a new cluster;
       else
          if (join tasks number of ti >1)
             if (ti is copied)
                put ti into the same cluster with its fpred(i);
             else
                put ti into the same cluster with its fork task;
                put the join task of ti into a new cluster;
          else
             if (ti is not the last task in Ci)
                put ti into the same cluster with its fork task;
             else
                if (fork task of ti is copied and its join tasks number >=1)
                   put ti into the same cluster with its fork task;
                else
                   put ti into a new cluster;
```

Figure. 3 Pseudo Code of Sub Graph Decomposition

4.1.3 Circular Clustering

The result of sub graphs decomposition achieves the goal of regulating initial clusters to new ones which satisfy some properties of tasks. However, two overall constraint conditions would not be met after clustering just once under some circumstances, so we have to continue to regulate clusters until the number of clusters is less than that of routers or the whole communication volumes do not decrease any more. The circulation is sure to terminate since the communication volumes wouldn't be below zero, no matter whether constraint B is met or not.

The key of this step is to execute initial clustering and sub graphs decomposition circularly until all constraints are met before the algorithm is terminated. The result of this step can be shown as Fig.1-(3).

4.1.4 Clusters Merging

Only when the algorithm is terminated with the number of clusters exceeding that of router nodes, do we need to execute this step of merging clusters. Since the exceeded number of clusters will largely increase the computational complexity of this algorithm, we implement this step to guarantee the simplicity of next mapping stage.

A. Definitions

The generated clusters set is C={C_i}. Select two clusters

C_i、C_j arbitrarily, and denote that the number of their same tasks in two clusters is χ, the number of tasks in cluster C_i is α, and the number of tasks in cluster C_j is β. The relevance coefficient between C_i and C_j, is defined as $\eta_{ij} = \chi/\beta$, while the relevance coefficient between C_j and C_i is defined as $\eta_{ji} = \chi/\alpha$.

We could also define a correlation cluster set $CL = \{C_j | C_j \neq C_i\}$ for cluster Ci, which holds all other clusters in a queue, which is called *para_queue*, down ordered by the relevance coefficients with C_i.

B. Merging Algorithm

In this step, not only we should reduce the total number of clusters, but we also need to try our best to reduce the number of copied tasks to constrain total energy consumption, and need to increase the individuality among clusters by cutting down communications among clusters.

(1) Merging none-individual clusters

Consider a cluster in the clusters set *C*. If the fork clusters of the current cluster do not have any previous clusters, merge all the fork clusters into the current cluster.

(2) Merging individual clusters

For a cluster C_i, first arrange clusters from its correlation cluster set *CL* in a new queue called *cluster_size_queue*, with the up order of α, which is the number of tasks in clusters. Then take the first cluster from the *cluster_size_queue* and search for clusters in *para_queue*. If the cluster from the *cluster_size_queue* is found in the *para_queue* and the corresponding relevance coefficient is 1, merge this cluster into C_i. Then check the termination conditions. If they are not met, the second cluster in the *cluster_size_queue* will be fetched and the process will be executed similarly as before. If all the clusters in the *cluster_size_queue* have been checked and the termination conditions are still not satisfied, we have to choose clusters with the highest coefficient values to merge. The sample result is shown in Fig. 1-d.

4.2 Clusters Mapping Stage

In this stage, ready clusters that meet the required constraints are mapped onto router nodes. Here we use a heuristic method to allocate clusters in the down sequence of their Level values and put them onto the nodes where they will bring about communication cost as low as possible. Also, the availability of the link path should be considered. The clusters mapping and link scheduling method is described as below:

(1) Source cluster mapping: In order to make it convenient for other clusters to communicate with the source cluster, put the source cluster in the center of the architecture so that the communication hops can be reduced between source cluster and related clusters.

(2) Other clusters mapping and their communication links scheduling:

Let *CommCost* represent the communication cost between clusters on different nodes.

$$CommCost = \sum_{join\_task} hop \times com\_volume$$

Firstly, we allocate other clusters by the down order of their level values so that early clusters would be mapped early.

Secondly, we check the parameter *CommCost* of all the available nodes for the current cluster. Then check the availability of the link paths between the current cluster and a previous cluster, by the order of the node with the least *CommCost* value to the one with highest *CommCost* value. If the link paths are schedulable for the first available node, map the current cluster onto this node. Else, check whether it is possible to map it on the second available node. After each cluster is mapped on a router node, we update the information of links. In this way, we repeat this work above until the clusters are all mapped onto the network. This would be a simple approach because the number of clusters is not greater than that of the router nodes.

4.3 Task Scheduling Stage

Since we have already mapped all the clusters onto the router nodes, this stage is to further map and schedule tasks inside clusters on IP cores within a router node.

For each task to be allocated, we first check each core on the router node to judge whether the task can be schedulable. In order to apply to the streaming application, pipelined scheduling method is introduced for tasks scheduling inside clusters. Tasks are iterated by a sample period, which is also called pipeline period. And because of the periodicity, once it is schedulable at this time period, then it will be schedulable one circle later in the next period. By the down sequence of level values, when a task is allocated onto the proper core, the scheduling table on each IP core is updated. In doing so, we not only insert the execution time of this task into it, but also duplicate it with the pipeline period for several circles because of the pipelined implementation. Fig. 5 shows the pipelined scheduling. Non-primed labels t_0, t_1, t_4 and t_6 indicate tasks from the current iteration, primed labels t_4 and t_6 indicate tasks from the previous iteration, while double-primed labels t_0 and t_1 indicate tasks from the next iteration.

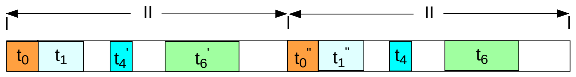

Figure. 4 Pipelined Scheduling

On the other hand, the fork tasks whose finishing time is exactly known, however, have been allocated on the best core, thus we just involve finishing time of task in the first iteration. After the maximal finishing time among fork tasks, we search for the idle time that is suitable for the next task to be allocated, and then its latest start time can be found.

V. Experimental Results

Two kinds of streaming applications with significant amount of communication among tasks are applied to our experiments as Task Graph input, which is shown in Table 1. The task graphs of two applications include 36 and 72 tasks respectively. The chosen homogenous IP core is AMD ElanSC520-133 MHz – square whose idle power is 0.16watt, and working power is 1.6watt with the supply voltage of 1.5V and threshold voltage of 0.6V. The network ran at the frequency of 100MHz with a bandwidth of 6400Mbit/s. We implement a cycle accurate Simulator written in SystemC which describes the models of the routers architecture, network interface, and IP cores to verify this algorithm. Then we choose other mapping & scheduling algorithms, PHTM [3] and a random mapping algorithm EACS [24] for comparison.

Table. 1 Applications Characteristic

Application ID	Nodes	Edges
Application1	36	49
Application2	72	87

In order to evaluate the proposed algorithm, we regard Ming Li's work PHTM and Qingli Zhang's work EACS as a compare in the aspects of scheduling length and energy consumption. Although Qingli Zhang's work is applied to heterogeneous NoC architecture, we can change it into homogenous one by fixing the category of PE without influencing the core of his algorithm. Each of them is mapped onto a 5×5 homogenous NoC. Fig. 5 and 6 show the normalized scheduling length result of three algorithms. Compared with PHTM and EACS, our algorithm CM has saved scheduling length by 72% and 76% respectively, when implemented on the graph of 36 tasks, as is shown in Fig. 5. As the increase of complexity of task graph, scheduling length has been saved by 24% and 36% respectively, when implemented on the graph of 72 tasks. It is shown in Fig. 6. The average saved scheduling length is 48% and 56%, which reveals an obvious reduction of applications running time and a remarkable performance improvement.

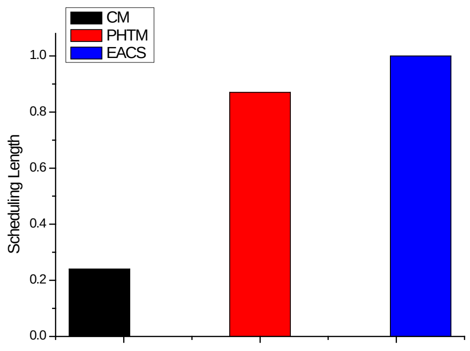

Figure. 5 Scheduling Length Comparison on the graph of 36 tasks

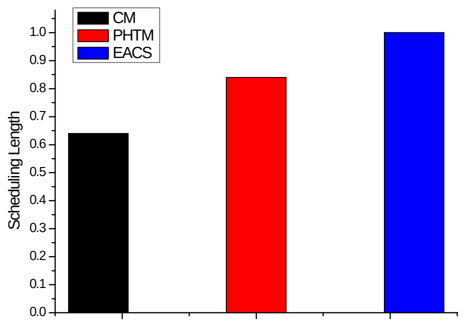

Figure. 6 Scheduling Length Comparison on the graph of 72 tasks

Fig. 7 shows the normalized energy consumption result of three algorithms. Our algorithm has an energy increment by 15% and 3% respectively, compared with PHTM and EACS on the graph of 36 tasks. When algorithms are implemented on the graph of 72 tasks, CM saves energy by 3% compared with EACS but has a 13% energy increase compared with PHTM. That's because duplicated tasks in clusters of CM Algorithm increase extra tasks execution energy consumption, though communication consumption energy has been reduced. Thus the overall energy consumption would somehow task-graph-dependent. On average, the energy consumption of CM Algorithm is almost equivalent to that of EACS and increases by 14% when compared with PHTM. Therefore, for our CM Algorithm, energy performance is just slightly sacrificed when a huge improvement in scheduling length is brought about.

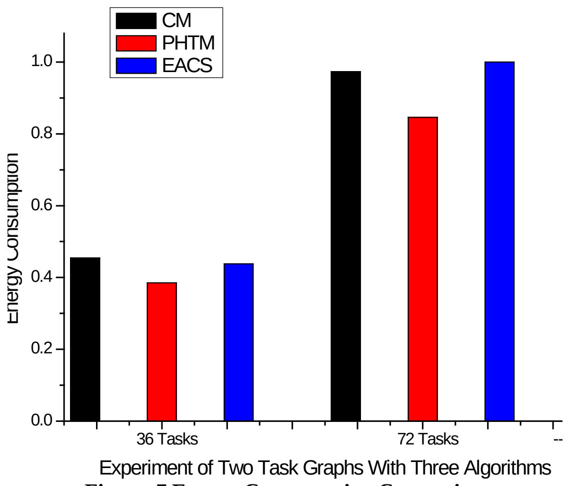

Figure. 7 Energy Comsumption Comparisons

VI. Conclusion

In this paper, a Clustering & Mapping Algorithm is presented, which performs TDT-based clustering, clusters mapping, and tasks scheduling simultaneously on homogenous cluster-based NoC platform. This clustering method can fully exploit the parallel characteristics within task graphs and can utilize the individuality among clusters to further reduce their inter-communication cost. It is verified on a cycle accurate Simulator written in SystemC and compared with PHTM [3] and a random mapping algorithm EACS [24] running on graphs of 36 and 72 tasks. And experimental results show that CM algorithm has achieved a significant improvement in shortening scheduling length on homogenous cluster-based NoC platform, with almost equivalent energy performance with EACS and slightly more energy sacrifice compared to PHTM, thus our further work would be focused on reducing energy consumption and on implementing our algorithms onto heterogonous architectures.

Acknowledgment

Thank Ming Li, Liguo Wang, and Junjie Song for their help to this work as well as Siyue Sun etc. for the advice on this paper.

References

[1] Han Yu. Optimizing Task Schedules Using An Artificial Immune System Approach. 2008 ACM 978-1-60558-130-9/08/07.
[2] I. Ahmad and Y. Kwok. On exploiting task duplication in parallel program scheduling. IEEE Transactions on Parallel and Distributed Systems, 9(9):872–892, 1998.
[3] Ming-Yan Yu, Ming Li, Jun-Jie Song, Fang-Fa Fu, Yu-Xin Bai. Pipeling-based High Throughput Low Energy Mapping on Network-on Chip. In Proc. of the Euromicro Symposium on Digital Systems Design, Patras, Greece, 2009.
[4] S. W. Bollinger and S. F. Midkiff. Processor and link assignment in multicomputers using simulated annealing. In Proceedings of the International Conference on Parallel Processing, pages 1–7, 1988.
[5] A. Costa, P. Vargas, F. V. Zuben, and P. Franca. Makespan minimisation on parallel processors: An immune based approach. In Proceedings of the Congress on Evolutionary Computation, pages 920–926, 2002.

[6] M. R. Garey and D. S. Johnson. Computers and intractability, a guide to the theory of NP-Completeness. W. H. Freeman, New York, 1979.

[7] E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In Proceedings of Genetic and Evolutionary Computation Conference, pages 1559–1565, 1999.

[8] E. S. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems, 5(2):113–120, 1994.

[9] K. Hwang and J. Xu. Mapping partitioned program modules onto multicomputer nodes using simulated annealing. In Proceedings of the International Conference on Parallel Processing, pages 292–293, 1990.

[10] S. J. Kim and J. C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. In International Conference on Parallel Processing, volume 2, pages 1–8, 1988.

[11] B. Kruatrachue and T. G. Lewis. Duplication Scheduling Heuristic, a new precedence task scheduler for parallel systems. Technical Report 87-60-3, Oregon State University, 1987.

[12] Jingcao Hu. "Design Methodologies For Application Specific Networks-on-Chip". PhD thesis. Carnegie Mellon University, May, 2005

[13] Y. Kwok and I. Ahmad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. Journal of Parallel and Distributed Computing, 47(1):58–77, 1997.

[14] M. Mori, M. Tsukiyama, and T. Fukuda. Adapative scheduling system inspired by the immune system. In Proceedings of the IEEE Conference on Systems, Man and Cybernetics, pages 3833–3837, 1998.

[15] A. K. Nanda, D. DeGroot, and D. Stenger. Scheduling directed task graphs on multiprocessors using simulated annealing algorithms. In Proceedings of the 12th International Conference on Distributed Computing Systems, 1992.

[16] S. C. S. Porto and C. C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. International Journal of High-Speed Computing, 7(2), 1995.

[17] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel & Distributed Systems, 13(3):260–274, 2002.

[18] T. Tsuchiya, T. Osada, and T. Kikuno. Genetic-based multiprocessor scheduling using task duplication. Microprocessors and Microsystems, 22:197–207, 1998.

[19] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogenous computing environments using a genetic-algorithm-based approach. Journal of Parallel and Distributed Computing, 47(1):8–22, 1997.

[20] G. Wojtyla, K. Rzadca, and F. Seredynski. Artificial immune systems applied to multiprocessor scheduling. In Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics, pages 904–911, 2005.

[21] A. S. Wu, H. Yu, S. Jin, G. Schiavone, and K.-C. Lin. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel & Distributed Systems, 15(9):824–834, 2004.

[22] M. Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. IEEE Transactions on Parallel & Distributed Systems, 1(3):330–343, 1990.

[23] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel & Distributed Systems, 5(9):951–967, 1994. 158.

[24] Qing-Li Zhang, et al. "Energy-aware HW/SW co-synthesis algorithm for Heterogeneous NoC ", ASP-DAC, 2009.