Cluster-based Simulated Annealing for Mapping Cores onto 2D Mesh Networks on Chip

Zhonghai Lu, Lei Xia and Axel Jantsch
Dept. of Electronic, Computer and Software Systems
Royal Institute of Technology (KTH), Stockholm, Sweden
{zhonghai,leix,axel}@kth.se

Abstract—**In Network-on-Chip (NoC) application design, core-to-node mapping is an important but intractable optimization problem. In the paper, we use simulated annealing to tackle the mapping problem in 2D mesh NoCs. In particular, we combine a clustering technique with the simulated annealing to speed up the convergence to near-optimal solutions. The clustering exploits the connectivity and distance relation in the network architecture as well as the locality and bandwidth requirements in the core communication graph. The annealing is cluster-aware and may be dynamically constrained within clusters. Our experiments suggest that simulated annealing can be effectively used to solve the mapping problem with a scalable size, and the combined strategy improves over the simulated annealing in execution time by up to 30% without compromising the quality of solutions.**

I. Introduction

In a top-down design flow, Network-on-Chip (NoC) application design is to map a set of IP cores onto a set of network nodes. The IP cores communicate with each other and impose timing constraints on delay and bandwidth. It is the core-to-node mapping, which determines which nodes host which cores, that fundamentally dominate whether the network can fulfill the constraints and the cost and power consumption for the fulfillment. The mapping is therefore a crucial design decision to make at an early design phase.

The core-to-node mapping problem is a NP-Hard (Nondeterministic Polynomial-time Hard) combinatorial optimization problem, thus inherently intractable [4]. Even for a small problem size, the entire search space is huge. One simple example illustrates this. Mapping 16 cores onto 16 nodes has a search space of 16!, which is about 2.09×10^{13}. Suppose that each search takes 0.01 millisecond, to enumerate the entire search takes about 6.6 years. In general, there are two broad classes of approximation algorithms to solve the mapping problem, *systematic* search and *heuristic* search. A systematic approach attempts to constructively enumerate the solution space while a heuristic does not. A heuristic is a local search method relying on a neighborhood function to search near-optimal solutions [1]. Systematic approaches can adopt branch-and-bound or back-tracking algorithms [12]. In both methods, cutting branches that may not lead to a solution is the key to balance run time and the quality of solution. However, systematic approaches are difficult to scale with the problem size. The branch-and-bound is memory hungry while the back tracking is time consuming in general. In contrast to a systematic search, heuristics such as simulated annealing, tabu search, genetic algorithms and neural networks [1] are based on empirical techniques in nature. Simulated annealing (SA) [7] is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. It is inspired by annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. SA has been widely used in a large range of applications.

In the paper, we present a cluster-based simulated annealing that combines a clustering technique with simulated annealing to map cores onto 2D mesh NoCs. Such 2D tiled structures have gained popularity over the last six years of NoC research due to its modularity and regularity [2]. Clustering is a general technique to partition nodes into groups according to the node "*distance*" property. The distance can have a very different meaning depending on the problem of interest. Clustering can greatly simplify the mapping problem because the mapping can now be conducted cluster wise instead of node wise. Since a cluster can be much coarser, the mapping complexity can be greatly reduced. If we partition 16 nodes/cores into 4 clusters with each cluster accommodating 4 nodes/cores, the entire search space is shrunk to $(4!)^5$, which is approximately 7.96×10^6. However this is gained by sacrificing cross-cluster permutations. In our approach, we allow cross-cluster moves in the annealing stage. Therefore our technique does not limit search space but make the search more efficient. By clustering, nodes and cores with a certain property are grouped together. Thus, a good initial configuration can be achieved and the annealing process can be conducted more efficiently. Our experimental results show that this method saves run time without compromising solution quality and is scalable to larger problem sizes.

The rest of the paper is organized as follows. We outline related work in Section II. In Section III, we define the mapping problem and give an overview of our Cluster-based Simulated Annealing (CSA) approach. We present the CSA in detail in Section IV. Experimental results are then reported in Section V. Finally, we conclude in Section VI.

978-1-4244-2277-7/08/$25.00 ©2008 IEEE

II. Related Work

In the area of parallel computing, mapping algorithms or task clusters onto multiple interconnected processors have long been studied [8], [9]. They aimed to optimize the execution time under the circumstance that the application traffic knowledge is unknown. Since NoC is a continuation of System-on-Chip (SoC) and SoC is application specific, solving the NoC core-to-node mapping problem can take advantage of knowledge about both the traffic requirements and network properties. The following works address the NoC mapping problem.

Hu and Marculescu [6] formulated an IP-to-node mapping problem in presence of routing diversity to minimize energy. They used a branch-and-bound algorithm and decoupled the mapping and path selection phases. The UMARS (Unified MApping, Routing and Slot allocation) [5] is a single-objective algorithm unifying the IP-to-node mapping, path selection, and slot allocation for time-division-multiplexing virtual circuits. It is a greedy algorithm that iterates over a monotonically decreasing set of unmapped virtual circuits until all virtual circuits are allocated or until allocation fails. Lu and Jantsch [10] addressed the configuration of time-division-multiplexing virtual circuits using the concepts of logical networks while mapping IPs to cores.

Murali and De Micheli addressed the mapping problem with the aim of minimizing communication delay by exploiting the possibility of splitting traffic among diverse paths [11]. They used a heuristic (swapping vertexes based on initial mappings) to explore minimum-path routing and split-traffic routing. In [13], a genetic algorithm is used to map task graphs onto a tile-based NoC architecture with the objective of minimizing execution time.

We address the mapping problem using simulated annealing with the objective to minimize the delay-weighted bandwidth. Particularly, we combine the clustering technique with the simulated annealing, reducing run time without compromising quality of solution. To our knowledge, our work is the first to use a cluster-based simulated technique to address the core-to-node mapping problem for specific applications in the network-on-chip context.

III. The Mapping Problem and Our Approach

A. The Problem Definition

The core-to-node mapping problem is a specific graph embedding problem. We assume that

- the network topology is 2D mesh. The mesh has bidirectional links with uniform bandwidth between nodes.
- the number of cores is not greater than the number of nodes and one core is mapped to exactly one node. If there are more cores than nodes, two or more cores may be grouped into one coarser core before mapping.

The mapping problem has been formulated in [6] and [11]. Both formulations are similar but use a different cost function. Following their formulations, we first give definitions and then formulate the problem.

Definition 1. A Core Communication Graph (CCG) captures the communication patterns between IP cores. It is a directed graph $G = M \times A$, where each vertex m_i represents a core, and each directed arc $a_{i \to j}$ represents the communication from m_i to m_j and is associated with a bandwidth requirement $bw_{i,j}$ as its weight.

Definition 2. A Network Node Graph (NNG) reflects the connectivity and bandwidth capacity of the underlying implementation architecture. It is a directed graph $G' = N \times V$, where each vertex n_i represents a node, and each directed arc $v_{i \to j}$ represents the link from node n_i to node n_j and is associated with an available bandwidth $BW_{i,j}$ as its weight.

Given a CCG and an NNG, $size(CCG) \leq size(NNG)$, the mapping task is to find a mapping function: $map : M \to N$ that maps a core $m_i \in M$ to a node $n_i \in N$ such that the cost $Cost = \sum_{\forall a_{i \to j}} |R(map(m_i), map(m_j))| * bw_{i,j}$ is minimized, where $|R(map(m_i), map(m_j))|$ is the number of hops of the route from $map(m_i)$ to $map(m_j)$. Routing is governed by a routing algorithm which ensures minimal path and deadlock-free. The objective function minimizes delay and overall communication volume, thereby maximizing performance and minimizing power consumption in the network.

B. Overview of Cluster-based Simulated Annealing

We combine two techniques, namely, *clustering* and *simulated annealing*, to address the mapping problem. We call this combined technique *Cluster-based Simulated Annealing (CSA)*. An overview of the CSA is shown in Figure 1.

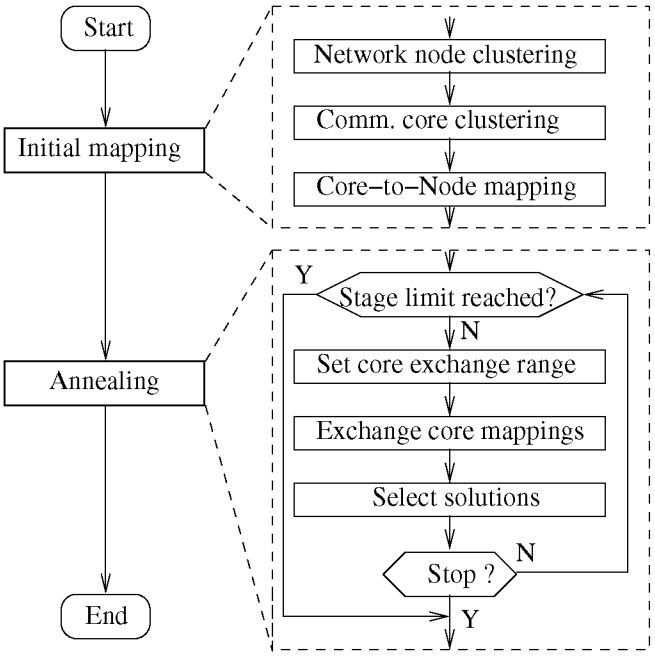

Fig. 1. The Cluster-based Simulated Annealing (CSA) Flow

The CSA flow is based on a typical SA flow, which consists of *initial mapping* and *annealing* steps. It enhances the SA flow by grouping network nodes and communication cores

into clusters before the core-to-node mapping. The clustering exploits the knowledge about the network architecture and communication demand of applications. Then the core-to-node mapping is to map core clusters to node clusters and find a feasible initial mapping, and later the annealing process is applied on the initial mapping in aware of the clusters. Therefore, the clustering technique would impact both the initial mapping and the annealing process of the SA. We describe the two steps in detail in Section IV.

IV. Cluster-based Simulated Annealing (CSA)

A. Cluster-based Initial Mapping

As depicted in Figure 1, the initial mapping consists of three steps, namely, *network node clustering*, *communication core clustering*, and *core-to-node mapping*. The first step is to cluster network nodes in the network topology. The second step is to cluster cores in the application communication graph. The second step uses the same number of clusters and the same number of nodes in each cluster as the first step. The third step is to make a one-to-one mapping between a core cluster and a node cluster. As the search is subject to bandwidth and delay constraints, both constraints will be checked in order to find a feasible initial mapping.

1) Network node clustering: The node clustering is based on the physical shortest distance between nodes in the network topology. We define the distance of node n_i as a tuple by c_i and h_i,

$$d(n_i) = (c_i, h_i) \tag{1}$$

where c_i is the number of output links of node n_i, and h_i is the overall distance from node n_i to all other network nodes. The parameter c_i reflects the node's local connectivity. It is important because a node with a larger number of neighboring nodes has a larger communication capability and thus should be preferably used by a core with a larger number of connections. In this way, communication locality can be better satisfied. h_i captures the global connectivity of node n_i and is calculated by

$$h_i = \sqrt{\sum_{j=1}^{K}((x_j - x_i)^2 + (y_j - y_i)^2)} \tag{2}$$

where K is the number of network nodes and (x_j, y_j) is the coordinates of node n_j on the mesh.

Using the distance property of nodes, the node clustering may be conducted with the following steps:

Step 1.1 Calculate each node's distance $d(n_i)$, and group nodes with the same distance into the same set;

Step 1.2 Sort the nodes in a descending order according to their distance. For the distance comparison with a tuple (c,h), the first element c will be first compared and then the second element h. $(c_1,h_1) > (c_2,h_2)$ if $c_1 > c_2$ or $c_1 = c_2$ and $h_1 > h_2$; $(c_1,h_1) < (c_2,h_2)$ if $c_1 < c_2$; $(c_1,h_1) = (c_2,h_2)$ if $c_1 = c_2$ and $h_1 = h_2$.

Step 1.3 Sequentially partition nodes into clusters.

In Step 1.3, the number of clusters should not be fixed as a prior. Rather it should be determined according to the distance distribution of nodes.

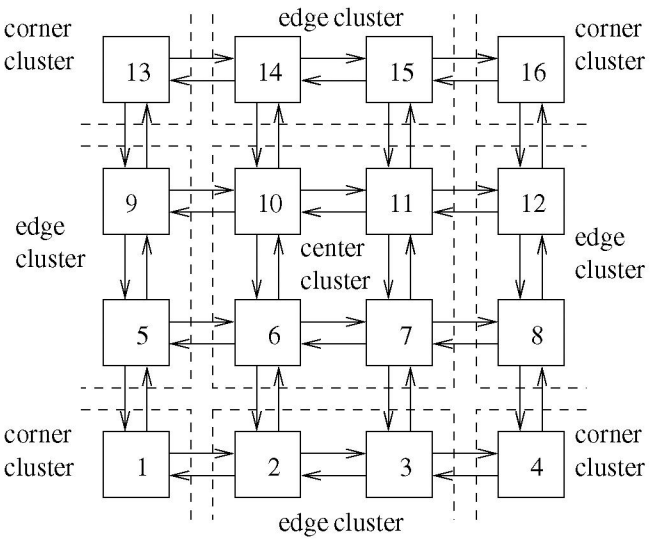

Fig. 2. A clustered 4 × 4 mesh

We exemplify the node clustering with a 4×4 mesh in Figure 2. By Step 1.1, we obtain three node sets, $\{6,7,10,11\}$, $\{2,3,5,9,8,12,14,15\}$, and $\{1,4,13,16\}$. In each set, nodes have the same distance. By Step 1.2, we order the three sets. Set $\{6,7,10,11\}$ includes nodes in the center. In this set, each node has four neighbors. Compared with the other two sets, this set has the highest communication capability. According to the communication capability, we order the three sets as a center set $\{6,7,10,11\}$, an edge set $\{2,3,5,9,8,12,14,15\}$, and a corner set $\{1,4,13,16\}$. Naturally, we obtain three clusters for Step 1.3 as a center cluster, an edge cluster and a corner cluster.

2) Communication core clustering: In this step, the number of clusters and the number of cores in each cluster, which are used to partition the core communication graph, inherit from network node clustering. The distance of a core depends on the number of connections and the communication bandwidth for each connection. To properly reflect these requirements, we define the distance of core m_i as a tuple by e_i and w_i,

$$d(m_i) = (e_i, w_i) \tag{3}$$

where e_i is the number of output connections of core m_i, and w_i is the weighted sum of the bandwidth requirement of core m_i to its neighbour cores. e_i reflects traffic locality. w_i captures the node's overall communication requirements and is calculated by

$$w_i = \sum_{j=1}^{l_i} k_{i,j} bw_{i,j} \tag{4}$$

where l_i is the number of connected cores with core m_i, $bw_{i,j}$ is the bandwidth requirement from core m_i to m_j, and $k_{i,j}$ is a user-adjustable parameter, thus a user can control the weight of a particular connection. Therefore the core distance may be biased toward some connections due to, for example, more stringent delay constraint on a particular connection.

The core clustering may be conducted as follows:

Step 2.1 Calculate each core's distance $d(m_i)$, and group cores with the same distance into the same set;
Step 2.2 Sort the cores in a descending order according to their distance. The distance comparison with tuple (e,w) is similar to that with (c,h).
Step 2.3 Sequentially partition the ordered cores into clusters, matching the number of elements in the corresponding node cluster.

3) Core-to-Node initial mapping: The initial mapping is to match core clusters with node clusters according to their distance property. The idea is to map a cluster of communication cores with high demand in delay and bandwidth onto a cluster of network nodes that can provide such high capability in delay and bandwidth. For example, the network nodes in the center of the 2D mesh network potentially have lower communication delay and higher bandwidth. Therefore, a cluster of cores with lower delay and higher bandwidth requirements should be mapped onto the center nodes.

In the initial mapping phase, bandwidth and delay constraints for each inter-core communication will be checked and must be satisfied. This implies that, after the cluster mapping is settled, cores within a cluster may need to adjust their mappings in order to satisfy the requirements.

B. Cluster-based Annealing

Besides the initial mapping, the annealing will also be affected by the clusters. We first describe the general annealing process and then the *cluster-aware* annealing, which considers both *inside-cluster (within a cluster boundary) annealing* and *cross-cluster (outside a cluster boundary) annealing*.

1) Annealing: The annealing technique is a cooling process, i.e., reducing the temperature, based on an initial configuration. In the mapping problem, the exchange of positions between two mapped cores reflects the temperature. If two mapped cores have a higher shortest distance on the topology, we say that the exchange of positions between the two cores is a higher temperature move. Otherwise, the exchange leads to a lower temperature move. Each move results in a new configuration. For the problem of mapping cores onto 2D meshes, the number of annealing stages thus corresponds to the diameter D of the network. For a $K_1 \times K_2$ mesh network, $D = K_1 + K_2 - 2$.

The annealing uses the initial mapping as the initial configuration. For a $K_1 \times K_2$ mesh, it first sets the number of annealing stages to diameter D, each stage corresponding to an exchange of mapped cores within a given distance. For instance, at the first stage, the mapped cores with distance $d \leq D$ may be exchanged. At the last stage, only neighbor cores ($d = 1$) are possibly exchanged. At each stage, the pair of mapped cores to be exchanged are randomly selected. A number of moves may be conducted at each stage. The resulting solutions will then be checked against bandwidth and delay requirements in order to filter out only feasible solutions. Moreover, only part of the feasible solutions can be accepted and are in turn used in the next stage annealing. This iterative process continues until a stop criterion is reached. The stop criterion can be either a time threshold or a sufficient number of feasible solutions reached.

The selection of feasible solutions for the next-stage annealing is a crucial step. On one hand, it should lead to low cost solutions, called "downhill" moves. On the other hand, it must be able to escape from local minimum. The way to do this is to allow also "uphill" moves in the annealing process, implying that a higher cost solution may be probabilistically accepted to the next stage annealing. We follow the Metropolis algorithm [3] for the solution selection. According to this method, the acceptance probability function is $P(\Delta E) = exp(-\Delta E/KT)$. Here K is the Boltzmann constant, T temperature and E energy. Both T and E are user defined. E refers to the cost of a mapping solution. By the equation, a decrease in the energy (negative ΔE) has an increasing acceptance probability, and an increase of energy (positive ΔE) has a decreasing acceptance probability. Our acceptance criterion is to accept all downhill moves and probabilistically accept uphill moves. The acceptance of uphill moves helps to escape from local minimum.

2) Cluster-aware annealing: The cluster-aware annealing follows the annealing procedure described above. However, in each stage, annealing must determine whether only inside-cluster moves are allowed. Let us assume that an annealing stage allows to swap nodes (thus core mappings) within distance d_0. After randomly choosing a candidate node, the annealing procedure first checks which cluster it belongs to and then finds the maximum distance d_1 to its cluster-mates *without crossing a cluster boundary*. If $d_0 \leq d_1$, only inside-cluster exchanges are permitted; otherwise, both insider-cluster and cross-cluster swappings are valid. This implies that, in the earlier annealing stages, the clusters are virtual and do not have impact on the annealing. This allows the greatest possible of optimal solution space. The cross-cluster annealing breaks the cluster boundaries, enabling to escape from local minimum and to further optimize cost. In the later annealing stages, the clusters are visible to make sure that only inside-cluster annealing is made. This helps to effectively constrain the search to converged near-optimal solutions by minimizing random displacements, which are unlikely lead to low cost solutions.

To show when only inside-cluster moves are allowed, we give an example. On the 4×4 mesh (Figure 2), at the annealing stage with a permissible move distance $d_0 = 2$, if node 7 is chosen, we find that this node belongs to the center cluster, and its maximum distance of this node to its cluster-mates, d_1, is 2, satisfying $d_0 = d_1$. Then only inside-center-cluster exchanges are valid. This means that the annealing only allows node 7 to swap with nodes 6, 10 and 11.

V. Experiments

The purpose of our experiments is to validate the advantages of our approach in runtime and quality of solution. To this end, we have also implemented the *simulated annealing (SA) without clustering* as the baseline. This enables us to compare the pure SA with the CSA. The SA and CSA algorithms are executed on a notebook with a 1.6 GHz processor and 768 MB memory. We assume the dimension-order XY routing, which is simple and guarantees deadlock freedom.

A. A Video Application

We use a Video Object Plane (VOP) decoder [11] to test our algorithms in runtime and quality of solution. As depicted in Figure 3, the VOP decoder has 16 cores. The bandwidth requirement is denoted on the connections. This application is to be mapped onto the 4×4 mesh.

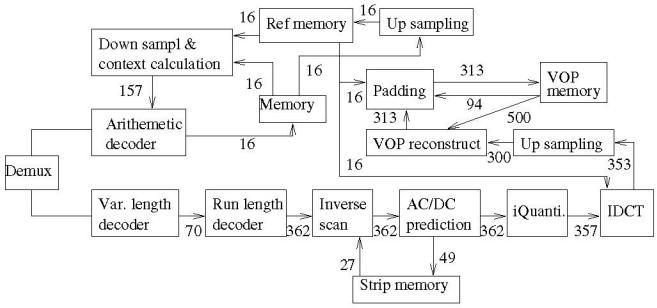

Fig. 3. The Video Object Plane decoder with bandwidth demand in MB/s

B. Results of Simulated Annealing (SA)

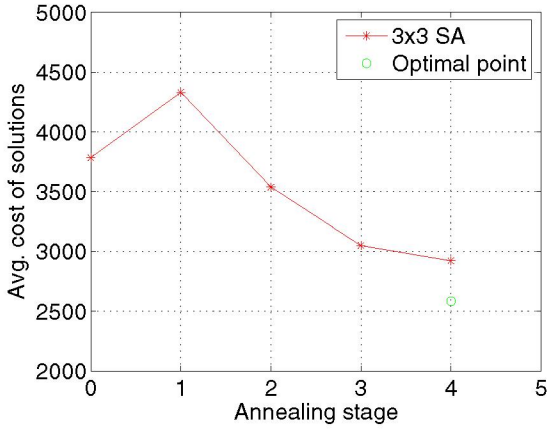

Fig. 4. 3×3 mapping results

We first validate the quality of the pure simulated annealing (SA). We use a partial of the video application (Figure 3), 9 cores to map onto a 3×3 mesh. We perform a complete search of the entire solution space in order to find the optimal solution. Such an exhaustive search is possible for the 3×3 case. Then we compare the results of SA with the optimal solution, as shown in Figure 4.

In Figure 4, the X axis represents the number of annealing stages, each corresponding to a distance range. "Stage= 0", refers to the initial mapping. "Stage= $1,2,3,4$" means that only a move within distance 4, 3, 2, 1 (an exchange of nodes between which the shortest distance is not greater than 4, 3, 2, 1), respectively, is permissible at the stage. The Y axis shows the average cost of accepted solutions. From the curve, we can see that the annealing results in converged low cost solutions. Interestingly, as indicated by the *Optimal point*, the minimum cost solution found in the last stage of annealing achieves the same optimal solution as the full search does. Note that the optimal solution is a single point. The SA takes 29 seconds to finish while the exhaustive search uses 80 seconds.

C. Cluster-based Simulated Annealing (CSA) Results

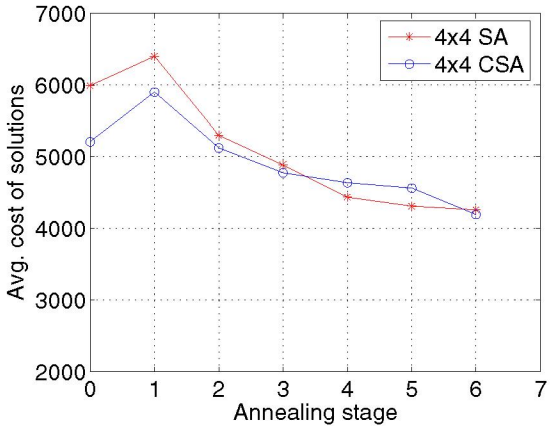

Fig. 5. 4×4 mapping results

In this section, we first compare the results of SA and CSA mapping the VOP decoder application onto a 4×4 mesh. As shown in Figure 5, both SA and CSA converge to the equilibrium state. The CSA achieves lower cost for the initial mapping and a slightly lower average cost (4189) than the SA (4255) in the last annealing stage.

To compare our results with NMAP [11], which is a heuristic mapping algorithm and could be a representative of the state-of-the-art, we extract approximate results from the bar diagrams in [11]. For the VOP application, NMAP achieves the lowest cost of about 4200. The minimum link bandwidth required to satisfy the bandwidth requirements of the application is about 450 MB/s. No execution time was reported in [11]. Our SA and CSA algorithms achieve the lowest cost of 4231 and 4169, respectively, with the same minimum required link bandwidth. Furthermore, the SA takes 497 seconds to finish while the CSA consumes 387 seconds, meaning a reduction of 22% in run time.

In [11], the studied problem sizes are no more than 16 cores. To show the performance of the cluster-based approach to a bigger network size, we conduct experiments on an 8×8 mesh. Due to lack of appropriate application, we create an application by quadruplicating the VOP decoder application

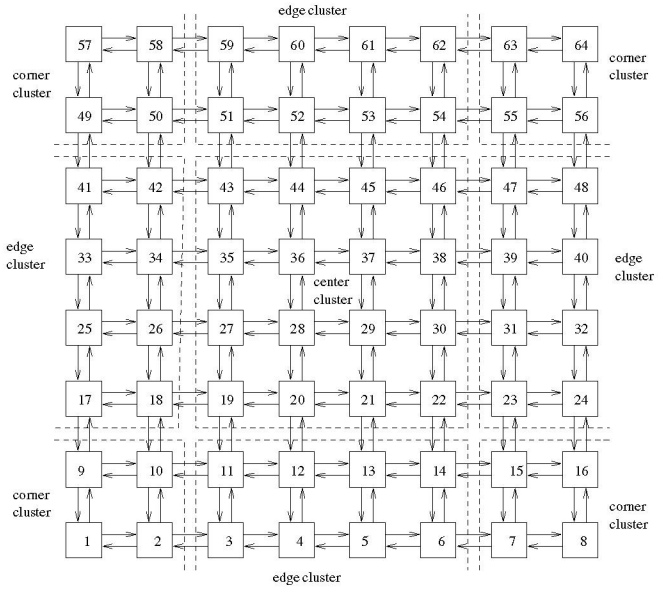

Fig. 6. A clustered 8×8 mesh

and placing additional links among the four instances. We coarsely partition the nodes into three clusters, a center cluster, an edge cluster and a corner cluster. Each cluster in this case is four times as big as the corresponding 4×4 cluster, as indicated in Figure 6. The results are shown in Figure 7. Both algorithms achieved the same lowest average cost in the last annealing stage. The SA takes 6785 seconds to finish, and the CSA completes in 4750 seconds, reducing 30% in run time.

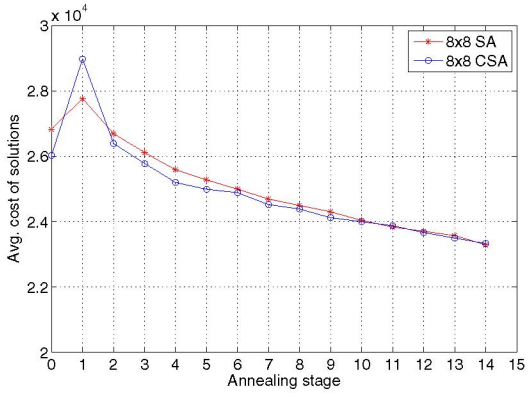

Fig. 7. 8×8 mapping results

VI. Conclusion and Future Work

In the paper, we propose to combine the clustering technique with the simulated annealing to further leverage the performance of SA. The clustering is based on the distance property of elements. In our mapping problem, the distance reflects the communication supply property in the network topology and the communication demand property in the core communication graph. By exploiting the distance knowledge from both the implementation architecture and application characteristics, we are able to obtain better initial mappings and later the annealing may dynamically constrain to inside clusters to speed up the convergence. The annealing can thus be more efficient. Our case study shows an up to 30% reduction in run time without compromising solution quality. We also show that our algorithms can effectively handle problems with a larger size.

While confident in the potential of our approach, we realize that clustering in itself has a rather huge design space to explore. For example, hierarchical clustering of network nodes and application cores may be applied when the system size scales up. Since a mesh has excellent composability, a big size mesh can be composed with smaller-size meshes by adding links without changing any existing links. For example, a 32×32 mesh may be assembled with 4 16×16 meshes or 16 8×8 meshes. This composability property can be explored using hierarchical clustering. Accordingly application cores must also be hierarchically clustered. This provides another angle of clustering and further impact on how to do annealing is to be studied in the future.

Acknowledgment

The research is partially supported by the EU project Mosart in the FP7 framework.

References

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons Ltd., 1997.

[2] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Survey*, 38(1):1–54, 2006.

[3] O. Catoni. Metropolis, simulated annealing, and iterated energy transformation algorithms: Theory and experiments. *Journal of Complexity*, 12(4):595–623, 1996.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[5] A. Hansson, K. Goossens, and A. Rădulescu. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, Sept. 2005.

[6] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proceedings of the Design Automation and Test in Europe Conference*, 2003.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[8] N. Koziris, M. Romesis, P. Tsanakas, and G. Papakonstantinou. An efficient algorithm for the physical mapping of clustered taskgraphs onto multiprocessor architectures. In *Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing*, pages 406–413, 2000.

[9] V. M. Lo, S. Rajopadhye, S. Gupta, D. Keldsen, M. A. Mohamed, B. Nitzberg, J. A. Telle, and X. Zhong. OREGAMI: Tools for mapping parallel computations to parallel architectures. *International Journal of Parallel Programming*, 20(3):237–270, 1991.

[10] Z. Lu and A. Jantsch. Slot allocation using logical networks for TDM virtual circuit configuration for network-on-chip. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'07)*, November 2007.

[11] S. Murali and G. D. Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings of Design, Automation and Test in Europe Conference*, pages 896 – 901, 2004.

[12] S. Sahni. *Data Structures, Algorithms, and Applications in C++, 2nd Edition*. Silicon Press, 2004.

[13] L. Tang and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of Euromicro Symposium on Digital System Design*, pages 180– 187, Sept. 2003.