Parameter-Optimized Simulated Annealing for Application Mapping on Networks-on-Chip

Bo Yang[1,3], Liang Guang[1], Tero Säntti[1,2], and Juha Plosila[1]

[1] Department of Information Technology, University of Turku, Finland
[2] Academy of Finland, Research Council for Natural Sciences and Engineering
[3] Turku Center for Computer Science, Turku, Finland
{boyan, liagua, teansa, juplos}@utu.fi

Abstract. Application mapping is an important issue in designing systems based on many-core networks-on-chip (NoCs). Simulated Annealing (SA) has been often used for searching for the optimized solution of application mapping problem. The parameters applied in the SA algorithm jointly control the annealing schedule and have great impact on the runtime and the quality of the final solution of the SA algorithm. The optimized parameters should be selected in a systematic way for each particular mapping problem, instead of using an identical set of empirical parameters for all problems. In this work, we apply an optimization method, Nelder-Mead simplex method, to obtain optimized parameters of SA. The experiment shows that with optimized parameters, we can get an average 237 times speedup of the SA algorithm, compared to the work where the empirical values are used for setting parameters. For the set of benchmarks, the proposed parameter-optimized SA algorithm achieves comparable communication energy consumption using less than 1% of iterations of that used in the reference work.

1 Introduction

In the past decade, the multi- and many-core processors have been rapidly developing and widely used for processing a massive amount of data in increasingly complex systems [2]. As the number of cores and their processing capabilities are continuously increasing, the communicational aspect, instead of the traditional computational aspect, is becoming the major concern in designing systems-on-chip (SoCs). The underlying communication architecture in many-core systems plays a great role in improving the performance and decreasing the energy consumption of the system. To deal with the emerging communication challenge in many-core system, the NoC has been proposed as a promising alternative to the conventional bus-based and point-to-point communication architectures [1].

Figure 1 shows an example of an 8-core SoC on a 2×4 2D mesh NoC. The NoC is a communication infrastructure composed of a set of routers connected by inter-router communication channels. The processing elements (PEs) such as CPU or DSP modules, FPGAs, embedded memory blocks, are connected to a router via the network interface (NI). Typically, the term node or tile on a NoC refers to a PE and the corresponding router. The data generated by the source PE is first transformed into packets coupled with appropriate control information, and then transmitted to the destination PE by traveling multiple routers and channels over the NoC. The routing decision is made on each router based on a specific routing protocol. By decentralizing the role of the arbitration into each router, the NoC architecture is suitable to deal with the great number of concurrent communications in modern many-core systems. The bandwidth on the NoC is also enhanced by sharing network channels among concurrent communications [4].

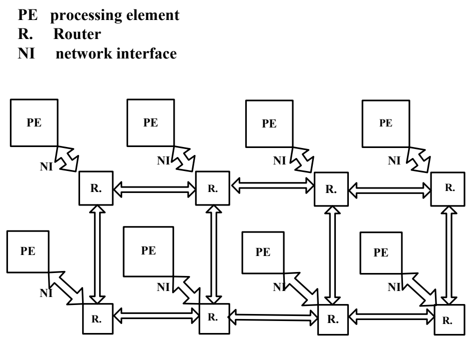

Fig. 1: An Example of 2D Mesh NoC

Based on the NoC communication architecture, there are set of problems related to many-core systems design. One of them is the *application mapping*. Given an application implemented by a set of tasks, and a many-core NoC, the problem of the application mapping is to decide how to map each task onto a node so that the predefined objectives and constraints can be met. The application mapping has a great impact on the system performance and energy consumption. The experiment in [7] shows that one optimized mapping algorithm achieves 51.7% communication energy savings compared to an ad hoc implementation. While exhaustive search is not possible for the *NP-hard* mapping problem, stochastic and heuristic searches are generally used for finding the near-optimal mapping solutions. As surveyed in [9], these stochastic and heuristic methods include, for example, simulated annealing (SA), tabu search (TS) and greedy incremental (GI) heuristic.

In these search heuristics, the SA has been often used since it is able to escape from the local minimum and find the global minimum of the dedicated cost function. The detail of the SA algorithm will be described in Section 4. To apply the SA algorithm, a set of parameters and functions needs to be specified, such as *initial temperature*, *final temperature*, *cooling ratio*, *temperature function*, *accept function*, etc. These parameters and functions determine how closely and how quickly SA can converge to the global minimum of the objective function. However, the fact is, there is no a straightforward way for specifying these parameters. In previous works, these parameters are set either by using empirical values [7] [13], or decided by the specific characteristics of a particular problem [11]. Apparently, we cannot make sure that these empirical or problem-specific values could be generally applicable to all other problems. Without a systematic way, the parameters of SA are usually randomly selected, and as a result, the quality of the set of parameters is not guaranteed.

A systematic method is necessary to generate the parameters of the SA algorithm for solving the application mapping problem, even though it has not so far been mentioned in literature. In this work, we propose to use the *Nelder-Mead simplex method*, which is originally introduced in [8], to automatically generate the optimized parameters of SA. The generated parameters are applied to the SA algorithm to find the optimized mapping solution which achieves minimized communication energy consumption on the NoC. In this work, by using the set of optimized parameters, we target to utilize less evaluations in SA while achieving comparably good quality of the final solution with the reference work.

The rest of the paper is organized as follows. Section 2 reviews previous works which use the SA algorithm for application mapping and outlines our considerations and contributions in this work. In Section 3, the problem of application mapping is modeled and the objective function is defined. Section 4 describes the general SA algorithm and Section 5 presents the Nelder-Mead simplex method. The proposed parameter-optimized SA algorithm is presented in Section 6. The experimental results are demonstrated in Section 7. Section 8 summarizes this paper.

2 Related Work

In [7], the authors developed an energy-aware mapping algorithm, namely Branch and Bound, for 2D mesh NoCs. The SA algorithm is implemented as a reference to evaluate the Branch and Bound algorithm. The comparison shows that SA can find the better mappings which achieve lower communication energy consumption than the ones found by the proposed Branch and Bound algorithm. However, the major drawback of SA is speed. The result in [7] shows that for a video/audio application, SA is 82 times slower than the Branch and Bound algorithm. The initial temperature was set 100 and the cooling ratio , 0.9. The final temperature was not set because a different termination criteria was used.

In [13], the authors tried to speed up SA by optimizing the number of iterations per temperature level and the core swapping process as well. A both NoC- and application-aware iteration number is used. In addition, to generate a neighboring solution, two cores are selected and swapped based on the possibility distribution function, instead of on an uniformly random possibility. The optimized SA is claimed to be 98% faster at the price of 13% memory consumption on average. In this work, the initial temperature, final temperature and cooling ratio are 1, 0.001 and 0.9 respectively.

In [11], the authors used SA to map an application on multiprocessor system-on-chip (MPSoC). The cooling ratio was set 0.95. Two functions, which are based on the execution time of each task defined in the task graph, were introduced to derive the initial and final temperature. It shows that by using parameter obtained by these two functions, the proposed method saves over half the optimization time and loses only 0.3% in performance.

We can see that for the application mapping problem, the parameters of SA were selected randomly by authors in these works. In this way, whenever we encounter a particular application mapping problem, we have to decide which set of parameters we should use. Since we are not able to determine which one is best suitable for our specific problem, the decision is difficult to make.

For the parameters used in the SA algorithm, we believe that:

1. The parameters of SA are problem-specific. One set of parameters would not be appropriate to other problems. The selection of the set of parameters has to be done with respect to the particular problem.
2. The parameters of SA have a joint impact on the performance of SA. This means that these parameters should be selected systematically, instead of being set independently.

Based on these two considerations, in this work, we apply the Nelder-Mead simplex method to systematically select the parameters of SA for the application mapping problem. The original Nelder-Mead simplex method is presented in [8] with purpose of finding the minimal value of a function of n variables. By using the Nelder-Mead simplex method, both the parameters to be selected and the cost function used in the selecting process can be defined with respect to the particular application mapping problem. This provides flexibility for us to use the SA algorithm for solving application mapping problems with single or multiple objectives. Moreover, since the set of parameters is selected by the systematic procedure for the particular problem, instead of being set by empirical values, the selected set of parameters is problem-specific and their qualities are guaranteed. To our best knowledge, this is the first work on investigating the systematic way of selecting optimized parameters of SA in the application mapping problem domain.

3 Application Mapping Problem

3.1 Application and NoC Model

The inputs of the mapping problem consist of two parts: one is the application and another is the many-core NoC. In this work, the application is modeled by a communication weighted graph (CWG), and the many-core NoC by a computation and communication resource graph (CCRG). For the sake of simplicity, in this paper, we use a 2D mesh NoC with homogeneous cores as the target computation and communication platform. We note that the method presented in this work is also applicable to other 2D NoCs with regular or irregular topologies. The X-Y deterministic routing is adopted by which a flit is first routed to the X direction and then the Y direction over the NoC.

Definition 1 A CWG is a directed graph $< V, E >$, where $V = \{v_1, v_2, \ldots, v_M\}$ represents the set of tasks of an application, corresponding to the set of CWG vertices, and $E = \{(v_i, v_j) | v_i, v_j \in V\}$ denotes the set of communications between tasks, corresponding to the set of CWG edges. Each edge (v_i, v_j), denoted by e_{ij} has weight vol_{ij} representing the total communication amount, in bits, transmitted from task v_i to v_j. M denotes the total number of tasks.

Definition 2 A CCRG is a directed graph $< TL, CH >$, where $TL = \{tl_1, tl_2, \ldots, tl_N\}$ denotes the set of tiles on the NoC, corresponding to the set of CCRG vertices. N denotes the total number of tiles. $CH = \{(tl_i, tl_j) | tl_i, tl_j \in TL\}$ designates the set of communication channels between tiles. The length of communication channel (tl_i, tl_j), denoted by $|ch_{ij}|$ is represented by the number of hops from tile tl_i to tl_j. On a 2D mesh NoC with X-Y routing, the length of the communication channel between tiles tl_i and

tl_j is calculated as follows:

$$|ch_{ij}| = |x_i - x_j| + |y_i - y_j| \tag{1}$$

where (x_i, y_i) and (x_j, y_j) are the coordinates of the tile tl_i and tl_j on a 2D mesh NoC respectively.

3.2 Objective Function

Using the preceding application and NoC model, the mapping of CWG to CCRG is defined by the one-to-one task-tile mapping function $map : V \to TL$:

$$map(v_i) = tl_i, \forall v_i \in V, \exists tl_i \in TL \tag{2}$$

When two tasks v_i and v_j of an edge e_{ij} in CWG are mapped on two tiles on the CCRG, an amount of vol_{ij} data will be transferred from the tile tl_i ($map(v_i)$) to the tile tl_j ($map(v_j)$). Based on the energy model proposed in [7], the communication energy consumption of e_{ij} is

$$E_{ij} = vol_{ij} \times (|ch_{ij}| \times E_{S_{bit}} + (|ch_{ij}| - 1) \times E_{L_{bit}}) \tag{3}$$

where $E_{S_{bit}}$ and $E_{L_{bit}}$ refer to the energy consumed by the switch and the link for transmitting one bit of data. And the total energy consumed by the application defined in CWG is

$$E_{app} = \sum_{\forall e_{ij} \in E} E_{ij} \tag{4}$$

From Equation 3 and 4, we can see that, given the constants $E_{S_{bit}}$ and $E_{L_{bit}}$, the communication energy consumption of an application is linearly proportional to the product of the data volume vol_{ij} and the length of communication channel $|ch_{ij}|$. In this work, since the objective of application mapping is to minimize the E_{app}, we need to minimize the sum of the product of vol_{ij} and $|ch_{ij}|$ for all communications in an application. Herein, we define the weighted communication of an application (WCA) as the objective function to evaluate the quality of each candidate mapping.

Definition 3 The Weighted Communication of an Application (WCA) is the sum of products of the data volume vol_{ij} and the length of communication channel $|ch_{ij}|$ for all communications in E.

$$WCA = \sum_{\forall e_{ij} \in E} vol_{ij} \times |ch_{ij}| \tag{5}$$

A mapping solution which can produce smaller WCA is considered to be a better solution because it will in turn yield a lower E_{app}.

4 Simulated Annealing

Simulated annealing is a stochastic search method for optimization problem. The pseudocode of the general SA algorithm, derived from [5], is shown in Algorithm 1. The symbols and corresponding definitions used in Algorithm 1 are listed in Table 1. SA simulates the metallurgical process of heating up a solid and then cooling down slowly until it crystallizes. It starts from an initial, higher temperature and stops at a final, lower temperature. An initial solution and its cost are given at the initial temperature. Thereafter, at each temperature, SA tries L times of attempt mappings. In each attempt, a new mapping solution is generated from current one using the move function $Move(S, T)$. The cost of the new solution is compared with current cost. The algorithm always accepts a move with lower cost. Contrary to the greedy algorithm, SA accepts a worse move with higher cost by a changing possibility. This helps to avoid local minimum and find the global minimum. The accept possibility is decided by acceptance function $Accept(\Delta C, T)$ and decreases along with the temperature. As temperature cools down, SA gradually becomes greedy and converges to the global minimum.

Algorithm 1 General Simulated Annealing Algorithm

```
1  $S \leftarrow S_0$
2  $C \leftarrow Cost(S_0)$
3  $S_{best} \leftarrow S$
4  $C_{best} \leftarrow C$
5  $R \leftarrow 0$
6  for $i \leftarrow 0\ to\ \infty$ do
7     $T \leftarrow Temp(i)$
8     $S_{new} \leftarrow Move(S, T)$
9     $C_{new} \leftarrow Cost(S_{new})$
10    $\Delta C \leftarrow C_{new} - C$
11    if $\Delta C < 0\ or\ Accept(\Delta C, T)$ then
12       if $C_{new} < C_{best}$ then
13          $S_{best} \leftarrow S_{new}$
14          $C_{best} \leftarrow C_{new}$
15       end if
16       $S \leftarrow S_{new}$
17       $C \leftarrow C_{new}$
18       $R \leftarrow 0$
19    else
20       $R \leftarrow R + 1$
21       if Terminate(i, R) = True then
22          $break$
23       end if
24    end if
25 end for
26 return $S_{best}$ and $C_{best}$
```

5 Nelder-Mead Simplex Method

The Nelder-Mead simplex method is proposed in [8] for the minimization of a function $f(p)$ with n variables $x_1, x_2, \ldots, x_n$. In this method, a number of $n + 1$ points (solutions) $p_0, p_1, \ldots, p_n$ are originally selected and form the so-called simplex. The set of points are then used to generate a new and better point which will replace the worst point in current simplex and forms a new simplex. Each point of the simplex is a n-tuple with n variables, i.e., $p_k = \left(x_1^k, x_1^k, \ldots, x_n^k\right)$. The Nelder-Mead simplex method compares the $n+1$ function values $f(p_i)\,(0 \leq i \leq n)$ and replaces the point with largest cost by the newly generated point. In each iteration, the replacement is realized by three operations: *reflection, expansion* and *contraction*. If it fails to do the replacement through these three operations, all points forming the simplex are updated with new values to generate a new simplex. The general Nelder-Mead simplex method is described in Algorithm 2.

Algorithm 2 Nelder-Mead Simplex Method for Minimizing $f(p)$

```
1  Select the initial $n+1$ points $p_i$ $(0 \leq i \leq n)$.
2  while $(!stop())$ do
3      Sort $f(p_i)$ $(0 \leq i \leq n)$ such that $f(p_0) \leq f(p_1) \leq \cdots \leq f(p_{n-1}) \leq f(p_n)$.
4      Let $\overline{p} = \sum_{i=0}^{n-1} p_i/n$.
5      Generate $reflection\ point$ $p_r = \overline{p} + \alpha * (\overline{p} - p_n)$.
6      if $f(p_r) \leq f(p_{n-1}))$ then
7          Replace $p_n$ by $p_r$.
8          Generate $expansion\ point$ $p_e = \overline{p} + \beta * (p_r - \overline{p})$.
9          if $(f(p_r) < f(p_0)) \wedge (f(p_e) < f(p_r))$ then
10             Replace $p_n$ by $p_e$.
11         end if
12     else
13         Let $f(p^*) = min(f(p_r), f(p_n))$.
14         Generate $contraction\ point$ $p_c = \overline{p} + \gamma * (p^* - \overline{p})$.
15         if $f(p_c) \leq f(p^*)$ then
16             Replace $p_n$ by $p_c$.
17         else
18             Update $p_j$ with $(p_j + p_0)/2$ for $j = 0, 1, \ldots, n$.
19         end if
20     end if
21 end while
22 Return the point $p_0$.
```

As shown in Algorithm 2, the principle of the Nelder-Mead simplex method is, if $f(p_r) \leq f(p_{n-1})$, then the point p_n is replaced by its reflection point p_r. Thereafter, if $f(p_r) < f(p_0)$, the reflection point is expanded to the expansion point p_e and the point p_n is replaced by p_e. The procedure restarts when the expansion is done. In the case that $f(p_r) > f(p_{n-1})$, the contraction point p_c is generated. If $f(p_c) < min\,(f(p_r), f(p_n))$, the point p_n is replaced by contraction point p_c. Otherwise, all points in current simplex are updated by $p_j = (p_j + p_0)/2 (j = 0, 1, \ldots, n)$ and a new simplex is generated. Thereafter the process restarts.

By continuously replacing the point p_n with a point which achieves smaller $f(p)$, the value of the function $f(p)$ converges to the minimum. The process terminates when the function $stop()$ becomes true. The state of function $stop()$ can be determined by whether the value of function $f(p)$ has converged to a final value [8], or whether the points forming the simplex have already converged to a final point [12]. In this work, because we try to find the optimized parameters for the SA algorithm, we adopt the latter way to define the function $stop()$. More precisely, in Algorithm 2, $stop()$ becomes true when $|x_i^k - x_j^k| \leq \varepsilon^k (i \neq j)$, for all i, j and k, where x_i^k and x_j^k are the kth element of point p_i and p_j respectively. Each element of vector ε, called *convergence degree* of variable x, is a predefined small positive value which determines the magnitude of the convergence.

In Algorithm 2, reflection coefficient α, expansion coefficient β and contraction coefficient γ give the factors by which the new simplex is generated by reflection, expansion and contraction respectively. These coefficients decide the speed of the convergence and the quality of the final point. In [8] and [12], different values of α, β and γ were used. In this work, we evaluated both sets of values by applying them in the Nelder-Mead simplex method for the same set of benchmarks. The result shows that both sets of parameters achieve comparable performance of the SA algorithm, but the Nelder-Mead simplex method using the coefficients in [12] can converge to the final point with 100 times less CPU time than that in [8]. Therefore, we use 1/3, 2.0 and 1.5 in [12] for α, β and γ respectively.

6 Parameter-Optimized Simulated Annealing

6.1 Parameters and Functions in SA

As shown in Algorithm 1, to apply SA to the application mapping problem, a number of parameters and functions have to be specified. In this section, we specify the parameters and functions used for implementing the SA algorithm in this work.

Cost Function The objective function of application mapping, i.e., the WCA defined in Equation (5), is used as the cost function $Cost(S)$ in SA.

Annealing Schedule: $\boldsymbol{Temp(i)}$ Function The annealing schedule determines how the temperature is cooling down. At each step of annealing, a new temperature is generated by temperature function $Temp(i)$. We choose the geometric annealing schedule presented in [5] where the $Temp(i)$ is defined as:

$$Temp(i) = T_0 \times q^{\left\lfloor \frac{i}{L} \right\rfloor} \tag{6}$$

The new temperature is decided by the initial temperature T_0, the cooling ratio q, the accumulated number of iterations i and the number of iterations at each temperature L.

Number of Iterations $\boldsymbol{L}$ The number of iterations at each temperature L is identically set as $M(N-1)$, where M and N are the number of tasks in CWG and that of tiles in CCRG respectively.

Acceptance Function: $\boldsymbol{Accept(\Delta C, T)}$ While an improving move ($\Delta C < 0$) is always accepted, the function $Accept(\Delta C, T)$ determines whether a worse move ($\Delta C > 0$) should be accepted or not at the temperature T. The normalized inverse exponential form is chosen to implement the acceptance function in this work.

$$\begin{aligned} Accept(\Delta C, T) = True \Leftrightarrow random() < p \\ p = \frac{1}{1 + \exp\left(\frac{\Delta C}{K C_0 T}\right)} \end{aligned} \tag{7}$$

With this acceptance function, the possibility of accepting a worse move, p, is less than 50%. On the basis of the original normalized inverse exponential form presented in [5], we add the normalizing ratio K in the acceptance function which works together with the initial cost C_0 to normalize the cost difference ΔC. This comes from the observation that using the original normalized inverse exponential form , in cases that the C_0 is huge, an accepting possibility close to 50% will be created even for a very small ΔC at a very low temperature. This makes SA inefficient at the last set of lower temperatures.

Initial and Final Temperature The acceptance function in (7) defines the relation between the accepting possibility p, cost difference ΔC and temperature T. Equation (7) can be solved with respect to T as follows:

$$T = \frac{\Delta C}{\ln(\frac{1}{p} - 1)} \tag{8}$$

If we define P_s the possibility of accepting the maximal ΔC at initial temperature T_0, and P_f the possibility of accepting the minimal ΔC at final temperature T_f, then the initial and final temperature can be calculated as follows:

$$T_0 = \frac{\Delta C_{max}}{\ln(\frac{1}{P_0} - 1)}, T_f = \frac{\Delta C_{min}}{\ln(\frac{1}{P_f} - 1)} \tag{9}$$

In the way that the T_0 and T_f are set manually using empirical values (the cases in [7] [13]), only a numerical range is given by T_0 and T_f, there are no realistic meanings behind T_0 and T_f. On the contrary, in this work, the usage of P_s and P_f is more meaningful and understandable for designers to choose the T_0 and T_f by Equation (9).

Move function:$\boldsymbol{Move(S,T)}$ We use the random swapping as the move function. A task in current mapping is randomly selected and then it is swapped with another randomly selected task.

Termination function:$\boldsymbol{Terminate(i,R)}$ We add one criteria $N_{\Delta C=0}$ into the termination function of coupled temperature and rejection threshold which is presented in [5], to determine the stopping condition in this work.

$$\begin{aligned} Terminate(i,R) = True \Leftrightarrow (temp(i) < T_f \wedge R \geq R_{max}) \\ \vee (N_{\Delta C=0} = Z) \end{aligned} \tag{10}$$

$N_{\Delta C=0}$ stands for the number of consecutive temperatures at which the lowest cost C_{best} has not been changed. Z is the maximal number of $N_{\Delta C=0}$ allowed in the SA algorithm. R is the number of consecutive rejections since last acceptance and R_{max} is the maximal number of rejections allowed in the SA algorithm. With this termination function, the annealing is stopped either when the temperature reaches to or below the final temperature and the moves in last R_{max} iterations are rejected, or in the last Z temperatures, no better solutions have been found. In this work, we set $R_{max} = L$ and $Z = 0.1N_T$, when N_T stands for the total number of temperatures from T_0 to T_f.

Initial Mapping A random mapping in which each task is randomly mapped on a tile, is generated as the initial mapping.

Summary of Parameters Table 1 summarizes the parameters and functions used in the SA algorithm in this work. We can see from Table 1, to apply the SA algorithm in this work, the values of 6 parameters need to be specified: $q, K, P_s, P_f, \Delta C_{max}$ and ΔC_{min}. As long as these parameters are specified, other parameters such as T_0 and T_f can be decided and all functions can work properly. In these 6 parameters, the values of ΔC_{max} and ΔC_{min} can be obtained from a set of mapping trials generated from the original mapping using the move function (see details in Section 7). The other 4 parameters, labeled "Nelder-Mead Simplex Method" in column "Value" in Table 1, are the most important parameters of SA in this work and they are going to be optimized by the Nelder-Mead simplex method presented in Section 5.

Table 1: Functions and Parameters for SA

Symbol	Definition	Value
S	Mapping solution (S_0: initial solution)	
$Cost(S)$	Cost function	WCA (Equation (5))
$Temp(i)$	Temperature function$i$	$T_0 \times q^{\lfloor \frac{i}{L} \rfloor}$
i	Accumulated number of iterations	
q	Geometric annealing schedule cooling ratio	Nelder-Mead Simplex Method
L	Number of iterations at each temperature	$M(N-1)$
N	Number of tiles in CCRG	
M	Number of tasks in CWG	
$Accept(\Delta C, T)$	Return accept (True) or reject (False) for a worse move	$random() < 1/(1 + \exp \frac{\Delta C}{KC_0T})$
K	Normalizing ratio	Nelder-Mead Simplex Method
C_0	Initial cost	$Cost(S_0)$
T_0	Initial temperature	$\Delta C_{max} / \ln(\frac{1}{P_s} - 1)$
T_f	Final temperature	$\Delta C_{min} / \ln(\frac{1}{P_f} - 1)$
ΔC_{max}	The maximal ΔC at initial temperature T_0	Experiment
ΔC_{min}	The maximal ΔC at final temperature T_f	Experiment
P_0	The possibility of accepting the maximal ΔC at initial temperature T_0	Nelder-Mead Simple Method
P_f	The possibility of accepting the minimal ΔC at final temperature T_f	Nelder-Mead Simplex Method
$Move(S, T)$	Return a neighboring mapping of S	
$Terminate(i, R)$	Return terminate (True) or continue (False)	$temp(i) < T_f \wedge R \geq R_{max} \vee N_{\Delta C=0} = Z$
R	Number of rejections	
R_{max}	Allowed maximal number of rejections	L
N_T	The total number of temperatures from T_0 to T_f	$\ln(\frac{T_0}{T_f}) / \ln(q)$
Z	The allowed maximal number of temperatures with $\Delta C = 0$	$0.1N_T$

6.2 Parameter Optimization

To apply the Nelder-Mead simplex method to get the optimized parameters q, K, P_s and P_f, we need to define the function $f(p)$ and specify the boundaries and convergence degree of each parameter from which the parameter is chosen.

Function $f(p)$ Since using various sets of q, K, P_s and P_f, the SA algorithm will produce different minimized values of WCA, we can define the output of the SA, i.e., the minimized WCA, as the function $f(p)$ of variables q, K, P_s and P_f. With this definition, it is possible to use the Nelder-Mead simplex method for finding the final point of variables q, K, P_s and P_f which produces the minimum WCA by SA.

Parameter Boundaries and Convergence Degrees Contrary to the work in [8] where the variables x is unbounded, the parameters q, K, P_s and P_f in our specific application mapping problem are bounded. Among them, the parameters P_s and P_f are theoretically in the range $(0.0, 0.50]$ according to Equation (7). For the SA algorithm, it is reasonable to set a higher acceptance possibility at the initial temperature and a relatively lower acceptance possibility at the final temperature. In this work, we set the range of P_s and P_f by $[0.20, 0.49]$ and $(0.0, 0.10]$ respectively. And the convergence degrees ε_{P_s} and ε_{P_f} are set 0.01 and 0.005 respectively. The cooling ratio q is supposed to be in range $(0.0, 1.0)$. In this work, we set the range $[0.80, 0.99]$ for q. The convergence degree ε_q is set 0.005. The value of K is allowed in the range of $(0.0, 1.0]$ and the convergence degree ε_K is set 0.05.

At the beginning of the Nelder-Mead simplex method, 5 initial points are generated by choosing 4 elements, i.e, q, K, P_s and P_f, from their allowable range. During the process of the Nelder-Mead simplex method, whenever an element of a point exceeds its boundary, the bound value is used for the element. The function $stop()$ becomes true and the process is terminated when these 5 points converge to one point.

6.3 Parameter-Optimized Simulated Annealing Algorithm

Algorithm 3 Parameter-Optimized Simulate Annealing

```
1 Define the boundaries and convergence degree $\varepsilon$ for parameters $q, K, P_s$ and $P_f$.
2 Obtain the final point of the Nelder-Mead simplex method, $p_{opt} = simplex()$.
3 Set $q_{opt} = p_{opt}.q, K_{opt} = p_{opt}.K$.
4 Set $P_{s_{opt}} = p_{opt}.P_s, P_{f_{opt}} = p_{opt}.P_f$.
5 Find the best solution by applying the final point to SA,
  $S_{best} = sa(q_{opt}, K_{opt}, P_{s_{opt}}, P_{f_{opt}})$.
6 Return $S_{best}$.
```

Applying the Nelder-Mead simplex method, we develop the parameter-optimized simulated annealing algorithm for application mapping problems on many-core NoCs. The proposed algorithm is described in Algorithm 3 where the function $sa()$ and $simplex()$ apply the Algorithm 1 and 2 respectively. After defining the boundaries and convergence degree for four target parameters, i.e., q, K, P_s and P_f, the optimized set of parameters are obtained by the Nelder-Mead simplex method. The optimal mapping solution with minimized WCA is then found by running the SA algorithm with the optimized set of parameters.

Note that, with various implementations of the SA algorithm, the parameters needed to be selected are different. Since the variables and objective functions applied in the Nelder-Mead method can be arbitrary, the Algorithm 3 is applicable for obtaining different sets of optimized parameters corresponding to different implementations. This makes the method proposed in this work viable for selecting optimized parameters of the SA algorithm which deals with diverse problems.

7 Experiment

To evaluate the efficiency of the proposed parameter-optimized simulated annealing (POSA) algorithm, we experiment POSA with a set of benchmarks and compare with the implementation of the SA algorithm in [7].

7.1 Setup

The implementation of the SA algorithm in [7] is available in the NoCmap project [3]. In the NoCmap, the geometric annealing schedule is used and the q is set 0.9. T_0 is fixed to 100 and the final temperature T_f is unbounded. The objective of the NoCmap is to minimized the total communication energy consumption and the energy model presented in [7] is adopted in the simulator. In this work, we also use the NoCmap simulator to obtain the communication energy consumption of the mappings generated by the POSA algorithm.

Four benchmark applications are selected for the comparison, including a video object plane decoder (VOPD) and a MPEG4 from SUNMAP [10], a multimedia systems application (MMS) [6] and a H.264 decoder (H264) [14]. The CWGs of these applications are derived from original descriptions in these works. The benchmarks and corresponding NoCs used in this work are summarized in Table 2.

The optimized mapping of each benchmark is found both by the NoCmap and the POSA algorithm. The communication energy of both mappings are produced by the NoCmap simulator. For POSA, the average ΔC_{max} and ΔC_{min} are obtained from $5 * L$ move trials starting from the original random mapping, which are used to calculate the T_0 and T_f with given parameters P_0, P_f, C_0 and K. Both algorithms were executed on a Desktop PC having a 3.0 GHz Intel Core2 Duo CPU and 8.0 GB of memory.

7.2 Results

Table 2: Optimized Parameters of SA for Benchmarks

Benchmark	Cores	NoC	q	P_0	P_f	K
VOPD	16	4x4	0.91	0.44	0.05	0.72
MPEG4	12	3x4	0.95	0.34	0.05	0.36
MMS	25	5x5	0.94	0.36	0.05	0.62
H264	16	4x4	0.89	0.42	0.05	0.49

Optimized Parameters Applying the POSA algorithm, the optimized mapping solution with minimized WCA of each application is achieved. At the same time, the optimized parameters of the SA algorithm are obtained. Table 2 shows the optimized parameters of SA for mapping the four benchmarks. As mentioned in Section 2, the parameters of SA are problem-specific. Table 2 illustrates that, instead of using an identical set of parameters, to find an optimized mapping, different parameters should be used in SA for mapping different applications.

Iterations and Runtime Table 3 shows the iterations (I_s) of NoCmap and POSA algorithm for finding the final mapping solution of each application. The column T_0

and T_f are the initial and final temperature respectively. T_t refers to the temperature at which SA terminates. I_s is the number of iterations that the SA algorithm has run until it terminates. pct is the percentage of the iterations of POSA to that of NoCmap. We can see that, since optimized parameters are applied, a much lower initial temperature is set in POSA. As a result, POSA uses significantly smaller number of iterations which is on average less than 1% of that used in NoCmap, to get the final mapping.

Table 3: Iterations of SA for Benchmarks

Benchmark	T_0		T_f		T_t		I_s		
	NoCmap	POSA	NoCmap	POSA	NoCmap	POSA	NoCmap	POSA	pct
VOPD	100	2.69	-	1.35e-4	2.28e-6	9.88e-5	4.30e6	2.74e4	0.64%
MPEG4	100	1.90	-	1.26e-4	5.80e-7	8.38e-5	2.61e6	2.77e4	1.06%
MMS	100	1.36	-	1.26e-5	5.80e-7	1.25e-5	1.14e7	1.18e5	1.04%
H.264	100	3.11	-	1.94e-4	0.15	1.43e-4	1.61e6	1.94e4	1.02%

Table 4 shows the runtimes of SA in NoCmap and POSA (in seconds) and the speedup achieved by POSA. POSA is, on average, 1.41 times faster than that in NoCmap. Note that, the runtime of POSA includes the time consumed by the Nelder-Mead simplex method in which the SA is run more than hundred times. In terms of the runtime of a single run of SA, a significant speedup is achieved by POSA due to less evaluating iterations. In Table 4, $POSA^{'}$ and $Speedup2$ represent the runtime of a single run of SA applying the set of optimized parameters, and the speedup over that in the NoCmap respectively. We can see that in POSA, the SA with optimized parameters is on average 237 times faster than that in NoCmap. This indicates how important the selection of parameters is regarding to the runtime of the SA algorithm.

Table 4: Runtimes and Speedup for Benchmarks

Benchmark	NoCmap	POSA	Speedup1	POSA´	Speedup2
VOPD	31.69	15.50	2.04	0.087	364
MPEG4	15.74	9.67	1.63	0.059	267
MMS	171.74	181.75	0.94	1.17	147
H.264	12.34	11.90	1.04	0.072	171
Average	-	-	1.41	-	237

WCA and Energy Consumption In this work, minimizing communication energy consumption on NoC is the objective of applying SA to solve the application mapping problem. Figure 2 shows the WCA achieved by NoCmap and POSA for each application respectively. The results of both algorithms vary slightly. The maximum of WCA variance is less than 4% in the case of application H.264.

As anticipated from the results of the minimized WCA, the communication energy consumptions achieved by NoCmap and POSA are almost same. Figure 3 shows that the maximal difference exists again in the case of application H.264, which is less than 2%. The comparable energy consumption verify the efficiency of the proposed POSA algorithm. Although a significantly smaller number of iterations is processed, POSA still can find the optimized mapping solution which is similar to the one found in NoCmap where a huge searching space is explored.

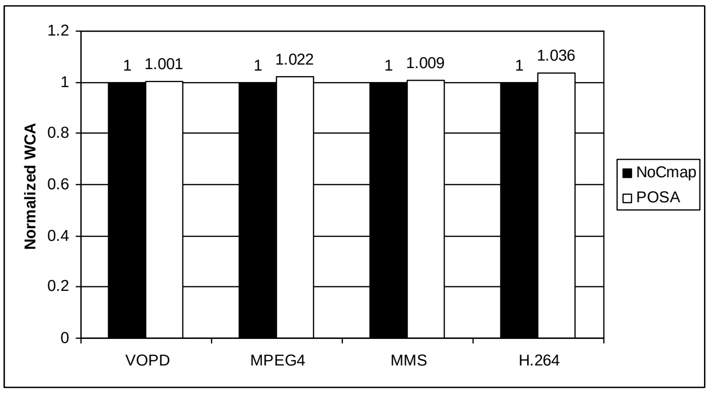

Fig. 2: Comparison of WCA

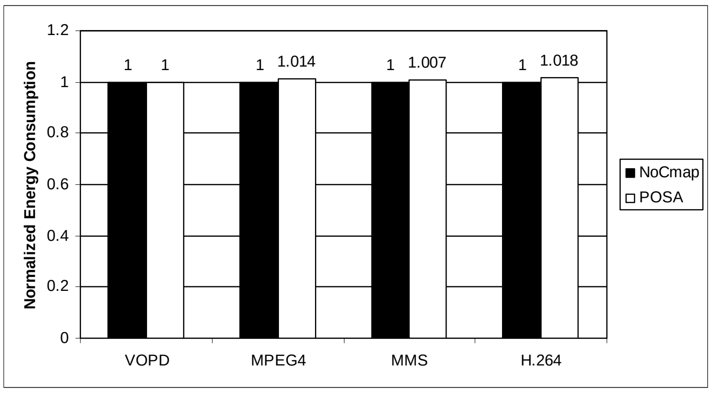

Fig. 3: Evaluation of Energy Consumption

8 Conclusions

The set of parameters applied in the SA algorithm has great impact on the runtime and the quality of the final solution. A method to systematically select the parameters of the SA algorithm for the application mapping problem is proposed in this work. The Nelder-Mead simplex method, which is used to get the minimization of a function of n variables, is applied to find the optimized parameters of the SA algorithm. With the set of optimized parameters, less evaluations are performed and the SA algorithm is accelerated. In addition, this work also points out that the parameters of SA are problem-specific. Instead of using an identical set of empirical parameters, the proposed POSA algorithm provides a way to flexibly select various number of parameters with respect to different cost functions for different kinds of mapping problems.

The experiment shows that the proposed POSA algorithm is time- and energy-efficient. The POSA algorithm only uses on average less than 1% iterations of that used in NoCmap algorithm to converge to the final optimized solution. An average speedup of 1.4 times is achieved by POSA over NoCmap. With the optimized parameters, the

SA instance in the POSA is 237 times faster than that in the NoCmap, while the optimal mapping is still found.

Acknowledgement

The authors would like to thank the Academy of Finland for the financial support for this work.

References

1. L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.
2. Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 746–749, New York, NY, USA, 2007. ACM.
3. SLD:: System Level Design Group @ CMU. Nocmap: an energy- and performance-aware mapping tool for networks-on-chip, url http://www.ece.cmu.edu/ sld/wiki/doku.php?id=shared:nocmap.
4. William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 684–689, New York, NY, USA, 2001. ACM.
5. Erno Salminen Heikki Orsila and Timo D. Hmlinen. Best practices for simulated annealing in multiprocessor task distribution problems. *Simulated Annealing, I-Tech Education and Publishing KG.*, pages 321–342, 2008.
6. Jingcao Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pages 233 – 239, jan. 2003.
7. Jingcao Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(4):551 – 562, april 2005.
8. J.A.Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
9. C.A.M. Marcon, E.I. Moreno, N.L.V. Calazans, and F.G. Moraes. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *IET Computers & Digital Techniques*, 2(6):471–482, 2008.
10. S. Murali and G. De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *Design Automation Conference, 2004. Proceedings. 41st*, pages 914 –919, july 2004.
11. H. Orsila, E. Salminen, and T.D. Hamalainen. Parameterizing simulated annealing for distributing kahn process networks on multiprocessor socs. In *System-on-Chip, 2009. SOC 2009. International Symposium on*, pages 019 –026, oct. 2009.
12. Moon-Won Park and Yeong-Dae Kim. A systematic procedure for setting parameters in simulated annealing algorithms. *Comput. Oper. Res.*, 25:207–217, March 1998.
13. Ciprian Radu and Lucian Vinţan. Optimized simulated annealing for network-on-chip application mapping. In *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS-18), Bucharest, Romania*, volume 1, pages 452–459, Bucharest, Romania, May 24-27 2011. Politehnica Press.
14. E.B. van der Tol, E.G.T. Jaspers, and R.H. Gelderblom. Mapping of h.264 decoding on a multiprocessor architecture. In *Image and Video Communications and Processing*, pages 707–718, 2003.