Design Automation for Embedded Systems September 2013

Recommendations for using Simulated Annealing in task mapping

Heikki Orsila · Erno Salminen · Timo Hämäläinen

September 2013

Abstract A Multiprocessor System-on-Chip (MPSoC) may contain hundreds of processing elements (PEs) and thousands of tasks but design productivity is lagging the evolution of HW platforms. One problem is application task mapping, which tries to find a placement of tasks onto PEs which optimizes several criteria such as application runtime, intertask communication, memory usage, energy consumption, real-time constraints, as well as area in case that PE selection or buffer sizing are combined with the mapping procedure. Among optimization algorithms for the task mapping, we focus in this paper on Simulated Annealing (SA) heuristics. We present a literature survey and 5 general recommendations for reporting heuristics that should allow disciplined comparisons and reproduction by other researchers. Most importantly, we present our findings about SA parameter selection and 7 guidelines for obtaining a good trade-off made between solution quality and algorithm's execution time. Notably, SA is compared against global optimum. Thorough experiments were performed with 2-8 PEs, 11-32 tasks, 10 graphs per system, and 1000 independent runs, totaling over 500 CPU days of computation. Results show that SA offers 4-6 orders of magnitude reduction is optimization time compared to brute force while achieving high quality solutions. In fact, the globally optimum solution was achieved with a $1.6 - 90\%$ probability when problem size is around $1e9$-$4e9$ possibilities. There is approx. 90% probability for finding a solution that is at most 18% worse than optimum.

Heikki Orsila
E-mail: heikki.orsila@iki.fi

Erno Salminen
Tampere University of Technology
E-mail: erno.salminen@tut.fi

Timo Hämäläinen
Tampere University of Technology
E-mail: timo.d.hamalainen@tut.fi

Keywords Simulated Annealing · task mapping · task graph · global optimum

1 Introduction

An efficient multiprocessor SoC (MPSoC) implementation requires automated exploration to find an efficient HW allocation, task mapping and scheduling [13]. Heterogeneous MPSoCs are needed for low power, high performance and high volume markets [37]. The central idea in MPSoCs is to increase performance and energy-efficiency. This is achieved by efficient communication between cores and keeping clock frequency low while providing enough parallelism.

Mapping means placing each application task onto some processing element (PE), as depicted in Fig. 1. Task refers here to the smallest unit of computation that can be re-located. *Scheduling* means determining execution timetable of the application components on the platform. Example Fig. 1 shows that tasks 0 and 1 are mapped to PE_0, task 2 is mapped to PE_1, and task $N-1$ is mapped to PE_{M-1}. In a strict mapping problem, state of the system is defined as a mapping of each task to a PE. The state is optimized with respect to given criteria, such as system's throughput, latency or power. The optimized criteria is defined by a cost function whose value is minimized. For example, system's performance can be optimized by setting the cost function to be its total execution time for a given program and input.

Tasks in a general mapping problem may execute arbitrary programs, deterministic or non-deterministic. In this paper, however, tasks are restricted to a subset known as Kahn Process Networks (KPNs) [15]. Each PE executes

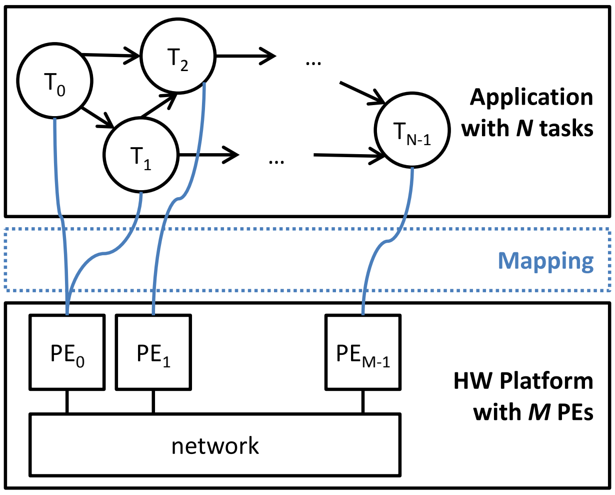


Fig. 1 Conceptual view of mapping application tasks to processing elements

a program where reads are blocking, and testing for existing readable data on a communication channel can only be done by a blocking read. This enforces a deterministic result of the computation regardless of timing and mapping of tasks.

A large design space must be pruned systematically, since the exploration of the whole design space is not feasible [13]. Fast optimization procedure is desired in order to cover reasonable design space. However, this comes with the expense of accuracy. Iterative optimization algorithms evaluate a number of application mappings for each resource allocation candidate. The application is simulated for each mapping to evaluate the *cost* of a solution. The cost may depend on multiple factors, such as cycle count, latency, energy consumption, silicon area and others.

This paper investigates the use of Simulated Annealing (SA) algorithm in task mapping. We analyze the research question of how to select SA optimization parameters for a given point in design space exploration. Section 2 presents a survey of current state-of-the-art in SA task mapping. Section 4 presents our results on SA global optimum properties, and Section 5 properties of SA acceptance functions. Section 6.1 presents recommendations for reporting SA results for disciplined comparisons and reproduction of the experiments by other researchers. Section 6.2 presents recommendations for selecting SA parameters based on related work and new analysis. Finally, we conclude the paper with high level directions on how to improve existing state-of-the-art in Section 7.

2 Related work

We limit the discussion about mapping heuristics to Simulated Annealing and mention other approaches when direct comparison has been reported in literature.

2.1 Simulated Annealing algorithm

SA is a widely used metaheuristic for complex optimization problems. Term *heuristic* means that optimality is not guaranteed but algorithm usually produces satisfactory results, whereas *meta* denotes that it can be fitted into many kinds of problems. SA is a probabilistic non-greedy algorithm that explores the search space of a problem by annealing from a high to a low *temperature.* Temperature is a historic term originating from annealing in metallurgy where material is heated and cooled to increase the size of its crystals and reduce their effects. Temperature indicates the algorithm's willingness to accept moves to a worse state.

Probabilistic behavior means that SA can find solutions of different goodness between runs. Non-greedy means that SA may accept a move into a worse state, and this allows escaping local minima. Local minimum is such a point in design space where no single move can improve the cost, but perhaps two or three consecutive moves can. The algorithm always accepts a move into a better state. Move to a worse state is accepted with a changing probability. This probability decreases along with the temperature, and thus the algorithm starts as a non-greedy algorithm and gradually becomes more and more greedy.

```
SIMULATED_ANNEALING($S, T_0$)
 1  $C \leftarrow$ COST($S$)
 2  $S_{best} \leftarrow S$
 3  $C_{best} \leftarrow C$
 4  for $i \leftarrow 0$ to $\infty$
 5  do $T \leftarrow$ TEMPERATURE($T_0, i$)
 6      $S_{new} \leftarrow$ MOVE($S, T$)
 7      $C_{new} \leftarrow$ COST($S_{new}$)
 8      $\Delta C \leftarrow C_{new} - C$
 9      if $\Delta C < 0$ or RANDOM() $<$ ACCEPT($\Delta C, T$)
10        then if $C_{new} < C_{best}$
11               then $S_{best} \leftarrow S_{new}$
12                    $C_{best} \leftarrow C_{new}$
13             $S \leftarrow S_{new}$
14             $C \leftarrow C_{new}$
15      if END-CONDITION()
16        then break
17  return $S_{best}$
```

Fig. 2 Pseudocode of the Simulated Annealing (SA) algorithm

Fig. 2 shows the SA pseudocode. The algorithm takes initial temperature T_0 and initial state S as parameters. State is modified on every iteration. **Cost** function evaluates the objective function to be optimized, e.g. by simulating the platform. This algorithm seeks to minimize the cost. **Temperature** function returns the annealing temperature as a function of T_0 and loop iteration number i. **Move** functions generates a new state from a given state. Hence, it re-locates some task(s) in our case and the associated cost (e.g. cycle count) is evaluated.

This new state can be accepted as a base for the next iteration or discarded. Move to better state is always accepted and **Accept** function calculates the probability for accepting a state change when cost function difference $\Delta C > 0$. **Random** function returns a random real number in range $[0, 1)$.

End-Condition function returns *True* iff optimization should be terminated. Parameters of the end conditions are not shown in the pseudocode. These may include various measures, such as the number of consecutive rejected moves, current and a given final temperature, current and accepted final cost. Finally the algorithm returns the best state S_{best} in terms of the **Cost** function.

Table 1 Publications where Simulated Annealing has been applied in Task mapping, Scheduling, or Communication routing.

Author	Mapping	Sched.	Com. rout.	Application type
Ali [1]	x	-	-	QoS in multisensor shipboard computer
Bollinger [4]	x	-	x	Synthetic app.
Braun [5]	x	-	-	Batch processing tasks
Coroyer [6]	x	x	-	Synthetic app.
Ercal [10]	x	-	-	Synthetic app.
Ferrandi [11]	x	x	-	C applications partitioned with OpenMP: Crypto, FFT, Image decompression, audio codec
Kim [16]	x	x	-	Synthetic app.
Koch [18]	x	x	-	DSP algorithms
Lin [22]	x x	- -	- -	Synthetic app. Synthetic app.
Nanda [24]	x	-	-	Synthetic app.
Orsila (ours) [30]	x	-	-	Synthetic app.
Ravindran [31]	x	x	-	Network processing: Routing, NAT, QoS
Wild [36]	x	-	-	Synthetic app.
Xu [38]	x	-	-	Artificial intelligence: rule-based expert system
# of pub.	14 (100%)	5 (36%)	1 (7%)	6 (43%)

2.2 SA in task mapping

Table 1 shows SA usage in 14 publications, each of which are summarized below. All publications use SA for task mapping, 5 for task scheduling, and 1 also for communication routing. None uses SA simultaneously for all purposes. There are many other methods for task mapping, such as genetic algorithms (GA), but they are outside the scope of this paper. Synthetic app. means the paper uses task graphs that are not directed toward any particular application, but artificial and meant for benchmarking the mapping algorithms. SA usually performs hundreds or thousands of iterations and therefore it is usually run off-line. Performing SA mapping at runtime is rare.

Ali [1] optimizes performance by mapping continuously executing applications for heterogeneous PEs and interconnects while preserving two quality of service constraints: maximum end-to-latency and minimum throughput. Mapping is optimized statically to increase QoS safety margin in a multisensor shipboard computer. Tasks are initially mapped by a fast greedy heuristics, after which SA further optimizes the placement. SA is compared with 9 other heuristics. SA and GA were the best heuristics in comparison. SA was slightly faster than GA, with 10% less running time.

Bollinger [4] optimizes performance by mapping a set of processes onto a multiprocessor system and assigning interprocessor communication to multiple communication links to avoid traffic conflicts. The purpose of the paper is to investigate task mapping in general.

Braun [5] optimizes performance by mapping independent (non-communicating) general purpose computing tasks onto distributed heterogeneous PEs. The goal is to execute a large set of tasks in a given time period. An example task given was analyzing data from a space probe, and send instructions back to the probe before communication black-out. SA is compared with 10 heuristics, including GA and Tabu search (TS). GA, SA and TS execution times were made approximately equal to compare effectiveness of these heuristics. GA mapped tasks were $20 - 50\%$ faster compared to SA. Tabu mapped tasks varies from being 50% slower to 5% faster compared to SA. SA was run with only one mapping per temperature level, but repeating the annealing process 8 times for two different temperature coefficients. One mapping per temperature level means insufficient number of mappings for good exploration. However, GA is better with the same number of mappings. GA gave the fastest solutions.

Coroyer [6] optimizes performance excluding communication costs by mapping and scheduling directed acyclic graphs (DAGs) to homogeneous PEs. 7 SA heuristics are compared with 27 list scheduling heuristics. SA results were the best compared to other heuristics, but SA's running time was two to four orders of magnitude higher than other heuristics. The purpose of the paper is to investigate task mapping and scheduling in general.

Ercal [10] optimizes performance by mapping DAGs onto homogeneous PEs and a network of homogeneous communication links. Load balancing constraints are maintained by adding a penalty term into the objective function. Performance is estimated with a statistical measure that depends on task mapping, communication profile of tasks and the distance of communicating tasks on the interconnect network. Simulation is not used. The model assumes communication is Programmed IO (PIO) rather than DMA. SA is compared with a proposed heuristic called Task Allocation by Recursive Mincut. SA's *best* result is better than the *mean* result in 4 out of 7 cases. Running time of SA is two orders of magnitude higher than the proposed heuristics.

Ferrandi [11] optimizes performance by mapping and scheduling a Hierarchical Task Graph (HTG) [12] onto a reconfigurable MPSoC of heterogeneous PEs. HTGs were generated from C programs parallelized with OpenMP [33]. SA is compared with Ant Colony Optimization (ACO) [9], TS and a FIFO scheduling heuristics combined with first available PE mapping. SA running time was 28% larger than ACO and 12% less than TS. FIFO scheduling happens during runtime. SA gives 11% worse results (performance of the solution) than ACO and comparable results with TS.

Kim [16] optimizes performance by mapping and scheduling independent tasks that can arrive at any time to heterogeneous PEs. Tasks have priorities and soft deadlines, both of which are used to define the performance metric for the system. Dynamic mapping is compared to static mapping where arrival times of tasks are known ahead in time. SA and GA were used as a static mapping heuristics. Several dynamic mapping heuristics were evaluated. Dynamic mapping runtime was not given, but they were very probably many orders of magnitude lower than static methods because dynamic methods are executed during the application runtime. Static heuristics gave noticeably better results than dynamic methods. SA gave 12.5% better performance than dynamic methods, and did slightly better than GA. SA runtime was only 4% of the GA runtime.

Koch [18] optimizes performance by mapping and scheduling DAGs presenting DSP algorithms to homogeneous PEs. SA is benchmarked against list scheduling heuristics. SA is found superior against other heuristics, such as *Dynamic Level Scheduling* (DLS) [34]. SA does better when the proportion of communication time increases over the computation time, and the number of PEs is low.

Lin [22] optimizes performance while satisfying real-time and memory constraints by mapping general purpose synthetic tasks to heterogeneous PEs. SA reaches a global optimum with 12 node graphs.

Nanda [24] optimizes performance by mapping synthetic random DAGs to homogeneous PEs on hierarchical buses. The performance measure to optimize is an estimate of expected communication costs and loss of parallelism with respect to critical path (CP) on each PE. Schedule is obtained with list scheduling heuristics. Two SA methods are presented where the second one is faster in runtime but gives worse solutions. The two algorithms reach within 2.7% and 2.9% of the global optimum for 11 node graphs.

Ravindran [31] optimizes performance by mapping and scheduling DAGs with resource constraints on heterogeneous PEs. SA is compared with DLS and a decomposition based constraint programming approach (DA). DLS is the computationally most efficient approach, but it loses to SA and DA in solution quality. DLS runs in less than a second, while DA takes up to 300 seconds and SA takes up to 5 seconds. DA is an exact method based on constraint programming that wins SA in solution quality in most cases, but is found to be most viable for larger graphs where constraint programming fails due to complexity of the problem space.

Wild [36] optimizes performance by mapping and scheduling DAGs to heterogeneous PEs. PEs are processors and accelerators. SA is compared with TS, FAST [20] [21] and a proposed Reference Constructive Algorithm (ReCA). TS gives $6 - 13\%$, FAST $4 - 7\%$, and ReCA $1 - 6\%$ better application execution time than SA. FAST is the fastest optimization algorithm. FAST is 3 times as fast than TS for 100 node graphs and 7 PEs, and 35 times as fast than SA. TS is 10 as fast as SA.

Xu [38] optimizes performance of a rule-based expert system by mapping dependent production rules (tasks) onto homogeneous PEs. A global optimum is solved for the estimate by using linear programming (LP) [23]. SA is compared with the global optimum. SA reaches within 2% of the global optimum in 1% of the optimization time compared to LP. The cost function is a linear estimate of the real cost. In this sense the global optimum is not real.

In summary, SA performs rather well on average. The differences regarding solution quality are usually $1 - 10\%$ whereas in algorithm runtimes vary greatly, e.g. from few percents to even $100x - 1\ 000x$. Heuristics have been

Table 2 Utilized move heuristics, acceptance functions, and annealing schedule in Simulated Annealing. Geometric annealing schedules (G) have a temperature scaling co-efficient "q". Adaptive initial temperature "T_0" and Stop condition are also marked. "L" is the number of iterations per temperature level, where N is the number of tasks and M is the number of PEs.

Author	Move function	Acc. func.	Ann. sched.	T scale coeff. (q)	Adaptive T_0	Adaptive stop	#Iter per T level (L)
Ali [1]	ST, SW1	E	G	0.99	-	-	1
Bollinger [4]	MBOL	?	B	N/A	x	?	$N(N-1)/2$
Braun [5]	ST	IE	G	0.8, 0.9	x	-	1
Coroyer [6]	ST, P1	E	G, F	0.95	x	x	$N(M-1)$
Ercal [10]	ST	E	G	0.95	x	-	$5N(M-1)$
Ferrandi [11]	ST, P4	E	G	0.99	-	-	LFE
Kim [16]	ST, SW1, P3	E	G	0.99	-	-	1
Koch [18]	Koch	E	K	N/A	x	x	N
Lin [22]	MLI	E	G	0.8	x	x	LLI
Nanda [24]	ST, SW2	E	G	0.9	-	-	5000
Orsila [30]	ST	NIE	G	0.95	x	x	$N(M-1)$
Ravindran [31]	H1	E	K	N/A	-	x	?
Wild [36]	ST, EM	?	G	?	?	x	?
Xu [38]	SW1	E	G	0.5	x	x	N
Diff. choices	11	4	4	5	≥2	≥2	≥8
Most common	ST (64%)	E (71%)	G (79%)	0.95, 0.99 (21%,21%)	x (57%)	x (50%)	adaptive (57%)

commonly evaluated for systems with 6-8 PEs and the task graph sizes vary greatly, from 12-4000 tasks. So far, most of the comparisons were between heuristics and unfortunately not against global optimum (11 to 14 PE cases). Global optimum is the exact comparison point and should be sought when feasible. Even if near-optimal results in a small test cases do not guarantee performance in larger problem, at least they show the performance trend (e.g. linear vs. quadratic degradation). Nevertheless, extrapolating the performance is somewhat dubious.

3 SA parameters

Table 2 shows parameter choices for the publications presented in Section 2.2. Move and acceptance functions, and annealing schedule, and the number of iterations per temperature level were investigated, where N is the number of tasks and M is the number of PEs. "?" indicates the information is not available. "N/A" indicates that the value does not apply to a particular publication. Detailed explanation of parameters is presented in Sections 3.1, 3.2 and 3.3.

3.1 Move functions

Table 2 shows 11 different move functions applied in the publications. There are two kinds of functions. Agnostic ones do not consider application or platform structure at all, whereas the others may weigh some tasks more than rest.

Single Task (*ST*) move takes one random task and moves it to a random PE. It is the most common move heuristics, 9 out of 14 publications use it. It is not known how many publications exclude the current PE from solutions so that always a different PE is selected by randomization. Excluding the current PE is useful because evaluating the same mapping again on consecutive iterations is counterproductive.

EM [36] is a variation of ST that limits task randomization to nodes that have an effect on critical path length in a DAG. The move heuristics is evaluated with SA and TS. SA solutions are improved by $2 - 6\%$, but TS solutions are not improved. Using EM multiplies the SA optimization time by $50 - 100x$, which indicates it is dubious by efficiency standards. However, using EM for TS approximately halves the optimization time!

Swap 1 (*SW1*) move is used in 3 publications. It chooses 2 random tasks and swaps their PE assignments. These tasks should preferably be mapped on different PEs. *MBOL* is a variant of *SW1* where task randomization is altered with respect to annealing temperature. At low temperatures, only the tasks that are close in system architecture are considered for swapping. At high temperatures more distant tasks are considered.

Priority move 4 (*P4*) is a scheduling move that swaps the priorities of two random tasks. This can be viewed as swapping positions of two random tasks in a task permutation list that defines the relative priorities of tasks. *P1* is a variant of P4 that considers only tasks that are located on the same PE. A random PE is selected at first, and then priorities of two random tasks on that PE are swapped. *P3* scheduling move selects a random task, and moves it to a random position in a task permutation list.

Hybrid 1 (*H1*) is a combination of both task assignment and scheduling simultaneously. First ST move is applied, and then *P3* is applied to the same task to set a random position on a permutation list of the target PE. *Koch* [18] is a variant of H1 that preserves precedence constraints of the moved task in selecting the random position in the permutation of the target PE. That is, schedulable order is preserved in the permutation list.

MLI is a combination of ST and SW1 that tries three mapping alterations. First, tries ST greedily. The move heuristics terminates if the ST move improves the solution, otherwise the move is rejected. Then SW1 is tried with SA acceptance criterion. The move heuristics terminates if the acceptance criterion is satisfied. Otherwise, ST is tried again with SA acceptance criterion.

ST is the most favored mapping move. Heuristics based on swapping or moving priorities in the task priority list are the most favored scheduling moves. The choice and impact of mapping and scheduling moves have not been studied thoroughly.

3.2 Acceptance functions

Table 2 shows that 4 different acceptance functions have been used in publications. Acceptance function takes the change in cost ΔC and temperature T as parameters. There are 3 relevant cases to decide whether to accept or reject a move.

1. $\Delta C < 0$ is trivially accepted since the cost decreases.
2. $\Delta C = 0$ is probabilistically often accepted. The probability is usually 0.5 or 1.0.
3. $\Delta C > 0$ is accepted with a probability that decreases when T decreases, or when ΔC grows.

The most common choice is exponential acceptor function (denoted with E in the table)

$$Accept(\Delta C, T) = exp(\frac{-\Delta C}{T}). \tag{1}$$

The probabilities fall into range $[0, 1.0]$ which means that quite many worsening moves are accepted. Unfortunately, the behavior depends heavily on the ratio of cost (depends on system) and temperature (derived by SA developer).

In order to avoid manual parameter tuning, Orsila [30] [27] uses the normalized inverse exponential function (*NIE*)

$$Accept(\Delta C, T) = \frac{1}{1 + exp(\frac{\Delta C}{0.5\ C_0\ T})}. \tag{2}$$

First of all, temperature level T is in normalized range $(0, 1]$ regardless of the system. Moreover, ΔC is normalized with respect to initial cost C_0. Second, the denominator part is always ≥ 2, and hence probabilities are in range [0, 0.5].

The normalized exponential (*NE*) acceptor

$$Accept(\Delta C, T) = exp(\frac{-\Delta C}{0.5\ C_0\ T}) \tag{3}$$

uses similar normalizations as NIE acceptor (equation 2). The difference is that $Accept(0, T) = 1.0$ for NE (just like E) and 0.5 for NIE. Fig. 3 illustrates the differences. X-axis shows the temperature. It decreases during the process, and hence optimization proceeds from right to left. Y-axis shows the acceptance probability. The lines show the acceptance probabilities for moves that are 5% and 30% worse than current one. NE was found to be slightly better than NIE in [25] (see Section 5.1).

Braun [5] uses an inverse exponential function (*IE*)

$$Accept(\Delta C, T) = \frac{1}{1 + exp(\frac{\Delta C}{T})}. \tag{4}$$

Temperature normalization is not used, but the same effect is achieved by setting the initial temperature properly: $T_0 = C_0$.

Using an exponential acceptor and dynamic initial temperature is the most common choice.

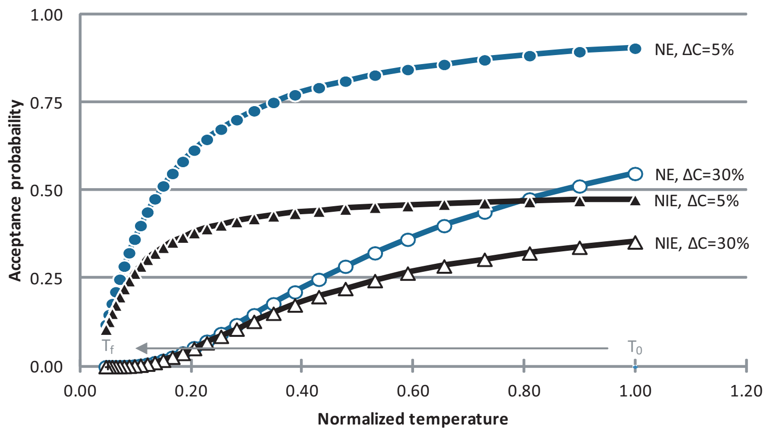


Fig. 3 Probabilities of Normalized Exponential (NE) and Normalized Inverse Exponential (NIE) acceptance function. Small worsening moves are accepted with higher probability. Moreover, NE accepts nearly twice as many moves as NIE at high temperatures (beginning of optimization).

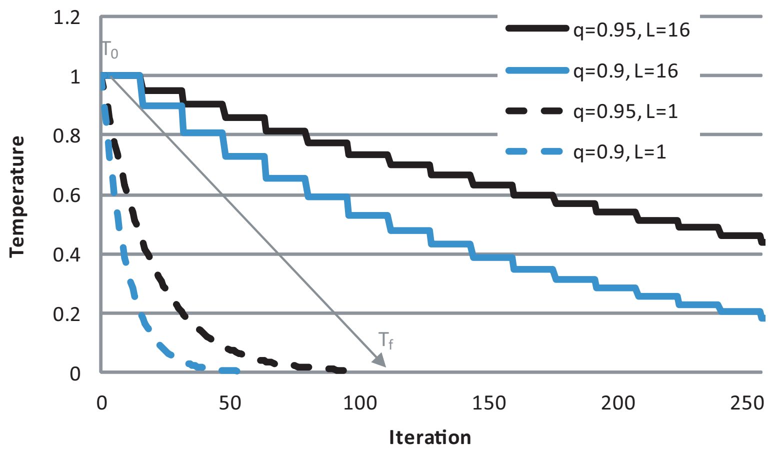


Fig. 4 Examples of geometric annealing schedule with two scaling coefficients $q = 0.95$ and $q = 0.9$. Moreover, two iteration counts are shown $L = 1$ and $L = 16$. Too small q and L will terminate SA quickly and likely cause sub-optimal results.

3.3 Annealing schedule

Table 2 shows that 4 different annealing schedules have been used in publications. The annealing schedule is a trade-off between solution quality and optimization time. The annealing schedule defines the temperature levels and the number of iterations for each temperature level. Optimization starts at the initial temperature T_0 and ends at final temperature T_f. These may not be constants, in which case initial temperature selection and/or stopping criteria are adaptive. Stopping criteria defines when optimization ends.

Geometric temperature scale (G) is the most common annealing schedule (11 out of 14). Temperature T is multiplied by a factor $q \in (0, 1)$ to get the temperature for the next level, that is,

$$T_{next} = qT. \tag{5}$$

Usually several iterations are spent on a given temperature level before switching to the next level. Scaling factor q is most often 0.95 or 0.99 in literature. Some works use $q = 0.9$ which was also used by Kirkpatrick et al. who presented SA [17]. They used SA for partitioning circuits to two chips. This is analogous to mapping tasks onto two PEs. Fig. 4 shows temperature as a function of iteration count for 4 different schedules. Naturally, temperature decreases faster with small q and optimization will terminate quicker. Moreover, iteration count per temperature level has a major impact (see subsection 3.5).

Bollinger's schedule [4], denoted B, consider the size of improvement during scaling. They computes a ratio R of minimum cost to average cost on a temperature level, and then applies a geometric temperature multiplier

$$T_{next} = min(R, q_L)T, \tag{6}$$

where q_L is an experimentally chosen lower bound for the temperature multiplier. q_L varied in range $[0.9, 0.98]$.

Koch [18] and Ravindran [31], denoted K, use a dynamic q factor that depends on the statistical variance of cost at each temperature level. The temperature is calculated with

$$T_{next} = \frac{T}{1 + \frac{T \; ln(1+\delta)}{3 \; \sigma_T}} \tag{7}$$

where δ is an experimental parameter used to control the temperature step size and σ_T is the standard deviation of cost function on temperature level T. Higher δ or lower σ_T means a more rapid decrease in temperature. Koch suggests δ in range $[0.25, 0.50]$.

Coroyer [6] experimented with a fractional temperature scale (F). Temperature T is calculated as $T_i = \frac{T_0}{i}$ where T_0 is the initial temperature and i is the iteration number starting from 1. Fractional temperature was found to be worse than geometric.

3.4 Adaptivity in start and stop conditions

The choice of initial temperature T_0 can be chosen experimentally or methodically. Ten works present methods to set T_0 based on given problem space characteristics and desired acceptance criteria for moves [26] [29] [30] [5] [4] [6] [10] [18] [22] [38]. For example, T_0 can be set purely based on simulation so that T_0 is raised high enough so that average move acceptance probability p is high enough for aggressive statistical sampling of the problem space, e.g. $P \sim 0.9$ [6] [18]. In contrast, our method determines T_0 from the task graph and the system architecture sizes [26] [29] [30].

There are several common stopping criteria. For example, optimization ends when a given T_f has been reached [6] [18], a given number of consecutive rejections happen, a given number of consecutive moves has not improved the solution, a given number of consecutive solutions are identical [6], or a solution (cost) with given quality is reached [18]. These criteria can also be combined. Constants associated with these criteria are often decided experimentally, but adaptive solutions exist too. We use adaptive T_f that is computed from problem space similarly to T_0 [26] [29] [30].

3.5 Iteration per temperature level

The number of iterations per temperature level L is defined as an experimental constant or a function of the problem space. It affects the schedule and hence also acceptance function notably, as highlighted in Fig. 4. No single method is clearly more common than others, but functions are more common than constants (8 vs. 4 papers). A function of the problem space (based on number of tasks and PEs; and cost) is more widely applicable, but a constant can be more finely tuned to a specific problem. Parameter L is often dependent on task count N and PE count M. Ercal [10] proposes L that is proportional to $N(M-1)$, the number of neighboring solutions in the mapping space.

Lin [22] uses an adaptive number of iterations per temperature level (LLI). Their approach starts with $L_0 = N(N+M)$ at T_0. The number of iterations for the next temperature level is calculated by $L_{next} = min(1.1L, N(N+M)^2)$. However, on each temperature level they compute an extra $X - L$ iterations iff $X > L$, where $X = \frac{1}{exp((C_{min}-C_{max})/T)}$ and C_{min} and C_{max} are the minimum and maximum costs on the temperature level T. They compute initial temperature as $T_0 = a + b$, where a is the maximum execution cost of any task at any PE, and b is the maximum communication cost between any two tasks. Then they compute T_0 average and standard deviation of cost at T_0 by sampling a number of moves. The temperature is doubled (once) if the minimum and maximum cost is not within two standard deviations from the average.

Ferrandi [11] has adaptive number of iterations per temperature level (LFE). Temperature level is switched on the first accepted move on the temperature level. However, there can be arbitrarily many rejected moves.

4 On global optimum results

This section analyzes how many mapping iterations are needed to reach a given solution quality. SA convergence rate and the number of mapping iterations

Table 3 The optimization cases for brute force optimization. For each case we find global optimum for 10 graphs generated with KPN generator. G means 10^9.

PEs (M)	Nodes (N)	Brute force iterations per graph	Mappings per second	Brute force runtime for 10 graphs [CPU days]
2	32	$2^{31} \sim 2.1G$	3 725	67
3	21	$3^{20} \sim 3.5G$	3 583	113
4	17	$4^{16} \sim 4.3G$	2 745	181
6	13	$6^{12} \sim 2.2G$	2 523	100
8	11	$8^{10} \sim 1.1G$	2 193	57

is compared with respect to global optimum solutions. The effect of SA parameters on convergence rate and quality of solution is analyzed. Specifically, the effect of L value and acceptance function is analyzed with respect to the number of mapping iterations and solution quality. We present convergence results with reference to global optimum solutions.

One difficulty in experiments with heuristics is the choice of appropriate reference point. Unfortunately, a global optimum is hard to find for large task graphs since the mapping problem is in NP-complexity class. Therefore, we present our thorough experiment where SA is compared to global optimum that was found with exhaustive search. Effectiveness of heuristics may decrease rapidly as the number of task nodes grows. One task can be placed on any of the M PEs and hence the graphs requires an N items long vector to store the full mapping. The total number of mappings X for N tasks and M PEs is $X = M^N$. The number of mappings grows exponentially with respect to tasks, and polynomially with respect to PEs. Adding PEs is preferable to adding nodes from this perspective.

4.1 Experiment setup

Our SA with automatic temperature (SA+AT) algorithm in [30] is further analyzed with respect to global optimum convergence. The automatic temperature algorithm defines the start temperature T_0, final temperature T_f, and iteration per temperature level L so that optimality and SA runtime are balanced. SA+AT is applied to problem space of variable number of PEs and nodes with Kahn Process Networks (KPNs). Specifically, we brute force solve the problem for cases shown in Table 3, and then compare SA+AT heuristics with the *brute force global optimum* results.

More than 2 PEs would be too costly to compute in brute force for 32 or more nodes. Already now, the experiments together required over 500 days of CPU time. Therefore, we limited the number of nodes for each case as follows. The number of nodes for each case is picked so that the number of brute force iterations to find global optimum is the same within factor 0.5 to 2. As one task mapping was always fixed to one PE, the brute force complexity of the problem is $X = 2^{31} = 2\,147\,483\,648 \sim 2.1 \times 10^9$ mappings for each graph. Note, fixing one node is a valid choice because PEs are homogeneous and connected with a single shared bus in our cases. Then, given M PEs, we selected number of nodes N so that following equation roughly holds: $M^{N-1} = X$.

In Table 3, mappings per second determines the speed at which brute force optimization can make progress. The average for 2 and 8 PE cases varies by a factor of $1.7x$. Optimization time in brute force and SA is dominated by the time to evaluate a mapping. The number of CPU days consumed in the brute force experiment is also presented. Note that since 10 graphs were optimized for each case, one tenth of this value presents brute force optimization time for a single graph.

Ten synthetic graphs were generated with kpn-generator [19] for each system size. For 2 PE graphs target distribution parameter was set to 10%, in other cases 30%. Target distribution defines the relative number of tasks that can act as a target for each task. For example, 10% target distribution for a 32 node graph means each task can communicate with at most 3 other tasks. All graphs were cyclic graphs (default) and b-model parameter for both communication size and computation time burstiness was set at 0.7 (default).

SA+AT was 1000 times independently for each of the 10 graphs for each system size. We measured the proportion SA runs that come within p percent of global optimum cost. Here, we use execution time t as the cost. We measured the number of SA+AT runs that got execution time $t \leq (1+p)t_o$, where p is the execution time overhead compared to global optimum execution time t_o. Although the graphs generated by kpn-gen are cyclic, each tasks is activated a certain fixed number of times. Execution of the graph is complete when there are no more activations and that instant defines the execution time.

2 PE graphs for the experiment available at DCS task mapper web page [7] under the section "Experimental data". The experiment data for 3, 4, 6 and 8 PEs is available at [8]. The first case with 2 PEs used NIE acceptor, but once NE was found better it was used for cases with $3-8$ PEs. A reference C implementation of the SA+AT algorithm is also available at [32]. The experiment can be repeated with a GNU/Linux system by using DCS task mapper and jobqueue [14].

4.2 Example convergence with one graph

Let us first consider the difficulty of the mapping. Fig. 5 which shows the task graph execution time for one graph plotted against the number of mappings. All possible mapping combinations for 32 nodes and 2 PEs were computed with brute force search. One node mapping was fixed and the rest 31 were which resulted in 2^{31} mappings. The initial execution time is 875 μs (all tasks on a single PE), mean 1033 μs and standard deviation 113 μs. It is interesting to note that there is exactly one optimum mapping which results in 535 μs. That is nearly $2x$ as fast as average and about $3.2x$ as fast as the worst case. Hence, mapping has a large impact on performance but optimal case is extremely unlikely to be found with random choice. It is like a needle in a haystack.

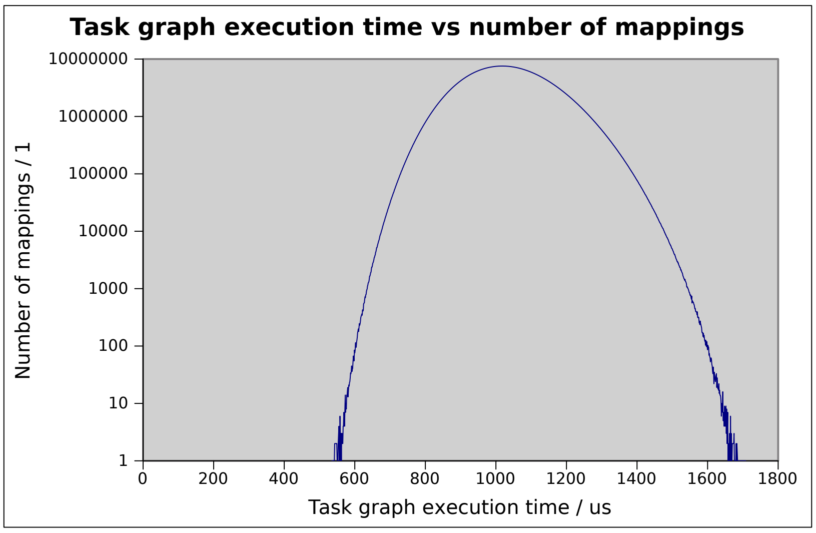


Fig. 5 Task graph execution time for one graph plotted against the number of mappings. All possible mapping combinations for 32 nodes and 2 PEs were computed with brute force search. One node mapping was fixed. 31 node mappings were varied resulting into $2^{31} \approx 2.1 \times 10^9$ mappings. The initial execution time is 875 μs, mean 1033 μs and standard deviation 113 μs. There is only one mapping that reaches the global optimum 535 μs.

Table 4 The number of mappings within p% global optimum computed with a brute force search. Furthermore, proportion of SA runs that converged to that are also shown. Note that there are only very few good mappings out of 2 milliard. Higher SA run proportion is better.

$p = \frac{t}{t_o} - 1$	Number of mappings	SA run proportion
+0%	1	2.1%
+1%	1	2.1%
+2%	4	5.8%
+3%	8	11.2%
+4%	16	17.1%
+5%	30	19.8%
+6%	45	21.2%
+7%	74	28.2%
+8%	127	34.7%
+9%	226	40.5%
+10%	355	49.0%

Table 4 shows how many SA+AT runs got execution time overhead p or less compared to global optimum execution time t_o. It also shows the associated number of mappings within that range computed from the brute force search.

There is only one mapping that is the global optimum: 535 μs but SA reaches it in 2.1% of the runs which is a very good result considering the difficulty of this mapping problem. Repeating SA yields the global optimum in approximately 10^5 iterations. Purely random mapping (RM) would find

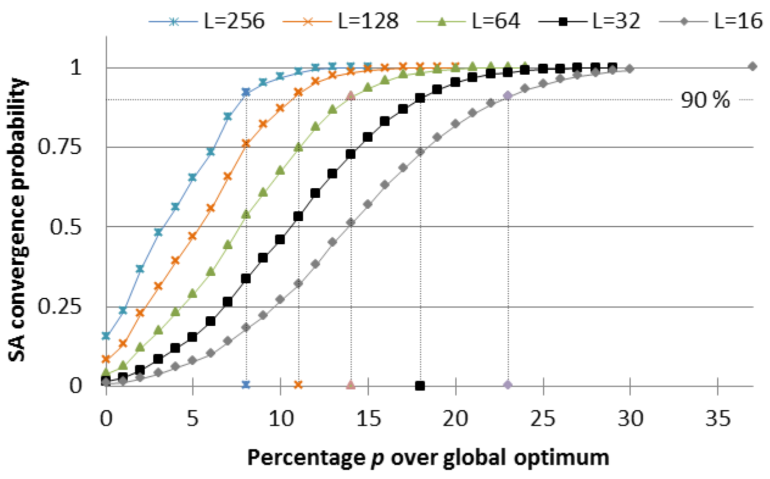


Fig. 6 SA+AT convergence with respect to global optimum with 2 PEs and 32 task nodes, as in Table 5. Curves show proportion of SA+AT runs that converged within p from global optimum for a given value of L. Lower p value on X-axis is better. Higher probability on Y-axis is better. SA+AT chooses $L = 32$.

global optimum in 10^9 iterations in approximately half the runs. RM only converges within 13% of the global optimum in a million iterations.

Interestingly there is only one mapping that is the worst solution: 1708 μs. Reversing the cost difference comparison in SA yields the worst solution in 0.9% of the runs. The SA algorithm can therefore also find the global maximum.

4.3 Convergence results for 10 graphs

Table 5 shows the execution time overheads for 2 PEs and 32 tasks considering all the 10 task graphs. The values are well aligned with Table 4 and also shown in Fig. 6. The first row, $p = 0\%$, means the global optimum (shortest execution time). $p = 10\%$ means execution time no more than 10% over the global optimum. The last two rows show mean and median values for the number of mappings tried in a single SA+AT run. SA uses only small fraction of the brute force iterations, but does not always converge to a global optimum. However, the optimum result is obtained in $0.6 - 15.2\%$ of cases depending on L. However, slightly looser bound $p = +5\%$ is already reached with $7.6 - 64.9\%$ of runs which is very good. For each L, the 90% probability level is marked in boldface in the table and dotted vertical lines in Fig. 6 show corresponding the overhead p.

Determining the cost is the most time consuming operation, as it usually requires some sort of simulation. Therefore, the number of mapping iterations should minimized to obtain reasonable runtime for the heuristic. The runtime

Table 5 SA+AT convergence with respect to global optimum for 2 PEs and 32 nodes. Values in table show proportion of SA+AT runs that converged within p from global optimum. The higher value is better. Experiment varied parameter $L = 16 - 256$ and automatic parameter selection method of SA+AT chooses $L = 32$. The 90% level is marked in boldface on each column.

	Proportion of runs within limit p				
	L value				
$p = \frac{t}{t_o} - 1$	16	**32**	64	128	256
+0%	0.006	0.016	0.038	0.080	0.152
+1%	0.010	0.026	0.061	0.129	0.232
+2%	0.023	0.051	0.117	0.224	0.365
+3%	0.039	0.084	0.173	0.311	0.477
+4%	0.057	0.119	0.230	0.391	0.558
+5%	0.076	0.154	0.287	0.468	0.649
+6%	0.101	0.201	0.355	0.553	0.732
+7%	0.137	0.264	0.439	0.653	0.834
+8%	0.178	0.337	0.536	0.757	**0.919**
+9%	0.220	0.400	0.606	0.819	0.949
+10%	0.267	0.461	0.675	0.870	0.969
+11%	0.319	0.531	0.747	**0.917**	0.985
+12%	0.378	0.605	0.812	0.952	0.994
+13%	0.449	0.664	0.864	0.972	0.998
+14%	0.509	0.726	**0.906**	0.983	0.999
+15%	0.568	0.780	0.935	0.992	1.000
+16%	0.627	0.831	0.958	0.996	
+17%	0.681	0.868	0.974	0.998	
+18%	0.731	**0.903**	0.984	0.999	
+19%	0.778	0.930	0.990	0.999	
+20%	0.820	0.953	0.994	1.000	
+21%	0.853	0.968	0.997		
+22%	0.884	0.979	0.998		
+23%	**0.908**	0.985	0.999		
+24%	0.931	0.991	1.000		
+25%	0.947	0.995			
+26%	0.961	0.996			
+27%	0.971	0.998			
+28%	0.980	0.999			
+29%	0.987	1.000			
+30%	0.992				
...	...				
+37%	1.000				
mean mappings	840	1 704	3 516	7 588	19 353
median mappings	836	1 683	3 437	7 428	18 217

of the heuristic is often negligible, and hence it pays off to design a more complex algorithms which cleverly reduces the iteration count.

Table 6 shows the expected number of mappings needed to obtain a result within p percent of global optimum by repeatedly running SA+AT. They are derived from convergence rates. For example take a look at cell $p = 5\%, L = 16$ in Table 5. It shows that roughly 7.6% of the runs reached that performance. Dividing the mean number of mappings with that, gives $\frac{840}{0.0757} = 11\ 099$, which

Table 6 Approximate expected number of mappings for SA+AT to reach performance within p percent from the global optimum with 2 PEs and 32 nodes. Values are derived from Table 5. SA+AT chooses $L = 32$. The best values (smallest) are in boldface on each row.

	Estimated number of mappings				
	L value				
p	16	**32**	64	128	256
+0%	137 741	108 507	**92 766**	94 966	127 744
+1%	84 022	65 021	**57 921**	58 865	83 347
+2%	36 531	33 208	**29 947**	33 919	53 023
+3%	21 767	20 402	**20 334**	24 429	40 556
+4%	14 793	**14 340**	15 300	19 406	34 683
+5%	11 099	**11 084**	12 242	16 206	29 820
+6%	**8 311**	8 471	9 907	13 721	26 439
+7%	**6 155**	6 458	8 001	11 627	23 208
+8%	**4 723**	5 060	6 561	10 027	21 059
+9%	**3 812**	4 255	5 804	9 264	20 402
+10%	**3 150**	3 695	5 212	8 724	19 970
+11%	**2 635**	3 208	4 708	8 276	19 654
+12%	**2 221**	2 817	4 328	7 972	19 472
+13%	**1 873**	2 564	4 069	7 810	19 394
+14%	**1 651**	2 348	3 881	7 717	19 367
+15%	**1 480**	2 183	3 761	7 647	19 357
+16%	**1 340**	2 051	3 670	7 618	
+17%	**1 234**	1 963	3 609	7 604	
+18%	**1 149**	1 886	3 572	7 595	
+19%	**1 080**	1 832	3 550	7 593	
+20%	**1 025**	1 787	3 537	7 589	
+21%	**985**	1 760	3 526		
+22%	**950**	1 741	3 524		
+23%	**926**	1 730	3 520		
+24%	**903**	1 719	3 518		
+25%	**888**	1 713			
+26%	**874**	1 710			
+27%	**865**	1 706			
+28%	**857**	1 705			
+29%	**851**	1 704			
+30%	**847**				
+31%	**845**				
...	...				
+37%	**841**				
mean	840	1 704	3 516	7 588	19 353
median	836	1 683	3 437	7 428	18 217

is shown in Table 6. Automatically selected L value works rather well as it has the smallest or second smallest iteration count for each values of p.

Note that convergence is not a guarantee but probabilistic, which means the actual global optimum might never be reached. Brute force for 2 PEs uses $2.0 \times 10^4 - 2.3 \times 10^4$ times the iterations compared to SA, for $L = 32$ and $L = 64$ respectively. Hence, SA offers 4 orders of improvement over brute force. Moreover, results within 3% of global optimum are reached with 5 orders of

Table 7 Global optimum convergence rate varies between the tested 10 32-node graphs and L values. Column minimum denotes the hardest graph and maximum the easiest one. Value 0% in Min column means there was a graph for which global optimum was not found in 1000 SA+AT runs. Note, the Mean column has same values as the $p = 0\%$ row in Table 5.

L	Min (%)	Mean (%)	Median (%)	Max (%)
16	0.0	0.6	0.4	1.6
32	0.1	1.6	0.9	4.7
64	0.3	3.8	2.9	8.6
128	0.3	8.0	6.9	15.7
256	0.4	15.2	15.6	27.8

magnitude improvement in iterations by using the SA+AT. Results within 16% are reached with 6 orders of magnitude improvement.

4.4 Differences between graphs

There are of course differences between 10 tested graphs. Table 7 shows the convergence rate variance between graphs for L values $16 - 256$ for 2-PE case. Minimum and maximum convergence rates vary by a factor of $50x$, approximately. Doubling L approximately doubles the mean convergence rate. For $L \geq 32$ global optimum was found for each graph when SA was run 1000 times. However, L should be scaled according to the problem. Otherwise, a small L obtains poor results with large problems, or a large value wastes time with simple problems.

With $L = 256$ the hardest 32-node graph converged with probability 0.4%, but the same graph converged within 1% with 4.4% probability, within 2% with 8.8% probability, and within 3% with 22.3% probability. Sacrificing just 3% from optimum more than doubles the convergence rate on average, as shown by Table 5.

The easiest graph converged to optimum with 27.8% probability, but within 3% with just little more likely, with 32.6% probability. This indicates that it does not have many solutions near the global optimum, but hitting the global optimum is rather probable (27.8%). This is opposite to the hardest graph where hitting the global optimum is hard (probability 0.4%), but there are surprisingly many solutions within 3% that are found with 22.3% probability. Such differences necessitate that evaluation of a heuristic uses many task graphs.

4.5 Applicability of SA on larger problems

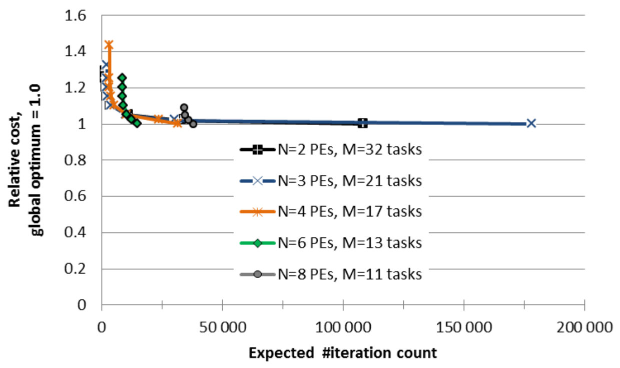
SA performed well on a 2-PE case and the experiments were repeated larger PE counts. As mentioned, task count was reduced since otherwise brute force solution would be infeasible. Tables 8 and 9 shows the results for 8 PEs and 11 tasks. The result tables for $N = 3, 4, 6$ PEs are found in Appendix.

Table 8 Proportion of SA+AT runs that converged within p from global optimum for 8 PEs and 11 nodes. A higher value is better. SA+AT chooses $L = 77$.

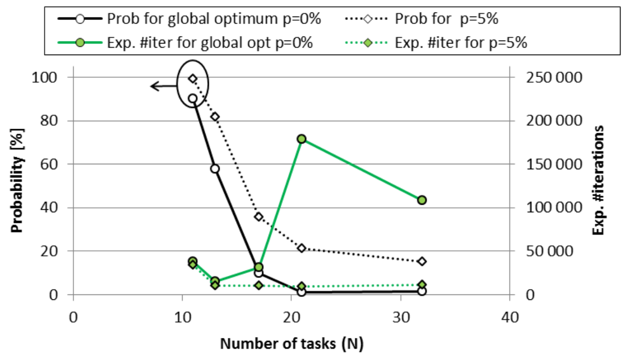
	Proportion of runs within limit p						
	L value						
$p = \frac{t}{t_o} - 1$	16	32	64	**77**	128	256	512
+0%	0.374	0.615	0.698	0.715	0.768	0.837	0.891
+1%	0.496	0.722	0.788	0.803	0.854	**0.918**	**0.960**
+2%	0.567	0.777	0.849	0.862	**0.904**	0.946	0.981
+3%	0.671	0.832	0.877	0.882	0.914	0.950	0.981
+4%	0.726	0.854	**0.900**	**0.904**	0.932	0.956	0.983
+5%	0.769	0.865	0.904	0.907	0.934	0.956	0.983
+6%	0.830	**0.909**	0.929	0.929	0.942	0.957	0.983
+7%	0.856	0.911	0.929	0.930	0.942	0.957	0.983
+8%	**0.947**	0.992	0.998	0.998	0.999	1.000	1.000
+9%	0.962	0.999	1.000	1.000	1.000		
+10%	0.974	1.000					
+11%	0.986						
...	...						
+24%	1.000						
mean mappings	914	4 730	27 227	34 502	58 206	116 427	232 856
median mappings	885	3 330	28 932	34 961	58 116	116 336	232 689

Table 9 Approximate expected number of mappings for SA+AT with 8 PEs and 11 nodes. SA+AT chooses $L = 77$.

	Estimated number of mappings						
	L value						
p	16	32	64	**77**	128	256	512
+0%	**2 442**	7 690	39 013	48 242	75 780	139 134	261 313
+1%	**1 843**	6 549	34 570	42 983	68 133	126 869	242 559
+2%	**1 613**	6 085	32 055	40 031	64 388	123 047	237 487
+3%	**1 363**	5 685	31 056	39 136	63 697	122 594	237 269
+4%	**1 260**	5 536	30 252	38 154	62 433	121 760	236 931
+5%	**1 189**	5 468	30 115	38 028	62 346	121 760	236 931
+6%	**1 102**	5 206	29 314	37 131	61 771	121 608	236 907
+7%	**1 068**	5 192	29 295	37 115	61 771	121 608	236 907
+8%	**966**	4 768	27 293	34 558	58 247	116 427	232 856
+9%	**951**	4 733	27 227	34 502	58 206		
+10%	**939**	4 732					
+11%	**927**	4 732					
...	...						
+17%	**917**	4 730					
+18%	**917**						
...	...						
+29%	**914**						
mean	914	4 730	27 227	34 502	58 206	116 427	232 856
median	885	3 330	28 932	34 961	58 116	116 336	232 689



(a) Cost as a function of iterations.



(b) Impact of task on solution quality and iterations count

Fig. 7 Convergence rates for M=2 – 8 PEs with automatically selected L values $(32 - 77)$.

Convergence is faster with 8 PEs than 2 PEs. First of all, the problem size is smaller $(1.1G < 2.1G)$ but that is not the only reason. Task count is also smaller $(N = 11 < N = 32)$ and this has bigger impact, as evident from Tables 13-18. Note that once convergence proportion is rounded to 1.000 no more values are shown, but iterations counts do still increase very little (e.g. $L = 32, p = 11 - 17\%$).

Fig. 7(a) we notice how quickly SA converges towards global optimum with all tested PE counts. Fig. 7(b) plots the probability of reaching solutions $p = 0\%$ (optimum) and $p = 5\%$ as a function of task count (left Y axis). Note that problem sizes are roughly equal from $1.1G$ to $4.3G$ solutions which allows estimating the impact of task count. Moreover, the expected iteration counts are shown on the right Y axis. Iteration ocunt for optimum solution increases with task count and has a peculiar spike at 20 tasks (3 PEs). In contrary, iterations count changes only modestly for for $p = 5\%$.

Table 10 Relative optimization time for reaching within p percent of global optimum. Time to reach the global optimum is scaled 1.0 time units for each PE count.

p	2 PEs	3 PEs	4 PEs	6 PEs	8 PEs
0	1.000	1.000	1.000	1.000	1.000
5	0.102	0.056	0.347	0.707	0.788
10	0.034	0.023	0.160	0.600	0.715
15	0.020	0.015	0.110	0.579	0.715

Reaching the optimum is most likely with 11 tasks and least likely with 32 tasks (90.1% vs. 1.6%). Sacrificing the quality by 5% increases the likelyhoods notably $(99.1\%, 15.4\%)$. The shapes of probability curve seem reciprocal, i.e. $y = 1/x$. Although the probability of a good solution drops quickly, the knee-point is around 20 tasks and the drop after that is much smaller. This gives hope that decent results could be achieved with larger task counts, but confirming that would need very heavy brute force runs, which are currently infeasible.

Table 10 shows relative optimization time for each PE case when targeting within p percent of the global optimum. The table shows that reaching a solution within 5 percent of the global optimum can mean a significant optimization time save. This is especially true for 2 and 3 PE cases where probability of reaching global optimum with a single SA+AT run is very low. For 2 PEs, reaching within 5% of optimum takes only 1/10 of the optimization time, and within 10% of optimum takes only 1/29 of time. For 3 PEs, these correposding cases take just 1/18 and 1/43 of time. We observe that repeating SA several times for a given problem is a benefitial strategy for finding good solutions when the probability of reaching an optimum solution is low for one optimization run.

Furthermore, executing one round of SA+AT takes between 0.4s and 12.6s for 2 and 8 PEs. Expected time to reach global optimum with SA+AT is 29.1s for 2 PEs, 49.8s for 3 PEs, 12.7s for 4 PEs, 5.9s for 6 PEs and 22.0s for 8 PEs. Sacrificing solution quality to be within 5 percent of the global optimum decreases optimization time so that it is 2.9s for 2 PEs, 2.8s for 3 PEs, 4.4s for 4 PEs, 4.2s for 6 PEs and 17.3s for 8 PEs.

Results show that large N makes optimization more complex although total number of solutions M^N is rather constant. We suspect that this might be due to homogeneity of our HW platform. For example, let us assume that we move all tasks from PE_0 to PE_1, $PE_1 \rightarrow PE_2$,... $PE_{M-1} \rightarrow PE_0$. Since PEs are identical, this is just a matter of their numbering and there should be $(M-1)!$ permutations with (practically) equal performance. Consequently the design space of the mapping would be actually much smaller. With heteregeneous resources or hierarchical network this would not apply. Moving the tasks has more complex consequences due to their dependencies.

Earlier, SA+AT has been compared to Group Migration (GM), hybrid of SA anf GM Random, and Optimal Subset Mapping algorithm with large $300-$task graphs and $2, 4, 8$ PEs. [28]. In that experiment, SA+AT was the

Table 11 Comparing average gain and iteration count values for the normalized inverse exponential (NIE) and normalized exponential acceptors (NE). Higher value is better in gain and smaller in the iterations.

	Mean gain			Mean iterations		
PEs	NIE	NE	$\frac{NIE}{NE}-1$	NIE	NE	$\frac{NIE}{NE}-1$
2	1.329	1.334	−0.3%	6 758	6 882	−1.8%
3	1.566	1.576	−0.6%	13 577	14 120	−3.8%
4	1.794	1.798	−0.2%	20 734	22 398	−7.4%

best heuristic studied. The speedups w.r.t single PE were about $1.9x$, $2.7x$ and $3.7x$, respectively, but the optimum was uknown. The relation between mean iteration count and problem size is far from linear but we haven't yet formulated it. For example in this paper, 8-PE case requires about $20x$ iterations compared to 2 PEs. However in [28], the ratio is only $2.4x$ and iterations counts are rather small - $37k$ and $88k$ - considering the size of task graph.

5 On SA acceptance probability

This experiment studied the impact of acceptance function.

5.1 On acceptor functions

Normalized exponential and normalized inverse exponential acceptor functions are compared with respect to solution quality and convergence. Very few SA papers try different acceptor functions. We ran an experiment to compare the normalized inverse exponential (equation 2) and normalized exponential (equation 3) acceptor functions. Parameter selection method from [30] was applied to (equation 3). The experiment was re-run for both acceptor functions, 100 graphs, 2-4 PEs, 10 times each case, totaling 6 000 optimization runs.

Table 11 shows the average gain and iteration count results for both acceptor functions. In terms of gain, normalized exponential acceptor is better by not more than half a percent.

However, exponential acceptor needs $2-7\%$ more iterations. Both SA algorithms have an equal number of iterations per temperature from T_0 to T_f. Optimization terminates when $T \leq T_f$ and L consecutive rejections happen. Acceptance function affects the number of consecutive rejections which makes the difference in the number of iterations. Inverse exponential acceptor has higher rejection probability, which makes it stop earlier than an exponential acceptor.

Furthermore, we compared brute force results with 32 nodes and 2 PEs between normalized exponential and normalized inverse exponential acceptors. Table 12 shows the difference between global optimum converge rates for the two acceptors for SA+AT. Results indicate normalized exponential acceptor (NE) converges slightly more frequently than normalized inverse exponential

Table 12 The difference in SA+AT convergence rate between normalized exponential acceptor (NE) and normalized inverse exponential acceptor (NIE). Overhead percentage p shows the convergence within global optimum. A positive value indicates normalized exponential acceptor is better. SA+AT chooses $L = 32$.

Exec. time overhead	Convergence rate difference: (NE - NIE) / NE				
	L value				
$p = \frac{t}{t_o} - 1$	16	**32**	64	128	256
+0%	0.001	0.003	0.006	0.007	0.009
+1%	0.003	0.006	0.008	0.003	0.011
+2%	0.001	0.009	0.014	0.009	0.006
+3%	0.003	0.010	0.016	0.015	0.015
+4%	0.004	0.011	0.016	0.012	0.015
+5%	0.006	0.014	0.021	0.011	0.010
+6%	0.011	0.014	0.025	0.012	0.013
+7%	0.011	0.016	0.028	0.010	0.003
+8%	0.011	0.015	0.030	0.006	−0.004
+9%	0.014	0.016	0.031	0.004	−0.004
+10%	0.016	0.019	0.035	0.002	−0.002
+11%	0.015	0.019	0.023	0.000	−0.002
+12%	0.013	0.014	0.015	−0.004	−0.002
+13%	0.003	0.024	0.012	−0.002	−0.001
+14%	0.002	0.020	0.007	0.001	0.000
+15%	0.001	0.019	0.006	0.000	
+16%	−0.001	0.015	0.004	0.000	
+17%	0.002	0.013	0.002	0.000	
+18%	0.001	0.009	0.002	0.000	
+19%	−0.002	0.006	0.002	0.001	
+20%	−0.004	0.001	0.002	0.000	
+21%	−0.004	−0.001	0.001		
+22%	−0.005	0.000	0.000		
+23%	−0.002	0.000	0.000		
+24%	−0.002	0.000	−0.001		
+25%	−0.002	0.000	0.000		
+26%	−0.002	0.001			
+27%	−0.001	0.000			
+28%	0.000				
+29%	−0.001				
...	...				
+35%	0.000				

acceptor (NIE). The SA+AT choice $L = 32$ displays difference of at most 2.4% in absolute convergence rate.

We recommend using normalized exponential acceptor NE for task mapping.

5.2 On zero transition probability

SA acceptance function defines the probability to accept a move for a given objective change and the temperature. Zero transition probability is the probability of accepting a move that does not change the cost. One might speculate that although the cost does not change, the new mapping may be a better base for the next moves. We studied the effect of zero transition probability to the solution quality.

The normalized inverse exponential acceptance function (equation 2) gives 0.5 probability for zero transition, that is, when $\Delta C = 0$. The acceptance function was modified to test the effect of zero transition probability:

$$Accept(\Delta C, T) = \frac{2a}{1 + exp(\frac{\Delta C}{0.5 C_0 T})}, \tag{8}$$

where a is the probability for $\Delta C = 0$. SA+AT was re-run for several graphs with the setup specific in detail in [30] with distinct probabilities $a \in [0.1, 0.2, \ldots, 1.0]$. Two 16 node cyclic graphs, one 128 node cyclic graph, two 16 node acyclic graphs, and one 128 acyclic graph, all with target distribution 10%, were tested with a 3 PE system. However, no causality was found between solution quality and the zero transition probability. It seems that this probability is insignificant.

6 Recommendations

In this Section we summarize our findings and give recommendations for reporting the results on taks mapping using simulated annealing. In addition, we present a compact list how to set up SA for task mapping.

6.1 On reporting results

There are several guidelines that apply for publishing data on parallel computation [2] and heuristics in general [3]. Unfortunately, we found out that many of the cited publications leave out important details to reproduce the results with SA. Therefore, we present recommendations for publishing results on task mapping with SA as follows:

1. Report at least the **pseudocode** of the algorithm. A textual description of the algorithm is often too ambiguous for reproducing the experiment. Specify the temperature scale, cost, move, and acceptance functions.
2. Report **numerical values for constants** for the algorithm. Specify temperature scaling factor, initial and final temperatures and the number of iterations per temperature level. These are needed to re-produce the same results.
3. Report the **convergence rate**. Plot the number of mapping iterations against optimized values (e.g. speedup). Report mean and median number of iterations.
4. Compare the result from heuristics with a **global optimum** result for a trivial problem that can be solved by brute force. Report the proportion of optimization runs and the number of iterations that reach within p percent of the global optimum where p is varied. For example, it can be informative

to know that $p = 95$ percent of optimization runs yield a result that is only 5 percent worse than the global optimum and uses a specific number of iterations on average for this result.

5. Report the **optimization time per iteration**. This corresponds mostly to simulation time per mapping.

6.2 Recommended practices for task mapping with Simulated Annealing

Based on existing data and results, we recommend following practices for task mapping with SA:

1. **Scale the number of iterations per temperature level with system size**. That means that L should be proportional to $\alpha = N(M-1)$, where N is the number of tasks and M is the number of PEs. α is the number of neighboring mapping solutions. Section 4 results and [26] [30] [6] indicate that α times a small multiplier is sufficient for good convergence provided that SA is repeated several times.
   Furthermore, as a generic result it is important to note that the number of mapping iterations should grow as a function of problem complexity parameters N and M unless experimental results indicate good results for a specific L value.
2. Use **geometric temperature schedule** with $0.90 \leq q \leq 0.99$. Most known results use values in this range. Our experiments have found $q = 0.95$ to be a suitable value. The L value has to be adjusted with the q variable. Dropping q from 0.95 to 0.90 implies that the number of iterations in a given temperature range halves unless L is doubled.
3. Use a **systematic method for choosing the initial and final temperatures**, e.g. one published in [26] [29] [30] [5] [4] [6] [10] [18] [22] [38]. A systematic method decreases manual work and the risk of careless parameter selection. A systematic method may also decrease optimization time.
4. Use a **normalized exponential acceptance** function (NE) (equation 3). Section 5 indicates that using exponential rather than inverse exponential acceptance function is preferred. The zero transition probability did not explain why inverse exponential did worse. Normalized acceptance function is defined with a normalized temperature range $T \in (0, 1]$ which makes annealing schedules more comparable between problems. It is easy to select a safe but wasteful range when temperature is normalized, e.g. $T \in (0.0001, 1]$. It is also easier to add new influencing factors into the cost function since the initial cost does not directly affect the selection of initial temperature when a normalized acceptance function is used. Instead, the relative change in cost function value in moves is a factor in the selection of initial temperature.
5. Use the **ST (single task) move** function 3.1, if in doubt. It is the most common heuristics which makes also the comparison easier to other works. However, exploring the impact of move function calls for more research.

6. Run the heuristics **several times for a given problem**. Section 4 shows that repetition, that is, restarting optimization, is necessary. The variance of solution quality can be significant which comes visible by running the heuristics several times. SA terminates usually in a low temperature state in a neighborhood of a local minimum. We call this neighborhood a *valley*. It is hard to escape from a valley when the temperature is low. Running SA several times increases the probability that better valleys are found in the optimization process. Increasing the number of iterations per optimization run may not address this because the algorithm will most likely end up in a single valley. Multiple valleys should be explored.
7. Record the **iteration number when the best solution is reached**. Some algorithms continue running but make no further progress at the end of optimization. If the termination iteration number is much higher than the best solution iteration number, maybe the annealing can be stopped earlier without sacrificing quality.

7 Conclusion

A survey of state-of-the-art of task mapping with SA was presented. It was found out that SA parameters are often incompletely presented and explained in publications. Recommendations were presented on how to better report this information. Despite SA is an efficient optimization method for task mapping, there are many practices for selecting the SA parameters. Based on our experiments and findings, we presented a set of recommended parameters in detail to help researchers reproduce and compare their works. Thorough experiments with $2-8$ PEs and $11-32$ tasks showed that SA can achieve solutions very close to globally optimum and also 4-6 orders of magnitude reduction in optimization time. Our future work includes adding memory and time constraints to the objective function, evaluate applicability to larger problems, and to further automate the task mapping process as whole.

References

1. Ali, S., Kim, J.-K., Siegel, H. J. and Maciejewski, A. A., "Static heuristics for robust resource allocation of continuously executing applications", Journal of Parallel and Distributed Computing, Vol. 68, Issue 8, August 2008, pp. 1070-1080, ISSN 0743-7315, DOI: 10.1016/j.jpdc.2007.12.007.
2. Bailey, D. H., "Twelve ways to fool the masses when giving performance results on parallel computers", Supercomputer Review, Vol. 4, No. 8, pp. 54-55, 1991. [online] `http://crd.lbl.gov/~dhbailey/dhbpapers/twelve-ways.pdf`
3. Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C. and Stewart, W. R, "Designing and Reporting on Computational Experiments with heuristic Methods", Springer Journal of Heuristics, Vol. 1, No. 1, pp. 9-32, 1995.
4. Bollinger, S. W. and Midkiff, S. F., "Heuristic Technique for Processor and Link Assignment in Multicomputers", IEEE Transactions on Computers, Vol. 40, pp. 325-333, 1991.

5. Braun, T. D., Siegel, H. J. and Beck, N., “A Comparison of Eleven Static Heuristics for Mapping a Class if Independent Tasks onto Heterogeneous Distributed Systems”, IEEE Journal of Parallel and Distributed Computing, Vol. 61, pp. 810-837, 2001.
6. Coroyer, C. and Liu, Z., “Effectiveness of heuristics and simulated annealing for the scheduling of concurrent tasks an empirical comparison”, Rapport de recherche de l’INRIA Sophia Antipolis (1379), 1991.
7. DCS task mapper, “a task mapping and scheduling tool for multiprocessor systems”, 2010. [online] http://wiki.tut.fi/DACI/DCSTaskMapper
8. This paper’s experiment data files, 2012. [online] http://zakalwe.fi/~shd/task-mapping/experiment-data-2012-11.tar.gz
9. Dorigo, M. and Stützle, T., “Ant Colony Optimization”, MIT press, ISBN 0-262-04219-3, 2004.
10. Ercal, F., Ramanujam, J. and Sadayappan, P., “Task Allocation onto a Hypercube by Recursive Mincut Bipartitioning”, ACM, pp. 210-221, 1988. [online] http://dl.acm.org/citation.cfm?id=62323
11. Ferrandi, F., Pilato, C., Sciuto, D. and Tumeo, A., “Mapping and scheduling of parallel C applications with Ant Colony Optimization onto heterogeneous reconfigurable MPSoCs”, Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific, pp. 799-804, 2010.
12. Girkar, M. and Polychronopoulos, C. D., “Automatic extraction of functional parallelism from ordinary programs”, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 2, pp. 166-178, March 1992.
13. M. Gries, *Methods for evaluating and covering the design space during early design development*, Integration, the VLSI Journal, Vol. 38, Issue 2, pp. 131-183, 2004.
14. *jobqueue*, “A tool for parallelizing jobs to a cluster of computers”, 2010. [online] http://zakalwe.fi/~shd/foss/jobqueue/
15. Kahn, G., “The semantics of a simple language for parallel programming”, Proceedings of IFIP Congress 74, Information Processing 74, pp. 471-475, 1974. [online] http://www1.cs.columbia.edu/~sedwards/papers/kahn1974semantics.pdf
16. Kim, J.-K., Shivle, S., Siegel, H. J., Maciejewski, A. A., Braun, T. D., Schneider, M., Tideman, S., Chitta, R., Dilmaghani, R. B., Joshi, R., Kaul, A., Sharma, A., Sripada, S., Vangari, P. and Yellampalli, S. S., “Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment”, Journal of Parallel and Distributed Computing, Elsevier, Vol. 67, pp. 154-169, 2007.
17. Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., “Optimization by simulated annealing”, Science, Vol. 200, No. 4598, pp. 671-680, 1983.
18. Koch, P., “Strategies for Realistic and Efficient Static Scheduling of Data Independent Algorithms onto Multiple Digital Signal Processors”, Doctoral thesis, The DSP Research Group, Institute for Electronic Systems, Aalborg University, Aalborg, Denmark, December 1995.
19. *kpn-generator*, “A program for generating random Kahn Process Network graphs”, 2009. [online] http://zakalwe.fi/~shd/foss/kpn-generator/
20. Kwok, Y.-K., Ahmad, I. and Gu, J., “FAST: A Low-Complexity Algorithm for Efficient Scheduling of DAGs on Parallel Processors”, Proceedings of International Conference on Parallel Processing, Vol. II, pp. 150-157, 1996.
21. Kwok, Y.-K. and Ahmad, I., “FASTEST: A Practical Low-Complexity Algorithm for Compile-Time Assignment of Parallel Programs to Multiprocessors”, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 2, pp. 147-159, 1999.
22. Lin, F.-T. and Hsu, C.-C., “Task Assignment Scheduling by Simulated Annealing”, IEEE Region 10 Conference on Computer and Communication Systems, Hong Kong, September 1990.
23. Matousek, J. and Gärtner, B., “Understanding and Using Linear Programming”, Springer, ISBN 978-3540306979, 2006.
24. Nanda, A. K., DeGroot, D. and Stenger, D. L., “Scheduling Directed Task Graphs on Multiprocessors using Simulated Annealing”, Proceedings of 12th IEEE International Conference on Distributed Systems, pp. 20-27, 1992.
25. H. Orsila, “Optimizing Algorithms for Task Graph Mapping on Multiprocessor System on Chip”, Doctoral thesis, Tampere University of Technology, Department of Computer Systems, 2011. [Online] http://dspace.cc.tut.fi/dpub/handle/123456789/20519

26. H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, *Parameterizing Simulated Annealing for Distributing Task Graphs on multiprocessor SoCs*, International Symposium on System-on-Chip, pp. 73-76, Tampere, Finland, Nov 14-16, 2006.
27. H. Orsila, T. Kangas, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, *Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips*, Journal of Systems Architecture, Volume 53, Issue 11, ISSN 1383-7621, pp. 795-815, 2007.
28. H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, *Optimal Subset Mapping And Convergence Evaluation of Mapping Algorithms for Distributing Task Graphs on Multiprocessor SoC*, International Symposium on System-on-Chip, Tampere, Finland, Nov 19-21, 2007.
29. H. Orsila, E. Salminen, T. D. Hämäläinen, *Best Practices for Simulated Annealing in Multiprocessor Task Distribution Problems*, Chapter 16 of the Book "Simulated Annealing", ISBN 978-953-7619-07-7, I-Tech Education and Publishing KG, pp. 321-342, 2008.
30. H. Orsila, E. Salminen, T. D. Hämäläinen, *Parameterizing Simulated Annealing for Distributing Kahn Process Networks on Multiprocessor SoCs*, International Symposium on System-on-Chip 2009, Tampere, Finland, Oct 5-7, 2009.
31. Ravindran, K., "Task Allocation and Scheduling of Concurrent Applications to Multiprocessor Systems", Doctoral thesis, UCB/EECS-2007-149, 2007. [online] http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-149.html
32. SA+AT C reference implementation. [online] http://zakalwe.fi/~shd/task-mapping
33. Sato, M., "OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors", ACM, Proceedings of the 15th international symposium on System Synthesis, pp. 109-111, 2002.
34. Sih, G. C. and Lee, E. A., "A Compile-Time Scheduling Heuristics for Interconnection-Constrained Heterogeneous Processor Architectures", IEEE Transaction on Parallel and Distributed Systems, Vol. 4, No. 2, pp. 175-187, 1993.
35. Singh, A. K., Srikanthan, T., Kumar, A. and Jigang, W., "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms", Elsevier, Journal of Systems Architecture, Vol. 56, pp. 242-255, 2010. [online] http://www.es.ele.tue.nl/~akash/files/kumarJSA2010b.pdf
36. Wild, T., Brunnbauer, W., Foag, J. and Pazos, N., "Mapping and scheduling for architecture exploration of networking SoCs", Proc. 16th Int. Conference on VLSI Design, pp. 376-381, 2003.
37. W. Wolf, *The future of multiprocessor systems-on-chips*, Design Automation Conference 2004, pp. 681-685, 2004.
38. Xu, J. and Hwang, K., "A simulated annealing method for mapping production systems onto multicomputers", Proceedings of the sixth conference on Artificial intelligence applications, IEEE Press, ISBN 0-8186-2032-3, pp. 130-136, 1990.

8 Appendix: Convergence results to larger systems with 3-6 PEs

Table 13 Proportion of SA+AT runs that converged within p from global optimum for 3 PEs and 21 nodes. A higher value is better. SA+AT chooses $L = 42$. The 90% level is marked in boldface on each column.

	Proportion of runs within limit p						
	L value						
$p = \frac{t}{t_o} - 1$	16	32	**42**	64	128	256	512
+0%	0.003	0.011	0.012	0.027	0.054	0.110	0.203
+1%	0.007	0.028	0.036	0.068	0.128	0.229	0.369
+2%	0.015	0.056	0.070	0.125	0.232	0.352	0.509
+3%	0.031	0.093	0.120	0.195	0.328	0.460	0.612
+4%	0.044	0.128	0.162	0.249	0.407	0.557	0.713
+5%	0.062	0.170	0.213	0.318	0.494	0.663	0.810
+6%	0.083	0.214	0.273	0.393	0.593	0.774	**0.901**
+7%	0.112	0.271	0.339	0.475	0.682	0.857	0.951
+8%	0.141	0.321	0.398	0.543	0.750	**0.905**	0.977
+9%	0.177	0.378	0.464	0.618	0.809	0.940	0.989
+10%	0.215	0.438	0.529	0.685	0.862	0.964	0.996
+11%	0.259	0.502	0.597	0.746	**0.905**	0.983	0.999
+12%	0.304	0.563	0.660	0.801	0.939	0.993	1.000
+13%	0.354	0.626	0.718	0.849	0.963	0.997	
+14%	0.404	0.686	0.769	0.887	0.977	0.998	
+15%	0.453	0.740	0.815	**0.918**	0.986	0.999	
+16%	0.510	0.785	0.857	0.942	0.993	1.000	
+17%	0.564	0.828	0.890	0.962	0.996		
+18%	0.617	0.866	**0.918**	0.974	0.998		
+19%	0.668	0.896	0.939	0.983	1.000		
+20%	0.714	**0.922**	0.957	0.989			
+21%	0.761	0.945	0.970	0.994			
+22%	0.801	0.961	0.979	0.996			
+23%	0.836	0.972	0.986	0.998			
+24%	0.868	0.982	0.990	0.999			
+25%	0.897	0.987	0.995	0.999			
+26%	**0.920**	0.991	0.997	1.000			
+27%	0.939	0.995	0.998				
+28%	0.954	0.997	0.999				
+29%	0.965	0.998	0.999				
+30%	0.975	0.999	1.000				
+31%	0.982	0.999					
+32%	0.987	1.000					
...	...						
+39%	1.000						
+mean mappings	727	1 545	2 127	3 725	19 144	113 848	229 258
+median mappings	692	1 477	2 050	3 601	15 678	114 428	228 860

Table 14 Approximate expected number of mappings for SA+AT with 3 PEs and 21 nodes. SA+AT chooses $L = 42$. The best values (smallest) are in boldface for each performance level p (row).

	Estimated number of mappings						
	L value						
p	16	32	**42**	64	128	256	512
+0%	250 856	145 775	178 520	**138 977**	354 527	1 032 163	1 131 020
+1%	102 462	55 583	58 607	**54 693**	149 333	498 021	621 969
+2%	47 548	**27 593**	30 392	29 702	82 555	323 063	450 055
+3%	23 774	**16 615**	17 684	19 081	58 421	247 548	374 911
+4%	16 459	**12 100**	13 165	14 958	47 015	204 541	321 630
+5%	11 677	**9 095**	9 979	11 720	38 730	171 819	283 069
+6%	8 775	**7 234**	7 781	9 475	32 262	147 014	254 533
+7%	6 501	**5 712**	6 279	7 845	28 050	132 922	241 070
+8%	5 178	**4 812**	5 345	6 858	25 533	125 729	234 607
+9%	4 105	**4 091**	4 583	6 028	23 656	121 114	231 902
+10%	**3 385**	3 531	4 022	5 433	22 204	118 050	230 248
+11%	**2 804**	3 078	3 564	4 992	21 147	115 828	229 556
+12%	**2 394**	2 744	3 225	4 651	20 399	114 650	229 304
+13%	**2 054**	2468	2 965	4 388	19 888	114 190	229 258
+14%	**1 803**	2 254	2 766	4 199	19 595	114 041	
+15%	**1 606**	2 088	2 611	4 056	19 412	113 939	
+16%	**1 427**	1 967	2 482	3 953	19 287	113 870	
+17%	**1 290**	1 866	2 390	3 873	19 221	113 848	
+18%	**1 179**	1 785	2 317	3 824	19 181		
+19%	**1 089**	1 725	2 265	3 787	19 154		
+20%	**1 018**	1 676	2 223	3 767	19 150		
+21%	**955**	1 635	2 193	3 748	19 148		
+22%	**908**	1 609	2 173	3 740	19 144		
+23%	**870**	1 590	2 158	3 733			
+24%	**838**	1 574	2 149	3 729			
+25%	**811**	1 566	2 139	3 727			
+26%	**790**	1 560	2 134	3 725			
+27%	**775**	1 553	2 131	3 725			
+28%	**763**	1 550	2 130				
+29%	**754**	1 548	2 129				
+30%	**746**	1 547	2 128				
+31%	**741**	1 546	2 128				
+32%	**737**	1 546	2 127				
+33%	**734**	1 546					
+34%	**731**	1 545					
+35%	**730**	1 545					
...	...						
+42%	**727**						
mean	727	1 545	2 127	3 725	19 144	113 848	229 258
median	692	1 477	2 050	3 601	15 678	114 428	228 860

Table 15 Proportion of SA+AT runs that converged within p from global optimum for 4 PEs and 17 nodes. A higher value is better. SA+AT chooses $L = 51$.

	Proportion of runs within limit p						
	L value						
$p = \frac{t}{t_o} - 1$	16	32	**51**	64	128	256	512
+0%	0.011	0.042	0.090	0.110	0.198	0.305	0.460
+1%	0.013	0.049	0.104	0.126	0.220	0.334	0.495
+2%	0.017	0.056	0.116	0.139	0.243	0.371	0.545
+3%	0.025	0.081	0.156	0.186	0.321	0.469	0.641
+4%	0.036	0.114	0.207	0.253	0.407	0.558	0.719
+5%	0.046	0.146	0.260	0.317	0.484	0.653	0.813
+6%	0.060	0.183	0.310	0.371	0.559	0.731	0.874
+7%	0.084	0.229	0.372	0.433	0.629	0.786	**0.904**
+8%	0.110	0.279	0.440	0.509	0.705	0.853	0.955
+9%	0.137	0.325	0.497	0.565	0.755	0.890	0.970
+10%	0.171	0.381	0.565	0.631	0.816	**0.930**	0.987
+11%	0.206	0.429	0.612	0.680	0.852	0.945	0.990
+12%	0.244	0.487	0.669	0.736	0.889	0.964	0.995
+13%	0.291	0.544	0.727	0.791	**0.920**	0.976	0.997
+14%	0.337	0.600	0.780	0.837	0.945	0.986	0.999
+15%	0.390	0.656	0.823	0.872	0.962	0.992	1.000
+16%	0.445	0.706	0.858	**0.900**	0.972	0.995	
+17%	0.495	0.749	0.885	0.922	0.980	0.997	
+18%	0.546	0.791	**0.907**	0.940	0.984	0.998	
+19%	0.594	0.824	0.927	0.953	0.989	0.999	
+20%	0.643	0.852	0.939	0.963	0.993	0.999	
+21%	0.687	0.878	0.952	0.971	0.994	0.999	
+22%	0.722	0.897	0.962	0.977	0.996	0.999	
+23%	0.757	**0.915**	0.970	0.982	0.997	1.000	
+24%	0.789	0.929	0.977	0.986	0.998		
+25%	0.816	0.942	0.983	0.990	0.998		
+26%	0.843	0.953	0.988	0.993	0.999		
+27%	0.870	0.963	0.992	0.995	0.999		
+28%	0.889	0.971	0.995	0.996	1.000		
+29%	**0.905**	0.978	0.995	0.998			
+30%	0.917	0.982	0.996	0.998			
+31%	0.931	0.987	0.997	0.999			
+32%	0.942	0.990	0.998	0.999			
+33%	0.951	0.992	0.999	1.000			
+34%	0.959	0.994	0.999				
+35%	0.965	0.996	1.000				
+36%	0.969	0.996					
+37%	0.975	0.997					
+38%	0.980	0.999					
+39%	0.983	0.999					
+40%	0.986	0.999					
+41%	0.989	1.000					
...	...						
+51%	1.000						
mean mappings	786	1 703	3 146	4 549	38 332	115 650	231 412
median mappings	809	1 727	3 066	4 275	39 694	115 705	231 417

Table 16 Approximate expected number of mappings for SA+AT with 4 PEs and 17 nodes. SA+AT chooses $L = 51$.

	Estimated number of mappings						
	L value						
p	16	32	**51**	64	128	256	512
+0%	68 921	40 261	**34 875**	41 541	194 089	378 933	502 960
+1%	62 357	34 970	**30 306**	36 187	174 080	346 674	467 027
+2%	46 768	30 196	**27 119**	32 725	157 747	311 558	424 609
+3%	30 933	21 104	**20 165**	24 403	119 490	246 642	361 130
+4%	21 947	**14 979**	15 168	17 986	94 229	207 296	321 987
+5%	16 933	**11 641**	12 113	14 358	79 150	177 025	284 569
+6%	12 987	**9 317**	10 144	12 267	68 598	158 208	264 743
+7%	9 354	**7 424**	8 452	10 495	60 932	147 194	255 845
+8%	7 136	**6 115**	7 156	8 933	54 357	135 597	242 215
+9%	5 727	**5 242**	6 329	8 051	50 778	129 959	238 667
+10%	4 589	**4 476**	5 572	7 208	46 953	124 395	234 484
+11%	**3 810**	3 973	5 142	6 693	44 975	122 420	233 679
+12%	**3 223**	3 498	4 704	6 183	43 109	119 957	232 505
+13%	**2 703**	3 128	4 326	5 751	41 684	118 482	232 015
+14%	**2 334**	2 841	4 035	5 437	40 585	117 340	231 667
+15%	**2 013**	2 594	3 824	5 218	39 863	116 618	231 505
+16%	**1 764**	2 413	3 667	5 054	39 441	116 208	231 435
+17%	**1 588**	2 275	3 554	4 931	39 111	115 975	231 412
+18%	**1 439**	2 154	3 467	4 841	38 944	115 847	
+19%	**1 323**	2 067	3 394	4 775	38 771	115 801	
+20%	**1 222**	2 000	3 349	4 722	38 622	115 778	
+21%	**1 144**	1 939	3 305	4 687	38 556	115 743	
+22%	**1 088**	1 899	3 270	4 654	38 502	115 720	
+23%	**1 038**	1 861	3 242	4 630	38 459	115 662	
+24%	**996**	1 833	3 219	4 611	38 413	115 650	
+25%	**963**	1 808	3 200	4 596	38 394		
+26%	**932**	1 787	3 183	4 583	38 371		
+27%	**903**	1 769	3 170	4 572	38 355		
+28%	**884**	1 754	3 163	4 566	38 332		
+29%	**868**	1 742	3 160	4 560			
+30%	**857**	1 734	3 158	4 557			
+31%	**844**	1 726	3 154	4 554			
+32%	**834**	1 721	3 151	4 552			
+33%	**826**	1 717	3 149	4 550			
+34%	**820**	1 713	3 148	4 549			
+35%	**814**	1 711	3 146	4 549			
+36%	**811**	1 710	3 146	4 549			
+37%	**806**	1 707	3 146				
+38%	**802**	1 706	3 146				
+39%	**799**	1 705	3 146				
+40%	**797**	1 705	3 146				
+41%	**794**	1 704	3 146				
+42%	**792**	1 704	3 146				
+43%	**790**	1 703	3 146				
+44%	**789**	1 703					
+45%	**788**	1 703					
+46%	**787**						
...	...						
+57%	**786**						
mean	786	1 703	3 146	4 549	38 332	115 650	231 412
median	809	1 727	3 066	4 275	39 694	115 705	231 417

Table 17 Proportion of SA+AT runs that converged within p from global optimum for 6 PEs and 13 nodes. A higher value is better. SA+AT chooses $L = 65$.

	Proportion of runs within limit p						
	L value						
$p = \frac{t}{t_o} - 1$	16	32	64	**65**	128	256	512
+0%	0.086	0.325	0.576	0.578	0.731	0.869	**0.941**
+1%	0.184	0.448	0.666	0.662	0.784	0.897	0.962
+2%	0.206	0.471	0.692	0.684	0.810	**0.919**	0.972
+3%	0.249	0.548	0.767	0.758	0.861	0.943	0.982
+4%	0.292	0.596	0.793	0.782	0.872	0.949	0.986
+5%	0.325	0.638	0.825	0.818	**0.900**	0.962	0.990
+6%	0.374	0.689	0.852	0.846	0.915	0.968	0.993
+7%	0.434	0.751	**0.907**	**0.908**	0.967	0.993	0.999
+8%	0.484	0.785	0.920	0.921	0.974	0.995	1.000
+9%	0.533	0.816	0.936	0.938	0.982	0.997	
+10%	0.605	0.868	0.965	0.963	0.993	0.999	
+11%	0.646	0.886	0.970	0.968	0.995	0.999	
+12%	0.685	**0.907**	0.977	0.976	0.996	1.000	
+13%	0.728	0.929	0.985	0.986	0.998		
+14%	0.768	0.951	0.994	0.995	1.000		
+15%	0.812	0.967	0.997	0.999			
+16%	0.858	0.980	0.998	0.999			
+17%	0.888	0.987	0.999	1.000			
+18%	**0.912**	0.991	0.999				
+19%	0.937	0.995	0.999				
+20%	0.960	0.998	1.000				
+21%	0.969	0.999					
+22%	0.977	1.000					
...	...						
+32%	1.000						
mean mappings	768	1 983	8 271	8 662	55 426	115 705	231 417
median mappings	752	1 802	6 425	6 679	57 858	115 880	231 688

Table 18 Approximate expected number of mappings for SA+AT with 6 PEs and 13 nodes. SA+AT chooses $L = 65$.

	Estimated number of mappings						
	L value						
p	16	32	64	**65**	128	256	512
+0%	8 976	**6 094**	14 372	14 981	75 854	133 208	246 057
+1%	**4 174**	4 426	12 417	13 077	70 688	128 991	240 458
+2%	**3 732**	4 213	11 954	12 656	68 385	125 944	238 132
+3%	**3 086**	3 618	10 781	11 425	64 345	122 751	235 755
+4%	**2 628**	3 329	10 434	11 075	63 555	121 871	234 655
+5%	**2 366**	3 107	10 020	10 596	61 571	120 313	233 801
+6%	**2 054**	2 877	9 704	10 235	60 562	119 493	233 142
+7%	**1 770**	2 641	9 118	9 543	57 324	116 544	231 556
+8%	**1 588**	2 526	8 992	9 409	56 929	116 321	231 440
+9%	**1 440**	2 431	8 838	9 231	56 442	116 053	231 417
+10%	**1 270**	2 283	8 573	8 993	55 800	115 878	
+11%	**1 190**	2 238	8 527	8 945	55 716	115 774	
+12%	**1 121**	2 187	8 470	8 871	55 638	115 763	
+13%	**1 056**	2 135	8 398	8 785	55 543	115 728	
+14%	**1 000**	2 085	8 324	8 706	55 437	115 705	
+15%	**946**	2 051	8 292	8 675	55 426		
+16%	**895**	2 023	8 285	8 667			
+17%	**865**	2 009	8 279	8 666			
+18%	**843**	2 002	8 277	8 666			
+19%	**820**	1 994	8 277	8 665			
+20%	**800**	1 986	8 272	8 663			
+21%	**793**	1 985	8 271	8 663			
+22%	**786**	1 984		8 662			
+23%	**781**	1 983					
+24%	**777**	1 983					
+25%	**775**	1 983					
...	...						
+34%	**768**						
mean	768	1 983	8 271	8 662	55 426	115 705	231 417
median	752	1 802	6 425	6 679	57 858	115 880	231 688