Parameterizing Simulated Annealing for Distributing Kahn Process Networks on Multiprocessor SoCs

Heikki Orsila, Erno Salminen and Timo D. Hämäläinen
Department of Computer Systems
Tampere University of Technology
P.O. Box 553, 33101 Tampere, Finland
Email: {heikki.orsila, erno.salminen, timo.d.hamalainen}@tut.fi

***Abstract*—Mapping an application on Multiprocessor System-on-Chip (MPSoC) is a crucial step in architecture exploration. The problem is to minimize optimization effort and application execution time. Simulated annealing (SA) is a versatile algorithm for hard optimization problems, such as task distribution on MPSoCs. We propose an improved automatic parameter selection method for SA to save optimization effort. The method determines a proper annealing schedule and transition probabilities for SA, which makes the algorithm scalable with respect to application and platform size. Applications are modeled as Kahn Process Networks (KPNs). The method was improved to optimize KPNs and save optimization effort by doing sensitivity analysis for processes. The method is validated by mapping 16 to 256 node KPNs onto an MPSoC. We optimized 150 KPNs for 3 architectures. The method saves over half the optimization time and loses only 0.3% in performance to non-automated SA. Results are compared to non-automated SA, Group migration, random mapping and brute force algorithms. Global optimum solution are obtained by brute force and compared to our heuristics. Global optimum convergence for KPNs has not been reported before. We show that 35% of optimization runs reach within 5% of the global optimum. In one of the selected problems global optimum is reached in as many as 37% of optimization runs. Results show large variations between KPNs generated with different parameters. Cyclic graphs are found to be harder to parallelize than acyclic graphs.**

I. Introduction

An efficient multiprocessor SoC (MPSoC) implementation requires automated exploration to find an efficient HW allocation, task mapping and scheduling [1]. Heterogeneous MPSoCs are needed for low power, high performance, and high volume markets [2]. 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 component to some processing element (PE). Scheduling means determining execution timetable of the application components on the platform. A large design space must be pruned systematically, since the exploration of the whole design space is not feasible [1]. 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 execution time, energy consumption and silicon area constraints etc. Figure I(a) shows the mapping process.

We present an experiment where a set of applications modeled as Kahn Process Networks (KPNs) [3] are mapped on MPSoCs that have 2 to 4 PEs connected with dual shared bus. Figure I(b) shows a conceptual view of the application, its mapping, and the hardware platform. The application is optimized (mapped) for each architecture with respect to the application execution time. Resulting execution and optimization time values are compared to other mapping methods and global optimum solutions. Global optimum solutions are found by brute force search for small KPNs. We have not seen a similar comparison in any paper. It is found that KPN and architecture structure has a significant impact on optimization.

II. Related Work

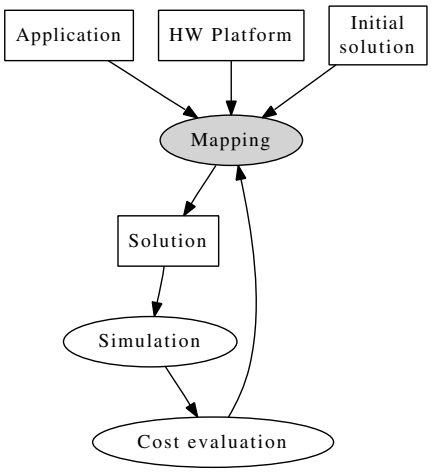
A. Kahn Process Networks (KPNs)

KPN is a distributed model of computation where a directed graph models communicating processes that can be mapped freely to PEs. Nodes in the graph do computation. Edges are communication links that are unbounded FIFOs. Each node executes a program that can read from its incoming FIFOs and write to outgoing FIFOs. There is no restriction to what the process may compute. Our earlier work used Static Task Graphs (STGs) [4]. STG is a special of KPN where the graph is acyclic, and each node does only one read-compute-write cycle, in that order.

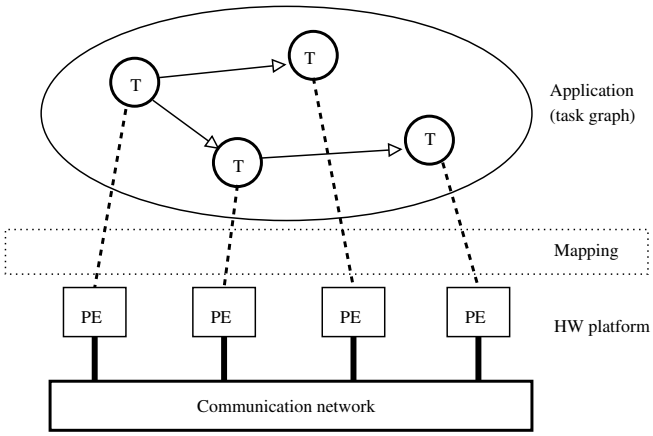
B. Algorithms For Task Mapping

Architecture exploration needs automatic tuning of optimization parameters for architectures of various sizes. Without scaling, algorithm may spend excessive time optimizing a small system, or the solution is sub-optimal for a large system. Wild *et al.* [5] compared SA, Tabu Search (TS) [6] and various other algorithms for task distribution. The parameter selection for SA had geometric annealing schedule that did not consider application or system architecture size, and thus did not scale up to bigger problems without manual tuning of parameters.

Braun *et al.* [7] compared 11 optimization algorithms for task distribution. TS outperformed SA in [5], but was worse in [7], which can be attributed to different parameter selection used. Braun's method has a proper initial temperature selection



(a) Optimization process. Boxes indicate data. Ellipses indicate operations.



(b) An example MPSoC that is optimized. The system consists of the application and the HW platform. T denotes a task, and PE denotes a processing element.

Fig. 1. Diagram of the optimization process and the system that is optimized

for SA to normalize transition probabilities, but their annealing schedule does not scale up with application or system size, making both [5] and [7] unsuitable for architecture exploration.

Our earlier papers have surveyed SA for task distribution [8], devised a method for automatically determining SA parameters [9] for distributing STGs, optimized memory consumption in distributing STGs [10], compared SA to other algorithms with respect to its convergence proprties [11], and optimized task graphs for heterogeneous architectures [12].

This paper extends and improves the method presented in [9]. The transition from STGs to KPNs requires changes to the parameter selection method that determines an efficient annealing schedule. Also, the method is improved by doing sensitivity analysis for processes. Processes with lowest execution times are ignored in the temperature range calculation to save optimization effort. Number of mapping iterations becomes smaller (optimization time is saved) but quality of results is not significantly reduced. Finally, this paper presents comparison to brute force global optimums that does not exist in earlier work. The presented method can be applied with general process networks, not just KPNs in this experiment.

```
SIMULATED_ANNEALING(S_0, T_0)
 1  S ← S_0
 2  C ← COST(S_0)
 3  S_best ← S
 4  C_best ← C
 5  R ← 0
 6  for i ← 0 to ∞
 7  do T ← TEMPERATURE(T_0, i)
 8     S_new ← MOVE(S, T)
 9     C_new ← COST(S_new)
10     ΔC ← C_new − C
11     if ΔC < 0 or RANDOM() < ACCEPT(ΔC, T)
12        then if C_new < C_best
13                then S_best ← S_new
14                     C_best ← C_new
15             S ← S_new
16             C ← C_new
17             R ← 0
18        else if T ≤ T_f
19                then R ← R + 1
20                     if R ≥ R_max
21                       then break
22  return S_best
```

Fig. 2. Pseudo-code of the Simulated annealing algorithm. Bolded functions are modified for the presented parameter selection method.

C. Simulated Annealing

SA is a probabilistic non-greedy algorithm [13] that explores search space of a problem by moving from a high to a low temperature state. At each temperature level, SA moves one or several tasks to different PEs and evaluates the cost of new mapping solutions for each move. The algorithm always accepts a move into a better state, but also into a worse state with a changing probability. This probability decreases along with the temperature, and thus the algorithm becomes greedier. The algorithm terminates when the final temperature is reached and sufficient number of consecutive moves have been rejected.

Fig. 2 shows the pseudo-code of the SA algorithm used with the new method for parameter selection. Implementation specific issues compared to the original algorithm are explained in Section III. The $Cost$ function evaluates execution time of a specific mapping by calling the scheduler. S_0 is the initial mapping of the system, T_0 is the initial temperature, and S and T are current mapping and temperature, respectively. **Temperature** function computes a new temperature as a function of initial temperature T_0 and iteration i. R is the number of consecutive rejects. $Move$ function moves a random task to a random PE, different than the original PE. $Random$ function returns an uniform random value from the interval $[0, 1)$. **Accept** function computes a probability for accepting a move that increases the cost. R_{max} is the maximum number of consecutive rejections allowed after the final temperature has been reached.

III. The Parameter Selection Method for KPNs

The parameter selection method defines $Temperature$ and $Accept$ functions for the pseudo-code presented in Figure 2. This creates an efficient annealing schedule with effective transition probabilities. We call this method *Simulated annealing with automatic temperature (SA+AT)*.

A. Temperature Function

$Temperature$ function is chosen so that annealing schedule length is proportional to application and system architecture size. Moreover the initial temperature T_0 and final temperature T_f must be in the relevant range to affect acceptance probabilities efficiently. The method uses

$$Temperature(T_0, i) = T_0 * q^{\lfloor \frac{i}{L} \rfloor}, \tag{1}$$

where L is the number of mapping iterations per temperature level, q is geometric temperature scaling factor, and i is the current iteration count. $\lfloor \frac{i}{L} \rfloor$ means round down. Temperature T is multiplied by q every L iterations. We use $q = 0.95$. It has been found suitable in our earlier papers [9][11][8]. Determining proper L value is important to assign more iterations for larger applications and systems. This method uses

$$L = N(M - 1), \tag{2}$$

where N is the number of tasks and M is the number of PEs in the system. Also, termination condition $R_{max} = L$.

When T_0 and T_f are known, the total number of iterations i_{total} is approximately

$$i_{total} \sim \frac{\log \frac{T_f}{T_0}}{\log q} L \tag{3}$$

Each iteration must determine the cost of mapping which is the most time consuming operation in optimization because it is done by simulation. Therefore, it is important to minimize the number of temperature levels by annealing only at efficient temperature range. Also, too low a value for L will result in poor optimization result, and hence a delicate trade-off is needed.

B. Accept Function

Moving tasks to different PEs affects the cost C (execution time in this case) of the system. The absolute cost value is case-dependent. Therefore the cost change ΔC is normalized in the acceptance function by a factor $C_0 = Cost(S_0)$, which is the initial cost of the non-optimized system. Relative cost $\Delta C_r = \frac{\Delta C}{C_0}$ adapts to problems that have different process execution times and makes the algorithm more general purpose. The accept function is defined as

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

This puts relevant transition probabilities into temperatures range $(0, 1]$ so that temperatures are more comparable between different problems. As the temperature decreases the accepted cost changes less chaotically and the algorithm becomes greedier.

C. Determining Temperature Upper And Lower Bounds

The initial temperature is chosen by

$$T_0 = \frac{k t_{max}}{t_{minsum}}, \tag{4}$$

where t_{max} is the maximum execution time for any task on any PE, t_{minsum} the sum of execution times for all tasks on the *fastest* PE in the system, and $k \geq 1$ is a constant that gives a temperature safety margin. Section V-A will show that $k = 2$ is sufficient in our experiment. A proper range for k is in $[1, 3]$. Mathematical properties of k is discussed in more detail in [8]. The rationale is choosing an initial temperature where the biggest single task will have a fair transition probability of being moved from one PE to another. Section V-A will show that efficient annealing happens in the temperature range predicted by the method. The chosen final temperature is

$$T_f = \frac{t_{min}}{k t_{maxsum}}, \tag{5}$$

where t_{min} is the minimum execution time for any task on any PE and t_{maxsum} the sum of execution times for all tasks on the *slowest* PE in the system. Derivation of Equations (4)(5) is explained in [8] (Orsila case). T_0 and T_f are inside range $(0, 1]$.

Execution times t_{min} and t_{max} should be determined by profiling or analyzing the KPN. To determine t_{min}, execute the KPN on the *fastest* PE available. Record execution time $t_i > 0$ for each process, i.e. exclude processes that are not executed, and compute $t_{minsum} = \sum t_i$. Sort execution times t_i in increasing order, drop the lowest r percent of values, then set t_{min} equal the lowest remaining value. Dropping the lowest r percent of execution times excludes a set of processes whose execution time is at most proportion $\frac{r}{100}$ of total execution time. This sensitivity analysis is done to save optimization iterations. It is an improvement over the method presented in [9]. We use $r = 5$. Setting $r = 0$ is a safe choice, but it often yields longer annealing schedules and does not improve the solution.

Consider a KPN where one process executes for only a microsecond, and others a millisecond. Ignoring the microsecond process scales T_f up by 1000 saving many temperature levels and optimization effort (3). However, not ignoring it would add $\frac{\log \frac{1}{1000}}{\log q} = 135$ temperature levels of optimization effort, or evaluating $135L$ mappings. Effort of evaluating a single mapping depends on the simulation model. Evaluating a single mapping takes only a fraction of a second in this paper's experiment, but others may use instruction set simulators that take minutes to evaluate a single mapping. Optimizing the placement of the microsecond process is not significant for execution time, but it could double the optimization effort.

Execution time t_{max} is computed in a similar way. Execute the KPN on the *slowest* PE available and record execution time $t_i > 0$ for each process, and compute $t_{maxsum} = \sum t_i$. Set $t_{max} = \max t_i$.

If execution times are non-deterministic, run the profiling several times, and choose the lowest T_f and highest T_0. Alternatively, it is possible to increase the value of k.

Choosing initial and final temperature properly saves optimization iterations. On a high temperature, the optimization is practically *Monte Carlo* optimization because it accepts moves to worse positions with a high probability. And thus, it will converge very slowly to optimum because the search space size is in $O(M^N)$. Also, too low a probability reduces the annealing to greedy optimization. Greedy optimization becomes useless after a short time because it cannot escape local minimums.

IV. Experiment Setup

A. Algorithms

Five optimization algorithms were used to optimize mappings on several architectures. The goal is to minimize KPN's execution time with least mapping iterations.

1) SA+AT presented in Section III
2) SA+ST is the same as SA+AT but uses static temperature bounds: $T_0 = 1.0000$ and $T_f = 0.0001$. That is, automatic temperature range selection is not used.
3) Brute force search is only used for 16 node KPNs due to $O(N^M)$ mapping space size. This gives global optimum results that are compared to SA+AT.
4) *Group migration* [14] is a greedy deterministic clustering algorithm
5) *Random mapping* chooses a random PE for all processes at each iteration. As new solutions do not depend on previous iterations, random mapping is completely unsystematic. It is the simplest non-greedy stochastic heuristics that only reveals inherent parallelism in the problem. Any algorithm should be better than random mapping to justify its existence. Ironically, we got hit by this in convergence experiment in Section V-B.

B. Simulated HW Architecture

Several experiments were run by simulating 3 MPSoC architectures. They differ only in the number of PEs. Parameters of simulated architectures are listed in Table I. The architecture is a message passing system where each PE has some local memory, but no shared memory. Each PE and interconnection resource is available for a single action at a time. PEs are interconnected with two shared buses that are independently and dynamically arbitrated. Shared bus contention, latency and throughput set limits on performance. The system uses an event based time-behavior level simulator that rolls a timewheel. The timewheel uses continuous time with 64 bit floating point accuracy. Execution times for instructions in processes come from the KPN model. A process is blocked until it gets the input that it requests. Each PE has processes that are scheduled in the order that they become ready for execution (FIFO). Shared bus messages are queued in FIFO order. Figure I(b) presents a conceptual view of the architecture.

C. Generated Kahn Process Networks

KPNs were generated with kpn-generator [15] snapshot 2009-01-28. Table II lists parameters for kpn-generator. Acyclic and cyclic directed graphs (KPNs) were generated.

TABLE I
System Architecture Parameters

Parameter	Value
Number of PEs	2 - 4
PE frequency	300 MHz
Number of buses	2
Bus frequency	200 MHz
Bus type	Shared bus, 32 bits wide, 8 cycle arbitration, arbitration policy: FIFO, either bus that can be acquired first

TABLE II
Kahn Process Networks. N is the number of nodes. (*) indicates a sum value for the whole KPN. x is the target distribution value for kpn-generator.

Parameter	Value
N	16, 32, 64, 128 and 256
KPN categories	T1: 50 acyclic graphs with $x = 100\%$, T2: 50 cyclic graphs with $x = 100\%$, T3: 50 cyclic graphs with $x = 10\%$
For each category	10×16 node graphs, 10×32, ... 10×256, totaling 50 graphs for each KPN category
Computation cycles (T)	$2^{13}N$ (*)
Computation events (C)	$8N$ (*)
Communication bytes (S)	$2^{14}N$ (*)
b model value	0.7 for computation time and communication size randomization
kpn-generator parameters	-n N -c C -t T -s S –target-distribution=x

Acyclic graphs are such that there is no directed path from a node back to itself. If this criterion is not met, the graph is cyclic. Cyclic graphs have feedback, and therefore, they are closer to real applications than acyclic graphs.

Target distribution defines the maximum number of write target nodes for each node. Target distribution 10% means a node can only have directed edges to $1 + round(0.1N)$ processes. Lower value means more regular structure. Targets are uniformly randomized. 100% target distribution is very uncommon in real applications. A low target distribution value is more realistic.

Each KPN process is a program that consists of 3 kinds of instructions: reads, computation events that consume CPU cycles, and writes. Arbitrary flow control is allowed inside programs. Therefore, KPN is a general model of computation. The total sum of computation cycles and communication sizes (writes) was generated by a b model that is a random number sequence generator that produces *self-similar fractal-like* patterns [16]. Default value $b = 0.7$ was used.

Varied KPN parameters are number of nodes N, target distribution, cyclicity of graphs, total execution time, total communication size and the number of computation events. There are 3 KPN categories: T1, T2 and T3. Each category has 50 KPNs, totaling 150 KPNs.

D. Setup A and B

Experiments were done with two setups: A and B. Setup A uses category T1 and T2 KPNs from Table II and architecture from Table I. Setup B uses category T1 and T3 KPNs from Table II and the architecture from Table I with the modification that each byte that is sent across the bus also costs one cycle

TABLE III
AUTOMATIC TEMPERATURE: SA+AT VS SA+ST SPEEDUP VALUES WITH SETUP A

	2 PEs SA+AT	2 PEs SA+ST	3 PEs SA+AT	3 PEs SA+ST	4 PEs SA+AT	4 PEs SA+ST
Min	1.662	1.673	2.004	2.008	2.065	2.076
Mean	**1.923**	**1.928**	**2.458**	**2.465**	**2.585**	**2.588**
Std	0.073	0.071	0.152	0.150	0.101	0.098
Med	1.948	1.951	2.486	2.493	2.628	2.629
Max	2.022	2.046	2.665	2.678	2.777	2.770

for the CPU. The motivation for difference between setup A and B is explained in Section V-B. In both setups SA+AT and SA+ST algorithms were run 10 times independently for each KPN and architecture combination. Initially all nodes are mapped to a single PE.

E. Software

The optimization software and simulator was written in C language and executed on a GNU/Linux cluster of 9 machines, each machine having a 2.8 GHz x86 processor and 1 GiB of memory. Jobs were distributed to a cluster with *jobqueue* [17]. A total of $5.23 \cdot 10^8$ mappings was evaluated in 66.9 computation days leading to average of $90 \frac{mappings}{s}$.

V. EXPERIMENTS AND RESULTS

Three experiments were performed.

A. Automatic Temperature Experiment

This experiment compares speedups obtained with SA+AT and SA+ST for 2, 3 and 4 PEs with setup A and B. The purpose is to study the trade-off between optimization time and optimality of the results.

SA+AT uses dynamic temperature range that is analyzed from the KPN. SA+ST uses a static (non-automatic) temperature range $[0.0001, 1]$. With $q = 0.95$ this yields 180 temperature levels for SA+ST, whereas SA+AT uses approximately half the number of levels. For 2 PEs the $L = N(M-1)$ range is $[16, 256]$, 3 PEs $[32, 512]$ and 4 PEs $[48, 768]$, regardless of the temperature selection method,

Table III shows average speedup values for each case. In the 2 case, SA+AT has 1.923 mean speedup, and SA+ST has 1.928. SA+ST yields only 0.3% larger speedup which is negligible. The 3 PE case also has a 0.3% difference. The 4 PE case has only 0.1%. TableIV shows the number of mappings for each case. For 2 PE case SA+AT uses only 36% of SA+ST iterations. For 3 PE and 4 PE cases the same values are 37% and 49%, respectively. Results for setup B are similar.

Therefore, results indicate that automatic temperature method does not underestimate temperature bounds. A negligible speedup loss of 0.3% is observed but at least half the iterations are saved.

Table V shows values that were generated by the parameterization method. Max iterations was obtained from experiment results.

Figure 3 shows the effect of L to speedup and number of iterations for 4 PEs 256 node cyclic T3 KPNs from setup B. The parameterization method selects $L = 768$ which yields

TABLE IV
AUTOMATIC TEMPERATURE: SA+AT VS SA+ST NUMBER OF MAPPINGS WITH SETUP A

	2 PEs SA+AT	2 PEs SA+ST	3 PEs SA+AT	3 PEs SA+ST	4 PEs SA+AT	4 PEs SA+ST
Min	470	2 900	960	5 800	1 590	8 690
Mean	6 500	17 960	13 340	35 910	26 150	53 900
Std	6 540	15 810	13 180	31 610	27 110	47 470
Med	3 530	11 590	7 390	23 170	13 800	34 760
Max	25 150	46 340	51 550	92 930	100 450	140 510

TABLE V
AUTOMATIC TEMPERATURE PARAMETERIZATION VALUES FOR SA+AT WITH SETUP B. MEAN TLEVS IS THE MEAN NUMBER OF TEMPERATURE LEVELS. MAX ITERS. IS OBTAINED FROM EXPERIMENT RESULTS.

	2 PEs		3 PEs		4 PEs	
N	16	256	16	256	16	256
L	16	256	32	512	48	768
T_0 mean	0.2443	0.0379	0.2443	0.0379	0.2443	0.0379
T_f mean	0.0223	0.0012	0.0223	0.0012	0.0223	0.0012
Mean Tlevs	47	67	47	67	47	67
Max iters	1 510	29 420	3 060	59 730	4 840	92 080

1.73 speedup, $L = 1024, 2048, 4096$ yield 1.74, 1.76 and 1.80, respectively. Setting $L = 4096$ multiplies number of iterations by 6.5 and increases speedup only by 4.1%. The default L is efficient, but well performing.

B. Convergence Experiment

This experiment compares convergence properties of SA+AT, Group migration and random mapping.

Figure 4 shows SA+AT, Group migration and random mapping speedup convergence plotted as a function of mapping iterations. Speedup is the non-optimized execution time divided by the optimized execution time. Higher is better. The system being optimized has 4 PEs and category T2 KPNs (setup A). The first 1000 iterations are omitted for clarity.

Both SA+AT and random mapping reach average speedup 2.56. Group migration reaches only 2.39 (a greedy algorithm

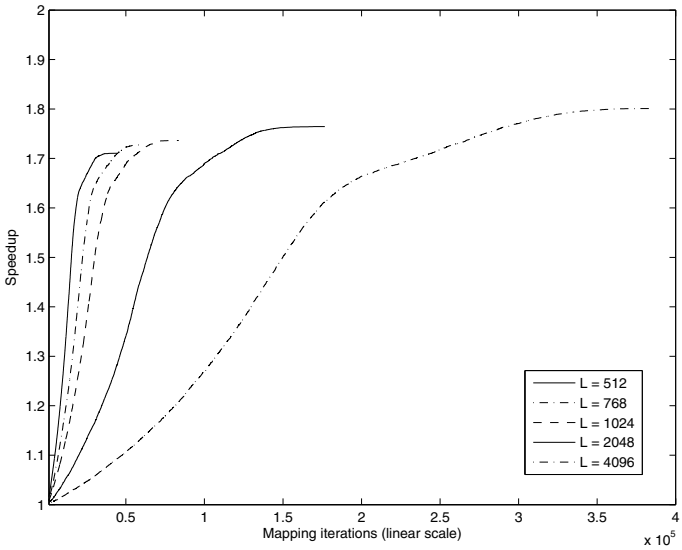


Fig. 3. SA+AT convergence: Setup B: Speedup and the number of iterations as a function of L for 4 PEs 256 node T3 KPNs. The method selects $L = 768$. Both total number of iterations and maximum speedup grow with L. In the leftmost line $L = 512$, rightmost $L = 4096$.

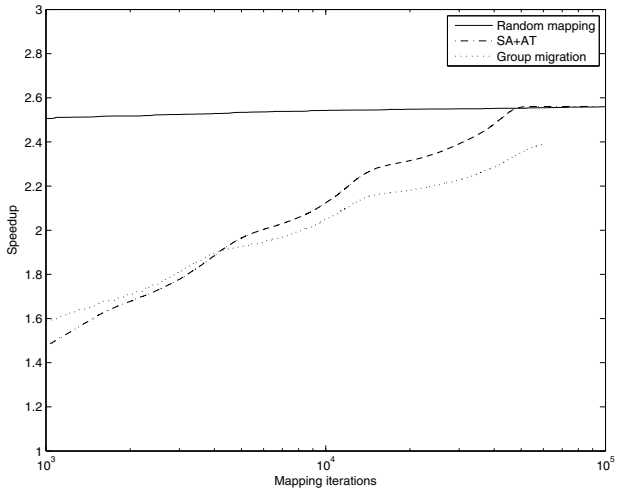


Fig. 4. Convergence: Setup A: Average speedup plotted against mapping optimization iterations. The system being optimized has 4 PEs and category T2 KPNs. Curiously all algorithms reach practically the same speedup. Speedup is computed as the non-optimized execution time divided by the optimized execution time. Higher is better.

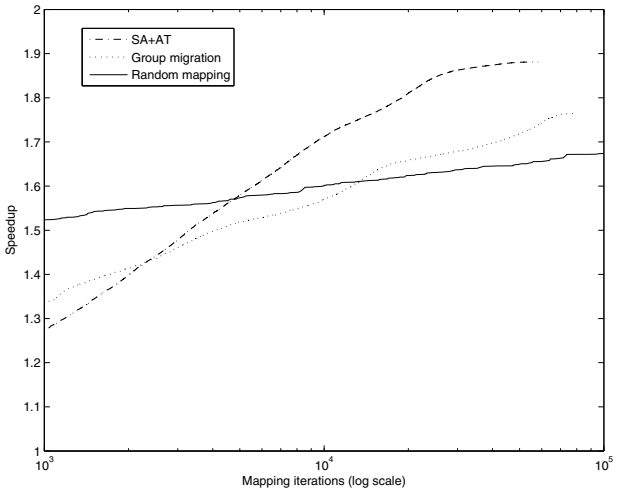


Fig. 5. Convergence: Setup B: Average speedup plotted against mapping optimization iterations for 4 PEs.

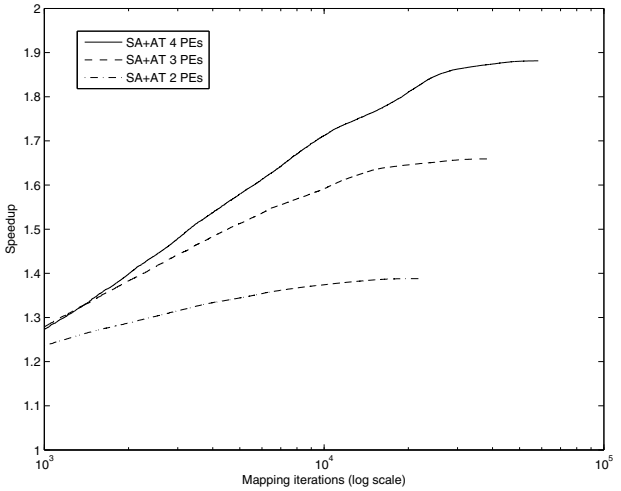


Fig. 6. Convergence: Setup B: Average speedup plotted against mapping optimization iterations for 2, 3 and 4 PEs.

cannot overcome local minimums), which is 6.6% worse than SA+AT and random. Random mapping being as good as SA+AT seems odd. Random mapping has been inferior to SA+AT in all other tests we have seen, which makes this an interesting case. Random mapping should converge very slowly as all iterations are independent of each other, no systematic techniques (such as local search) or memory is used, unlike in SA+AT. SA+AT reached 2.56 at $49\ 300$ iterations. Random mapping converged very rapidly near the maximum. The second iteration already had 2.22 speedup. At 1000 iterations it had 2.51 which is only 2% less than the best solution.

Next we tried category T3 KPNs with 4 PEs. These KPNs differ from previous case by having target distribution 10% which forces more organization into the graph. For example, 256 node graphs can have 27 distinct targets, 16 node graphs can have only 3 targets. Also, we added a CPU cost of 1 cycle per byte for writes from one PE to another. Statistically this encourages PE locality to nodes that interact with each other. Both of these changes, independently or together, changed results in favor of SA+AT over random mapping.

Convergence is shown in Figure 5. SA+AT reaches average speedup 1.88, Group migration 1.76 and random mapping 1.68. Now random mapping is the worst. Much lower random mapping speedups indicate that there is little easy parallelism. Random mapping cannot do local search, and now even the greedy Group migration outperforms it. This is due to increased organization in the graph that encourages PE locality between nodes that interact. We believe previous test with setup A had too little systematic optimization opportunities, and therefore random was equally good as SA+AT, but in this case SA+AT wins because of reparameterization. In both cases SA+AT has better speedup than Group migration.

SA+AT and Group migration show similar convergence between Figures 4 and 5, but random mapping does not.

Figure 6 shows convergence for SA+AT on 2, 3 and 4 PEs. Speedups are averaged: 4 PEs has 1.81, 3 PEs 1.58 and 2 PEs 1.34. There is not much parallelism available, but convergence looks similar in each case. It should be noted that the number of optimization iterations grows with the number of PEs. It is a consequence of our parameterization method that the number of iterations per temperature level is $L = N(M-1)$.

C. Brute Force Experiment

This experiment compares SA+AT heuristics to global optimum solutions obtained by brute force search. Brute force experiment parameters are listed in Table VI. The experiment is run for setup A and B with 16 node KPNs. We were unable to run brute force search with available computational capacity for problems larger than 16 nodes or more than 4 PEs. Without loss of generality, one of the 16 nodes was fixed to a PE to decrease optimization iterations. The brute force search space size becomes $2^{15} \sim 3.2E4$ and $3^{15} \sim 1.4E7$ mappings for 2 and 3 PE cases, respectively. A heuristic random algorithm does not always have the same results. Hence, SA+AT is

TABLE VI
BRUTE FORCE EXPERIMENT PARAMETERS

Parameter	Value
Algorithms	SA+AT, brute force
KPNs	16 node setup A KPNs from T1 and T2 (10 + 10), 16 node setup B KPNs from T1 and T3 (10 + 10),
Architectures	2 and 3 PEs from setups A and B
SA+AT runs	2 setups ×2 archs ×20 graphs ×1000 independent optimization runs = 80 000 SA+AT runs

TABLE VII
BRUTE FORCE VS. SA+AT WITH SETUP A (100% TARGET DISTRIBUTION): PROPORTION OF SA+AT RUNS THAT CONVERGED WITHIN p FROM GLOBAL OPTIMUM

Exec. time overhead	Proportion of runs within limit p			
	2 PEs		3 PEs	
$p = \frac{t}{t_o} - 1$	acyclic	cyclic	acyclic	cyclic
+0%	0.043	0.033	0.004	0.002
+1%	0.104	0.085	0.017	0.014
+2%	0.287	0.268	0.090	0.051
+3%	0.547	0.458	0.274	0.120
+4%	0.770	0.647	0.476	0.231
+5%	0.892	0.836	0.678	0.392
+6%	0.948	0.936	0.842	0.593
+7%	0.976	0.987	0.941	0.776
+8%	0.995	0.999	0.983	0.913
+9%	1.000	1.000	0.997	0.975
+10%	1.000	1.000	1.000	0.995
+11%	1.000	1.000	1.000	0.999
+12%	1.000	1.000	1.000	1.000
mean mappings	748	790	1774	1754
median mappings	760	756	1759	1736

run independently 1000 times for each KPN. The results are recorded and compared to global optimum.

The optimality of SA+AT is shown in Tables VII and VIII. Tables show the proportion of 1000 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. $p = 0\%$ means the global optimum result. Optimum values (mappings) were obtained by brute force search. The last two rows show mean and median values for the number of mappings tried in a single SA+AT run. The results are shown for 2 and 3 PEs with 16 node acyclic and cyclic Kahn Process Networks. Values from Table VIII are plotted in Figure 7. The difference between Tables VII and VIII is the setup, the former uses setup A and the latter uses setup B. Setup A has uniform target distribution (100%). Setup B graphs are more organized (target distribution 10%) and have less free parallelism. The number of needed iterations (i.e. optimization time) is shown in Tables IX and X.

Uniform target distribution (setup A) has an interesting property that the fewer optimization runs have optimum solution ($p = 0\%$) than in setup B, but all solutions in setup A are closer to global optimum. For example, in setup A 3 PE cyclic case, global optimum was reached in 2 out of 1000 SA+AT runs. The same value is 111 for setup B. However, all solutions came within 12% of optimum cost in setup A, but 28% in setup B. Therefore, it is easier to reach global optimum in setup B, but the variance is higher. Moreover, the mean number of mappings to reach global optimum was reduced from 923 150 to 15 230 which is 98% reduction due to 10% target distribution (Table IX and X). Brute force

TABLE VIII
BRUTE FORCE VS. SA+AT WITH SETUP B (10% TARGET DISTRIBUTION): SEE TABLE VII

Exec. time overhead	Proportion of runs within limit p			
	2 PEs		3 PEs	
$p = \frac{t}{t_o} - 1$	acyclic	cyclic	acyclic	cyclic
+0%	0.358	0.374	0.056	0.111
+1%	0.418	0.435	0.072	0.160
+2%	0.540	0.478	0.101	0.223
+3%	0.612	0.543	0.173	0.275
+4%	0.685	0.622	0.271	0.349
+5%	0.792	0.650	0.348	0.435
+6%	0.856	0.716	0.446	0.493
+7%	0.884	0.730	0.538	0.554
+8%	0.918	0.756	0.623	0.609
+9%	0.940	0.789	0.722	0.692
+10%	0.962	0.817	0.801	0.747
+11%	0.969	0.846	0.867	0.785
+12%	0.980	0.886	0.913	0.820
+13%	0.992	0.922	0.945	0.868
+14%	0.996	0.951	0.967	0.890
+15%	0.997	0.960	0.981	0.907
...	...	...	...	...
+18%	1.000	0.984	0.998	0.953
+20%		0.992	1.000	0.975
+25%		0.999		0.997
+26%		1.000		0.998
+28%				1.000
mean mappings	795	824	1637	1692
median mappings	778	788	1610	1579

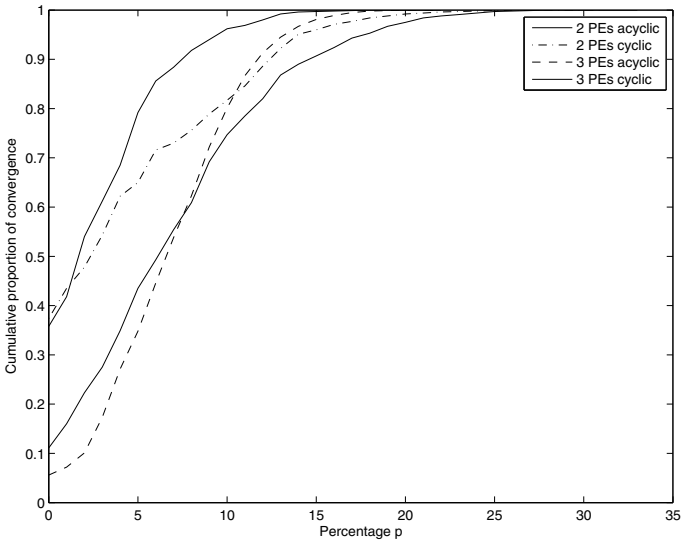


Fig. 7. Brute force vs SA+AT with setup B: Values plotted from Table VIII. 2 PEs acyclic converges fastest, 3 cyclic is the slowest.

search for setup B 3 PE cyclic graphs took $14.3M$ mappings, but SA+AT took 15 230 mappings on average, which means 99.9% reduction.

Limited target distribution (setup B) behaves quite differently w.r.t. optimization result. The optimal result is more likely achieved, e.g. 35.8% runs for 2 PEs and acyclic graphs. In contrast, the guarantees for optimality are weaker as the worst runs may have overhead of 18-28%. The number of iterations are comparable to setup A except for $p = 0\%$.

Larger architecture naturally makes mapping more difficult. This happens in two ways: the optimal result is less likely found with SA; probability drops by a factor of 3.4×-16.5×

TABLE IX
BRUTE FORCE VS. SA+AT WITH SETUP A: APPROXIMATE NUMBER OF MAPPINGS TO REACH GLOBAL OPTIMUM VALUE TIMES $1 + p$

	Number of mappings					
	2 PEs			3 PEs		
p	acyclic SA+AT	cyclic SA+AT	Brute force	acyclic SA+AT	cyclic SA+AT	Brute force
+0%	17 270	24 150	32 770	506 770	923 150	14.3M
+1%	7 220	9 350	...	102 530	128 970	...
+2%	2 600	2 950		19 640	34 390	
+3%	1 370	1 720		6 480	14 600	
+4%	970	1 220		3 720	7 610	
+5%	840	940		2 620	4 480	
+6%	790	840		2 110	2 960	
+7%	770	800		1 890	2 260	
+8%	750	790		1 800	1 920	
+9%	750	790		1 780	1 800	
+10%	$\leq$ 750	$\leq$ 790		1 770	1 760	
+11%	$\leq$ 750	$\leq$ 790		$\leq$ 1 770	1 760	
+12%	$\leq$ 750	$\leq$ 790		$\leq$ 1 770	1 750	

TABLE X
BRUTE FORCE VS. SA+AT WITH SETUP B: SEE TABLE IX

	Number of mappings					
	2 PEs			3 PEs		
p	acyclic SA+AT	cyclic SA+AT	Brute force	acyclic SA+AT	cyclic SA+AT	Brute force
+0%	2 220	2 200	32 770	29 030	15 230	14.3M
+1%	1 900	1 890	...	22 860	10 590	...
+2%	1 470	1 730		16 270	7 610	
+3%	1 300	1 520		9 450	6 170	
+4%	1 160	1 330		6 030	4 850	
+5%	1 000	1 270		4 700	3 890	
+6%	930	1 150		3 670	3 430	
+7%	900	1 130		3 040	3 060	
+8%	870	1 090		2 630	2 780	
+9%	850	1 040		2 270	2 450	
+10%	830	1 010		2 040	2 270	
+11%	820	980		1 890	2 160	
+12%	810	930		1 790	2 070	
+13%	800	890		1 730	1 950	
+14%	800	860		1 690	1 900	
+15%	$\leq$ 800	860		1 670	1 870	
...	...	...		...	...	
+18%		840		1 640	1 780	
+20%		830			1 740	
+25%					1 700	
+26%					1 700	
+28%					1 690	

and iteration count increases 6.9×-38.2×. The overhead p of the worst runs increases by few units: setup A from 9% to 12%, setup B from 26% to 28%.

A clear difference was found between KPN categories. Cyclic graphs are harder to map than acyclic. Also, lower target distribution is easier to map than high. We believe this is due to increased dependency of many nodes that increases the effect of a change in mapping.

The results indicate that SA+AT performs quite well; there is 91% probability to find mapping with $p \leq 15\%$, 75% probability to find mapping with $p \leq 10\%$ and 35% probability to find mapping with $p \leq 5\%$.

Also, note that mean and median mappings are almost constant due to parameterization. Also, saving a significant proportion of mapping iterations may only have a small effect on p. Unfortunately, there is no way to tell global optimum in termination condition without brute force search.

VI. CONCLUSIONS

Distributing Kahn Process Networks (KPNs) onto multiprocessor SoCs was analyzed. A Simulated annealing method to optimize process distribution was presented and validated. No significant loss of quality was found in solutions but over half the optimization time was saved.

Results were compared with global optimum solutions obtained from brute force search. It was found that near global optimum are frequently obtained, while in some cases as many as 37% of solutions reach global optimum. We are unaware of such results being published for KPNs. It is possible to save significant proportion of optimization effort while losing little in solution quality.

Future work should verify the method with real applications modeled as KPNs. Also, parameter generated KPNs should be studied to model performance characteristics of real applications to ease design space exploration.

REFERENCES

[1] 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.

[2] W. Wolf, *The future of multiprocessor systems-on-chips*, Design Automation Conference 2004, pp. 681-685, 2004.

[3] G. Kahn, *The semantics of a simple language for parallel programming*, Information Processing, pp. 471-475, 1974.

[4] Y.-K. Kwok and I. Ahmad, *Static scheduling algorithms for allocating directed task graphs to multiprocessors*, ACM Comput. Surv., Vol. 31, No. 4, pp. 406-471, 1999.

[5] T. Wild, W. Brunnbauer, J. Foag, and N. Pazos, *Mapping and scheduling for architecture exploration of networking SoCs*, Proc. 16th Int. Conference on VLSI Design, pp. 376-381, 2003.

[6] F. Glover, E. Taillard, D. de Werra, *A User's Guide to Tabu Search*, Annals of Operations Research, Vol. 21, pp. 3-28, 1993.

[7] T. D. Braun, H. J. Siegel, N. Beck, *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.

[8] H. Orsila, E. Salminen, T. D. Hämäläinen, Chapter in book "Simulated Annealing", ISBN 978-953-7619-07-7, *Best Practices for Simulated Annealing in Multiprocessor Task Distribution Problems*, I-Tech Education and Publishing KG, pp. 321-342, 2008.

[9] H. Orsila, T. Kangas, E. Salminen, T. D. Hämäläinen, *Parameterizing Simulated Annealing for Distributing Task Graphs on multiprocessor SoCs*, Symposium on SoC, Nov 14-16, pp. 73-76, 2006.

[10] 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, Elsevier, 2007.

[11] 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*, Symposium on SoC, 2007.

[12] H. Orsila, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, *Evaluation of Heterogeneous Multiprocessor Architectures by Energy and Performance Optimization*, Symposium on SoC, Nov 4-6, 2008.

[13] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Optimization by simulated annealing*, Science, Vol. 200, No. 4598, pp. 671-680, 1983.

[14] B.W. Kernighan, S. Lin, *An Efficient Heuristics Procedure for Partitioning Graphs*, The Bell System Technical Journal, Vol. 49, No. 2, pp. 291-307, 1970.

[15] *kpn-generator*: http://zakalwe.fi/~shd/foss/kpn-generator/

[16] R. Thid, I. Sander, A. Jantsch, *Flexible bus and noc performance analysis with configurable synthetic workloads*, DSD, pp. 681-688, 2006.

[17] *jobqueue*: http://zakalwe.fi/~shd/foss/jobqueue/