Optimal Subset Mapping And Convergence Evaluation of Mapping Algorithms for Distributing Task Graphs on Multiprocessor SoC

Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen
Institute of Digital and Computer Systems
Tampere University of Technology
P.O. Box 553, 33101 Tampere, Finland
Email: {heikki.orsila, erno.salminen, marko.hannikainen, 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. Applications are modeled as static acyclic task graphs which are mapped to an MPSoC. The analysis is based on extensive simulations with 300 node graphs from the Standard Graph Set.**

We present a new algorithm, Optimal Subset Mapping (OSM), that rapidly evaluates task distribution mapping space, and then compare it to simulated annealing (SA) and group migration (GM) algorithms. OSM was developed to make architecture exploration faster. Efficiency of OSM is $5.0\times$ and $2.4\times$ than that of GM and SA, respectively, when efficiency is measured as the application speedup divided by the number of iterations needed for optimization. This saves 81% and 62% in wall clock optimization time, respectively. However, this is a trade-off because OSM reaches 96% and 89% application speedup compared to GM and SA. Results show that OSM and GM have opposite convergence behavior and SA comes between these two.

I. Introduction

An efficient MPSoC implementation requires exploration to find an optimized architecture as well as mapping and scheduling of the application. A large design space must be pruned systematically, since the exploration of the whole design space is not feasible. However, fast optimization procedure is desired in order to cover reasonable design space. Iterative algorithms evaluate a number of application mappings for each resource allocation candidate. For each mapping, an application schedule is determined to evaluate the cost. The cost function may consider multiple parameters, such as execution time, communication time, memory, energy consumption and silicon area constraints.

SoC applications can be modeled as acyclic static task graphs (STGs) [1]. Nodes of the STG are finite, deterministic computational tasks, and edges denote dependencies between the nodes. Node weights represent the amount of computation associated with a node. Edge weights model the amount of communication needed to transfer results between the nodes. Computational nodes block until their data dependencies are resolved, i.e. when they have all needed data. Details of the task graph parallelization system employed in this paper can be found in [2].

This paper presents a new mapping algorithm called Optimal Subset Mapping (*OSM*) to speedup architecture exploration. It is compared to two variants of Simulated Annealing algorithm (*SA* and *SA+AT*) [2][3], Group Migration (*GM*), and their combination (*Hybrid*) [4]. Furthermore, a random algorithm is used as basis for comparison. The system has 2, 4, or 8 identical processing elements (PEs). Ten random 300-node STGs are selected from [5].

II. Related Work

Braun *et al.* [6] compared 11 optimization algorithms for distribution of tasks without data dependencies. 512 tasks were parallelized onto 16 machines and total execution time was measured for each heuristics. Our system schedules 300 tasks with data dependencies for 8 PEs. It must also be noted that our tasks have 10 times more edges than tasks in [6]. They do approximately $3\ 000$ mappings for each SA run, which can be shown to be far too few for 16 machines and 512 tasks in the case that tasks are dependent [2]. Our SA implementation does approximately $200\ 000$ mappings for a single run with dependent tasks. Their paper does not present convergence properties of SA as a function of iterations, and we are not aware of related work that measures SA convergence for task mapping with respect to iterations and the number of PEs.

Our earlier work [3] presented a heuristics for automatically selecting temperature schedule for SA to speedup convergence of dependent tasks. Also, [2] presented heuristics to select total iteration number for reasonable efficiency for SA with dependent tasks. This paper will merge those results and compare them to various mapping algorithms, including the new OSM.

III. Studied algorithms

The mapping algorithms can be classified as follows. First, is the algorithm deterministic (same results on every independent run) or probabilistic (result varies between runs). Second, does algorithm accept a move to worse state along the run (non-greedy) or only better moves (greedy). Hence, 4 categories can be identified.

Architecture exploration needs an automatic selection of optimization parameters depending on the architecture and application sizes. Otherwise, an algorithm may spend excessive time optimizing a small systems or result in a sub-optimal solution for a large system. The goal is to avoid unnecessary optimization iterations, while keeping application performance close to architecture limits. This will save optimization time and thus speed up architecture exploration.

The term *move* means here the change of the location (PE) of one or multiple tasks. All studied algorithms (except random) ensure that move is always made to a different PE, which saves many iterations. This is a crucial detail forgotten in many papers. For example, randomizing a single task for 2 PEs will result in 50% of iterations being useless because the task is not actually moved anywhere.

STGs are used because there exists well known efficient and near optimal scheduling algorithms for them. This ensures that the observed differences are due to mapping. Harder scheduling properties would diminish accuracy of mapping analysis. All presented algorithms are agnostic of STG structure, and so they will also work with general process networks like Kahn Process Networks (KPN) [7]. These algorithms are also used in our Koski flow, that has a KPN-like process network [8]. Koski is a high-level design tool for multiprocessor SoCs and applications.

Details of the used algorithms can be found in [2][3][4] but use of these algorithms is presented next. The new OSM algorithm is introduced in detail.

For each algorithm, tasks are initially mapped to one PE.

A. Group Migration (GM)

Group migration (GM), also known as Kernighan-Lin graph partitioning algorithm [9], is a deterministic algorithm that moves one task at time and finds an optimal mapping for that. It accepts only moves to a better state (one with smaller cost). Therefore, it is greedy algorithm and may get stuck to a local minimum. This happens when there is no single move that improves (decreases) the cost, and GM terminates. This algorithms does not need any parameters. The exact algorithm used here is presented in [2]. The worst case iteration count is in $O((M-1)N^2)$, where M is the number of PEs and N the number of tasks. A starting point near a local optimum will converge much more rapidly.

B. Variants of Simulated Annealing (SA)

SA is a probabilistic non-greedy algorithm [10] that explores search space of a problem by annealing from a high to a low temperature state. This paper uses two versions of SA that are presented in [2] and [3]. Algorithm performs random changes in mapping with respect to the current mapping state.

SA algorithm always accepts a move into a better state, but also into a worse state with a probability that decreases along with the temperature. Thus the algorithm becomes greedier on low temperatures. The acceptance probability function and the number of iterations per temperature level is set by the method presented in [2]. The annealing schedule function, initial temperature T_0 and the final temperature T_f are selected by the method in [3]. The algorithm terminates when the final temperature is reached and sufficient number of consecutive moves have been rejected.

The basic version of simulated annealing is referred here as *SA* and one with automatic temperature selection as *SA+AT*. In general, SA+AT achieves nearly the same performance as SA but in considerably fewer iterations. The Hybrid algorithm [4] uses SA for initial optimization and finishes the mapping with GM. The parameters for SA variants are temperature range (initial and final), number of temperature levels, scaling between levels, and number of iteration on each level. Furthermore, move heuristic, acceptance function, and end condition must be defined.

The total number of iterations for SA is

$$(\frac{\log \frac{T_f}{T_0}}{\log q} + 1)N(M-1), \tag{1}$$

where q is the temperature scaling factor [2].

C. Random

A simple random mapping algorithm is included just to obtain basis for algorithm comparison. The algorithm tries random mappings without regarding the results from previous iterations, hence it is probabilistic and non-greedy. The only parameter is the number of random mappings.

IV. Optimal Subset Mapping (OSM)

The new Optimal Subset Mapping (OSM) algorithm takes a random subset of tasks and finds the optimum mapping in that subset by trying all possible mappings (brute-force search). This is called a round. Tasks outside the subset are left in place. OSM is probabilistic because it chooses a subset randomly at every round. It is also greedy because it only accepts an improvement to the best known solution at each round. OSM algorithm was inspired by Sequential Minimal Optimization (SMO) algorithm [11]. SMO is used for solving a quadratic programming problem and has a static subset size of 2, but the subset size in OSM is dynamic during run-time. Also, the total number of iterations in OSM is bounded by task graph and architecture characteristics.

The pseudo-code of OSM is shown in Fig. 1. Variable S denotes the current (mapping) state, C is the cost, X is subset size, and R is a round number used to track progress of the algorithm. Function **Cost**(S) evaluates the cost function for the mapping state S and minimum S is sought. Function **Pick_Random_Subset**(S, X) picks a random subset of X separate tasks from mapping S. Function **Apply_Mapping**(S, S_{sub}) takes whole mapping S and subset mapping S_{sub}. It copies mappings from S_{sub} to S.

Initially, the subset size $X = 2$. If no improvement has been found within last $R_{max} = \lceil \frac{N}{X_{max}} \rceil$ rounds, the subset size X is increased by 1. If there was some improvement, X is decreased by 1. The subset size X is bounded to $[X_{min}, X_{max}]$, where $X_{min} = 2$. The algorithm terminates

```
OPTIMAL_SUBSET_MAPPING($S$)
 1  $S_{best} \leftarrow S$
 2  $C_{best} \leftarrow$ COST($S$)
 3  $X \leftarrow 2$
 4  for $R \leftarrow 1$ to $\infty$
 5  do $C_{old\_best} \leftarrow C_{best}$
 6      $S \leftarrow S_{best}$
 7      $Subset \leftarrow$ PICK_RANDOM_SUBSET($S, X$)
 8      for all possible $mappings\ S_{sub}$ in $Subset$
 9      do $S \leftarrow$ APPLY_MAPPING($S, S_{sub}$)
10          $C \leftarrow$ COST($S$)
11          if $C < C_{best}$
12              then $S_{best} \leftarrow S$
13                   $C_{best} \leftarrow C$
14      if $modulo(R, R_{max}) = 0$
15         then if $C_{best} = C_{old\_best}$
16                 then if $X = X_{max}$
17                         then break
18                      $X \leftarrow X + 1$
19                 else $X \leftarrow X - 1$
20              $X \leftarrow$ MAX($X_{min}, X$)
21              $X \leftarrow$ MIN($X_{max}, X$)
22  return $S_{best}$
```

Fig. 1. Pseudo-code of Optimal Subset mapping algorithm.

when none of the last R_{max} rounds improved the solution and maximum subset size is reached ($X = X_{max}$).

Upper bound for subset size X is needed to limit the number of iterations. It can be derived as

$$M^X = cN^{c_N}M^{c_M}, \quad (2)$$

where N is the number of tasks and M is the number of PEs. c, c_N and c_M are arbitrary positive coefficients used to limit iterations with respect to system size defined by N and M. It is recommended that $c_N, c_M \geq 1$ to reach acceptably good results. Solution to (2) is

$$X_{max} = \lfloor \frac{\log(c) + c_N \log(N) + c_M \log(M)}{\log M} \rfloor. \quad (3)$$

As a consequence, the number of iterations increases as N and M increase. The total number of mappings for R_{max} rounds is in

$$O(\frac{N^{1+c_N}M^{c_M}}{\log N + \log M}). \quad (4)$$

V. Experiment setup

The experiment uses 10 random graphs with 300 nodes from the Standard Task Graph set [5]. The communication weights were generated randomly from uniform distribution. The resulting *communication-to-computation* ratios varied between graphs. The minimum, average and maximum byte/s for tasks in graphs are 8.1 Mbyte/s, 217.8 Mbyte/s, 525.6 Mbyte/s. This is the rate at which tasks will produce data in these graphs. Random graphs are used to evaluate optimization algorithms as fairly as possible. Non-random applications may well be relevant for common applications, but they are dangerously biased for general algorithm comparison. Investigating algorithm bias and classifying computational tasks based on the bias are outside the scope of this paper. Random graphs have the property to be neutral of the application. The task graphs are summarized in Table I along with HW platform and measurement setup.

TABLE I
Application and architecture parameters for the experiment

	Variable	(note)	Value
Task graphs	# graphs		10
	# tasks per graph (N)		302
	# edges per graph	(1)	1594, 5231, 8703
	comp time per task [us]	(1)	3.2, 5.1, 7.0
	comm vol per task [byte]	(1)	26, 1111, 3679
	comm/comp -ratio [Mbyte/s]	(1)	8, 218, 526
	max theor. parallelism [no unit]	(1)	4.3, 7.9, 12.8
HW Platform	# PEs (M)		2, 4, 8
	PE freq [MHz]		50
	Bus Freq [MHz]	(2)	10, 20, 40
	Bus width [bits]		32
	Bus bandwidth [Mb/s]	(2)	320, 640, 1280
	Bus arb. latency [cycles/send]		8
Algorithms	# runs per graph per alg	(3)	10
	algorithms		6
	determ, non-greedy		1: OSM
	determ, greedy		1: GM
	stoch., non-greedy		4: SA, SA+AT, hybrid, random
	stoch, greedy		-

Notes:
(1) = min, avg, max
(2) = values for 2,4,8 PEs, respectively
(3) = only 1 run for GM

The SoC platform is a message passing system where each PE has some local memory, but no shared memory. Each graph was distributed onto 2, 4 and 8 identical PEs. The PEs are interconnected with a single, dynamically arbitrated shared bus that limits the SoC performance due to bus contention. Bus frequency is low in order to highlight the differences between algorithms when HW resources are very limited. However, bus frequency is scaled with system size, as shown.

Total of 6 algorithms are compared. Optimization was run 10 times independently for each task graph, except with GM that needs only 1 run due to its deterministic behavior. The optimization software was written in C language and executed on a 10-machine Gentoo Linux cluster, each machine having a single 2.8 GHz Intel Pentium 4 processor and 1 GiB of memory. A total of $2 \cdot 10^9$ mappings were tried in 1869 computation hours (78 days) leading to average $297 \frac{mappings}{s}$.

The optimization parameters of the experiment are shown in Table II.

TABLE II

OPTIMIZATION PARAMETERS FOR THE EXPERIMENT

Alg.	Variable	(note)	Value
SA, SA+AT, Hybrid	# iter per T, ($L = N \cdot (M\text{-}1)$)	(1)	602, 1208, 2416
	# temperature levels		181
	# temperature scaling		q=0.95
	range of T (SA and hybrid)	(2)	T_0= 1.0, T_f=0.0001
	range of T (SA+AT)		T range coefficient k=2
	annealing schedule (T_0, i)		$T_0 \cdot q^{floor(i/L)}$
	move heuristic		move 1 random task
	acceptance function		$(1 + \exp(\Delta C / (0.5\, C_0\, T))^{-1}$
	end condition		$T = T_f$ AND L rejected moves
Rand	# max interations		262 144
GM	no params needed		-
OSM	coefficient c		1.0
	exponent c_N		1.0
	exponent c_M		1.0
	subset size X [#tasks]	(1)	9, 5, 3
	# iterations per subset	(1)	512, 1024, 512

Notes:

(1) = values for 2,4,8 PEs, respectively

(2) = T_0 and T_f computed automatically in SA+AT

VI. RESULTS

For simplicity, the cost function considers only the execution time. Hence, gain equals speedup and speedup is defined as $\frac{t_1}{t_M}$, where t_i is the graph execution time on i PEs. The results are discussed according to average gain, progress of gain with respect to required iterations, and differences between graphs.

A. Gain

Figure 2 shows averaged speedups for each algorithm. Random mapping is clearly the worst algorithm and the difference between it and others grows with the number of PEs. Other 5 algorithms have almost equal performance, Hybrid being marginally better than others and GM and OSM slightly worse than others. SA, SA+AT and Hybrid are only marginally different in gain, from 0.01 to 0.04 gain units difference.

The average speedup grows with system size. For 2 PEs, total PE utilization varied from 77% to 99.7%. For 4 PEs, from 52% to 76%. And, for 8 PEs, from 37% to 51%. Interconnect utilization was nearly 100% as parallelization is communication bounded. Therefore, the gains are clearly lower than theoretical maximum parallelism. Average theoretical maximum parallelism is 7.9 for these graphs. It is defined by dividing the sum of computation times by the computation time of the critical path and neglecting the communication costs.

Variance in gain values is small, but there is a notable difference in the number of iterations that algorithms require during optimization. This will be analyzed next.

B. Time behavior of algorithms

Figure 3 shows the averaged speedups with respect to number of iterations for 8 PE system. The results with 2 and 4 PEs are similar but omitted for brevity. Note that

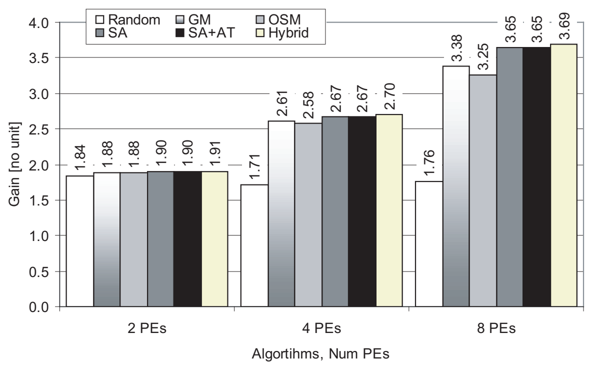


Fig. 2. Achieved gain averaged over 10 STGs.

the time for running an optimization algorithm is directly proportional to the number of iterations when graph size N and the architecture size M are fixed. Iterations are shown on a logarithmic scale and the first $1\ 000$ evaluations are omitted from the figure.

GM needs many iterations to achieve any speedup but once that occurs the speedup increases very rapidly. A total opposite is the new OSM algorithm. It reaches almost the maximum speedup level with very limited number of iterations. SA, SA+AT and Hybrid lie between these extremes, and they achieve the highest overall speedup.

Random mapping saturates quickly and further iterations are unable to provide any speedup.

Hybrid algorithm converges very slowly due to a simple but inefficient temperature schedule. But once it is in the right temperature range ($200\ 000$ iterations), it converges up very rapidly. SA+AT has an optimized temperature range that starts rapid convergence already at $20\ 000$ iterations and reaches the maximum before the Hybrid starts converging. The Hybrid algorithm does many independent annealings in different temperature ranges, and also uses group migration, and thus reaches a slightly higher maximum than SA+AT. If normal SA were plotted on Figure 3, it would follow Hybrid algorithm exactly till $380\ 000$ iterations, because Hybrid algorithm begins with a normal SA.

C. Trade-off between gain and required iterations

Clearly, algorithms proceed at different speeds, i.e. gain increases with varying slope. The average slope is defined as

$$average\_gain\_slope = \frac{gain}{\#iterations}.$$

It defines how much the gain increases with one iteration on average.

Figure 4 shows the average gain slope values of algorithms relative to that of random mapping. Random mapping algorithm was chosen to be a reference to measure ease of parallelization. The slopes of GM and Hybrid decrease rapidly with system size, i.e. they spend rapidly increasing time with larger systems. Considering this trade-off between optimization result and time needed, OSM and SA+AT are the best algorithms.

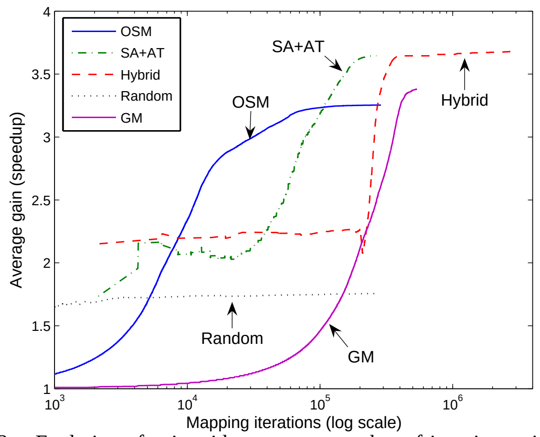


Fig. 3. Evolution of gain with respect to number of iterations with 8 PEs.

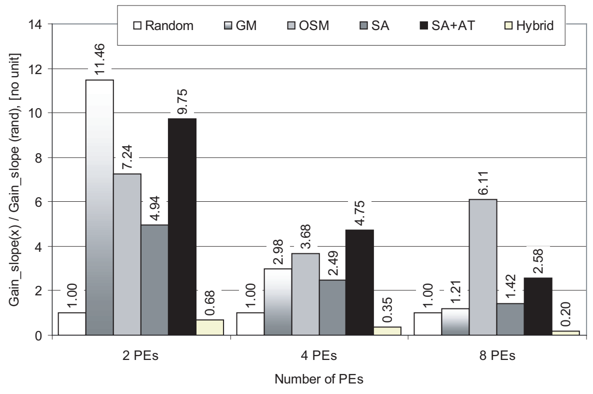


Fig. 4. The best gain divided by the number of iterations. Values are relative to random mapping.

TABLE III
ROUNDS AND MAPPING ITERATIONS FOR OSM

PEs	rounds (min, avg, max)	Thousands of iterations (min, avg, max)
2	271, 380, 611	34.1, 37.2, 73.6
4	239, 469, 899	80.6, 115.4, 259.1
8	199, 428, 1099	57.1, 88.8, 293.9

For 4 PEs, OSM algorithm is 23%, 48%, −23% and 951% more efficient than GM, SA, SA+AT and Hybrid, respectively. Efficiency is defined as achieved gain divided by the number of mapping iterations needed. The save in wall clock optimization time are 20%, 35%, −25% and 91%, respectively. This makes SA+AT a clear winner because it is only 0.03 gain units slower than Hybrid, but noticeably the fastest.

For 8 PEs, OSM algorithm is 405%, 330%, 137% and 2955% more efficient in average gain slope compared to GM, SA, SA+AT and Hybrid, respectively. The save in wall clock optimization time are 81%, 79%, 62% and 97%, respectively. That is, OSM is very efficient. However, SA+AT has a 12% or 0.40 units higher gain number than OSM, which makes SA+AT a very good candidate for the 8 PE case.

Table III shows rounds and iterations for OSM algorithm. The average number of rounds varies from 380 to 469 due to its parameter selection scheme, and the average number of iterations scales up with the number of processors.

D. *Differences between the graphs*

There are differences in obtained gain depending on the graph. The progress of OSM, SA+AT and GM for each graph is shown in Figure 5. Each line represents different graph. Results are for 8 PEs. OSM starts saturating always at the same point, after 10^4 iterations, for every graph, as illustrated in Figure 5(a). However, the gains differ at most $+50\%$ (gain of 3.7 vs 2.5). Both GM and SA+AT had similar difference between the best and worst case graphs. SA+AT achieves the largest speedup among algorithms at the beginning (iterations 1 - 1 000) due to its random mapping style in the beginning of annealing. Consequently, the differences between graphs become visible already at small iteration counts whereas they are observed only at the end with OSM and GM. The difference in iteration counts between graphs varied at most by factor of 3 for OSM, due to its subset size changing policy. Other algorithms were varied much less than that.

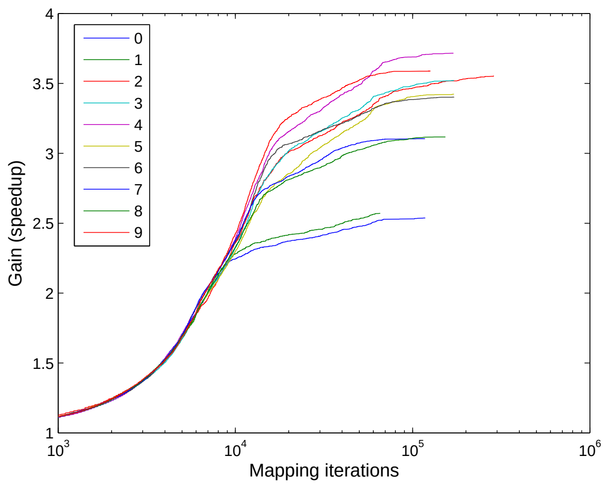
It turned out that the same graphs performed worse with every algorithm. The studied graphs varied in terms of connectivity and branching (number of edges) and computation amount. We carefully analyzed the correlation between the static properties of task graphs and achieved gain. However, no causal relation was found. For example, the two worst graphs had many edges (7109 and 8703) but the best had also many (7515). It is our optimistic hope that the presented results therefore present the "general" case as well.

E. *Discussion*

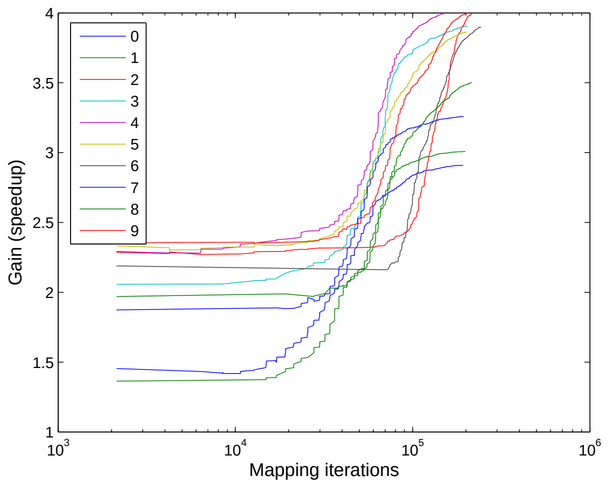
The Hybrid algorithm reaches the best speedup, but it is only marginally better than other SA variants. GM and OSM are clearly worse in 8 PE case, but do almost as well in 2 and 4 PE cases. This shows that Hybrid and SA variants are more scalable than OSM and GM. However, in terms of average gain slope, OSM and SA+AT the most scalable algorithms (see Section VI-C).

Hybrid and SA converge so slowly that they are useless for large scale architecture exploration. SA+AT is as good as SA in speedup but converges much more rapidly due to its parameter selection method. GM converges slowly compared to SA variants. OSM converges very rapidly, and therefore we suggest to use it early in the exploration. However, its final result is not as high as SA, which possibly means that another algorithm should continue after OSM, or OSM should be improved.

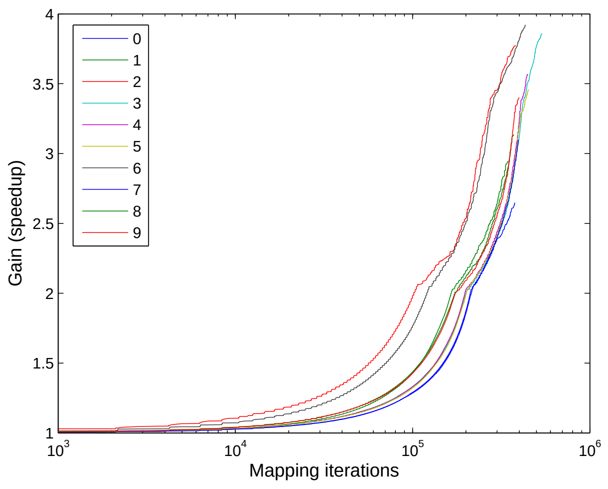
Hybrid and SA variants are insensitive to initial values due to their random nature in high temperatures. GM is highly sensitive to initial values due to its deterministic and greedy nature, and therefore we advice against using it without independent runs from different initial values. Effect of initial values to OSM is an open question, but it is reasonable to assume it depends on the maximum subset size and graph structure.



(a) OSM progress



(b) SA+AT progress



(c) GM progress

Fig. 5. OSM, SA+AT and GM progress plotted for each graph.

VII. Conclusion

This paper presented a new mapping algorithm, Optimal Subset Mapping (OSM), and it is compared to 5 other algorithms. OSM was developed to make architecture exploration faster. The results show large differences on the number or required iterations during the optimization so that OSM is a strong candidate for a rapid mapping algorithm when the number of iterations is taken into account. Simulated annealing with automatic temperature selection (SA+AT) gives nearly the best gain with reasonable effort, but OSM is faster in convergence. When only the speedup is measured, differences are small among algorithms.

Also, the paper presented convergence properties of 5 algorithms with respect to iteration number, number of processors and different random graphs. Convergence properties of OSM and GM have opposite behavior and SA comes between these two. The convergence figures presented in this paper should help architecture explorers choose a suitable algorithm for task mapping. OSM and SA+AT are the recommended choices of these algorithms.

Future research should try to integrate rapid convergence properties of OSM to other algorithms, create a non-greedy version of OSM to have similar advantages as SA+AT, increase the granularity of subsets of tasks (map several subsets of tasks optimally, instead of mapping a subset of tasks optimally), and analyze the relation between graph properties and gain.

References

[1] 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.

[2] 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, 2007, Elsevier, In print.

[3] 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 (SoC 2006), Tampere, Finland, November 14-16, 2006, pp. 73-76.

[4] H. Orsila, T. Kangas, T. D. Hämäläinen, *Hybrid Algorithm for Mapping Static Task Graphs on Multiprocessor SoCs*, International Symposium on System-on-Chip (SoC 2005), pp. 146-150, 2005.

[5] *Standard task graph set*, [online]: http://www.kasahara.elec.waseda.ac.jp/schedule, 2003.

[6] 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.

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

[8] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, J. Riihimäki, K. Kuusilinna, *UML-based Multi-Processor SoC Design Framework*, Transactions on Embedded Computing Systems, ACM, 2006.

[9] 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.

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

[11] J. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*, Microsoft Research Technical Report MSR-TR-98-14, 1998.