Evaluation of Heterogeneous Multiprocessor Architectures by Energy and Performance Optimization

Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen
Department of 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*— Design space exploration aims to find an energy-efficient architecture with high performance. A trade-off is needed between these goals, and the optimization effort should also be minimized. In this paper, we evaluate heterogeneous multiprocessor architectures by optimizing both energy and performance for applications. Ten random task graphs are optimized for each architecture, and evaluated with simulations. The energy versus performance trade-off is analyzed by looking at Pareto optimal solutions. It is assumed that there is a variety of processing elements whose number, frequency and microarchitecture can be modified for exploration purposes. It is found that both energy-efficient and well performing solutions exist, and in general, performance is traded for energy-efficiency. Results indicate that automated exploration tools are needed when the complexity of the mapping problem grows, starting already with our experiment setup: 6 types of PEs to select from, and the system consists of 2 to 5 PEs. Our results indicate that our Simulated Annealing method can be used for energy optimization with heterogeneous architectures, in addition to performance optimization with homogeneous architectures.**

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 multiprocessing SoCs is to increase performance while decreasing energy consumption. This is achieved by efficient communication between cores and keeping clock frequency low.

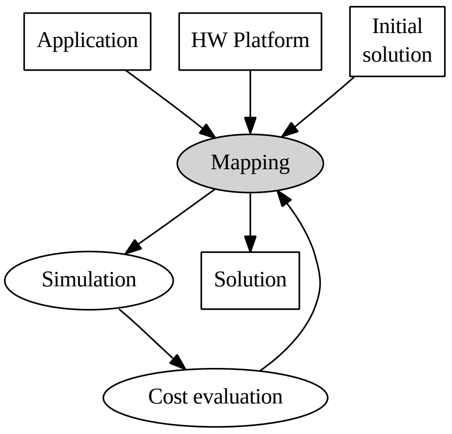
Mapping means placing each application component to some processing element (PE). Scheduling means determining execution order of the application components on the platform. A large design space must be pruned systematically, since the exploration of the whole design space is not feasible [1]. Fast optimization 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. For each mapping, the application is scheduled and simulated to evaluate the cost of the solution, i.e. the value of the objective function. The objective function may consider multiple parameters, such as execution time, communication time, memory, energy consumption and silicon area constraints. Figure 1(a) shows the mapping process.

We present an experiment where a set of hardware architectures is generated by random, and applications are mapped on them. Hardware architectures are 2 to 5 PE systems with both singlescalar and superscalar PEs with frequencies from 100MHz to 300MHz. The total area of the system is limited to $8mm^2$. Applications are 300 node acyclic static task graphs (STGs) [3]. Figure 1(b) shows the application, its mapping, and the hardware platform. The application is optimized for each architecture with respect to the energy for that is consumed when the application is run. Resulting energy and execution time values for each architecture are analyzed to find Pareto optimal architectures. Hence, applications are optimized with a single objective (energy), but architectures are analyzed by two objectives.

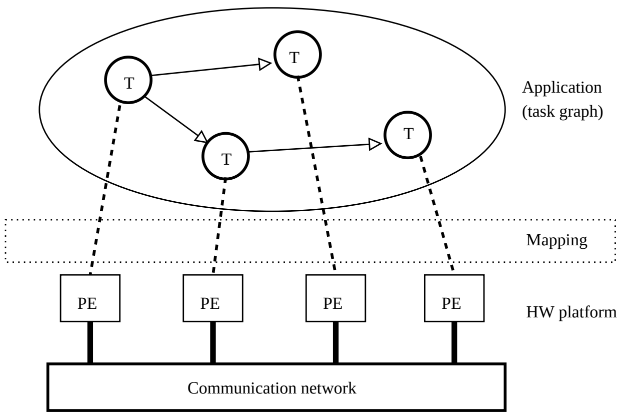
With the constraints of our experiment, the results show there is a clear trade-off for energy and performance. Low number of PEs means weak performance, but low power. High number of PEs means more performance, but loses energy-efficiency. Also, increasing number of PEs creates demand for automated exploration tools, as the mapping problem becomes more important and increasingly harder.

II. Related Work

Our earlier work [4] evaluated various mapping algorithms to determine optimization convergence rate when application performance was maximized. This paper adds heterogeneous PEs to the problem. Simulated annealing (SA) was found to be an efficient algorithm, and therefore, it is used also in this paper. SA is a probabilistic non-greedy algorithm [5] that explores search space of a problem by annealing from a high to a low temperature state. These methods are also used in our Koski flow [6]. Koski is a high-level design tool for multiprocessor SoCs and applications. Koski utilizes Kahn Process Networks [7] for application modeling.



(a) Optimization process. Boxes indicate data. Ellipses indicate operations. This chapter focuses on the mapping part.



(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

III. Experiment Setup

A. Objective And Optimization Algorithm

An experiment was done to investigate trade-offs between energy and application performance. The objective function (1) is to minimize the total energy consumption (static + dynamic). The energy is measured in relative values. It is not a physical energy unit.

$$E = T(P_s + P_d) = T(\sum_{i=1}^{N} A_i f_{max} + k \sum_{i=1}^{N} A_i f_i U_i) \quad (1)$$

where T is the execution time of the application, determined in simulation. N is the number of PEs, A_i is the area of PE i and f_i is its frequency. f_{max} is the maximum frequency of any PE or interconnect, which is at least 200MHz in this experiment because the bus operates at 200MHz. Utilization U_i is the proportion of non-idle cycles of the PE i. The HW architecture defines values A_i and f_i whereas the mapping indirectly defines T and values U_i.

Coefficient k is the factor that changes the relative proportion of static versus dynamic energy. Energy values are comparable when k value is constant. The effect of dynamic power can be eliminated by setting $k = 0$. The static power part P_s of the objective function depends on the number of transistors (relative to A) and their speed (relative to f_{max}). The dynamic power part P_d depends on total capacitance (relative to A), switching frequency f_i, and activity U_i. Supply voltage is assumed fixed.

Simulated annealing algorithm was used to optimize energy (1) by changing application mappings. The algorithm is specified in [8], but modified in two ways. First, the objective function is the application energy on a given platform. Second, the algorithm is run twice for each solution, and the second run always starts from the best solution of the first run. This was done to increase confidence in results, as the SA is stochastic. SA temperature margin value 2 was used to scale initial and final temperatures [4].

B. Simulated HW Platform

The simulated SoC platform for the experiment is a message passing system where each PE has some local memory, but no shared memory. The system is simulated on the behavior level. Each PE and interconnection resource is available for a single action at a time. PE task context switch overhead is 0 cycles, but bus arbitration time is 8 cycles for each transfer.

Figure 1(b) presents simulated HW platform. PEs are interconnected with two shared buses that are independently and dynamically arbitrated. The shared buses limit SoC performance due to contention, latency and throughput. Bus frequency is 200MHz for both buses and they are 32 bits wide. The bus silicon area is $0.1mm^2$ per processor. Each node in the task graph sends a specific number of bytes after computation, thus creating contention on the shared buses. Which ever bus is free at a time is used for communication by using FIFO arbitration, i.e. which ever PE comes first gets the bus.

Table I shows different types of PEs, each presented with a letter. Multiprocessor architectures were varied using these PEs. An architecture consists of 2 to 5 PEs, and the total area has $8mm^2$ upper-bound. Architectures are presented with fingerprint codes from these letters. Parameter p is the average number of instructions per cycle. Frequency f has 3 values: 100MHz, 200MHz and 300MHz. Processor speed is measured in millions of operations per second, which equals $p * f$. A is the area in square millimeters. Each PE can be implemented as a singlescalar or as a superscalar version. The superscalar version can execute $p = 1.8$ instructions per clock. The singlescalar version has area $A = 1mm^2$, superscalar has $A = 2mm^2$. 2 values of p and 3 values of f implies 6 different PEs. A task graph node of n cycles can be computed in $\frac{n}{fp}$ time.

C. Architecture Fingerprinting

Architecture fingerprinting is used to present results. An architecture is characterized by a series of letters from A to F. Letters are labels for different PEs specified in Table I. Letters are assigned in the order of increasing number of operations per second. Letter A is assigned for the slowest PE, and letter F

TABLE I

AVAILABLE PROCESSOR TYPES

PE type	f (MHz)	p ($\frac{Ops}{cycle}$)	Speed ($\frac{MOps}{s}$)	A (mm^2)
A	100	1.0	100	1
B	100	1.8	180	2
C	200	1.0	200	1
D	300	1.0	300	1
E	200	1.8	360	2
F	300	1.8	540	2

TABLE II

PROPORTION OF HOW MANY TIMES A GIVEN NUMBER AND TYPE OF PE WAS IN THE EXPERIMENT'S 141 FINGERPRINTS. TABLE VALUES ARE EXPLAINED IN SECTION III-D.

PE type	1 PE (%)	2 PEs (%)	3 PEs (%)	4 PEs (%)	5 PEs (%)	PE prop. (%)
A	28.4	10.6	2.8	0.7	0.7	43.3
B	28.4	7.1	1.4	0.0	0.0	36.9
C	30.5	9.2	4.3	1.4	0.7	46.1
D	31.9	8.5	5.7	1.4	0.7	48.2
E	31.9	10.6	2.1	0.0	0.0	44.7
F	29.1	9.2	1.4	0.0	0.0	39.7
Any	0.0	14.2	27.7	33.3	24.8	

is assigned for the fastest PE. Each PE in the architecture gets a single letter in the architecture fingerprint. The letters are organized into alphabetical order to facilitate human brain's pattern recognition, i.e. make it easier to see slow, fast and same type of PEs. For example, AAB fingerprint means a three PE architecture with two PEs of type A and one PE of type B. The two PEs of type A are in the beginning of the series to display that there are exactly two instances of A in the architecture and that they are the slowest PEs.

The architecture fingerprint can be extended for heterogeneous interconnections by separating PE selections and interconnection selections with a dash (-). However, the interconnection is the same for all architectures in this paper, and therefore, it is omitted.

D. Random Architectures

141 different architectures were generated. Homogeneous architectures, the architectures with only one type of PE, were inserted manually, and the rest were generated randomly. The total area for each architecture was limited to $8mm^2$. Table II shows the proportion of how many times a given number and type of PE was in all the fingerprints. Rows indicate PE types, and columns indicate proportion of PEs. The last row shows the proportion of architectures with a given number of PEs. The last column shows the proportion of architectures that had at least one PE of that row's type. For example, row A's third column value means that 10.6% of architectures had exactly 2 PEs of type A. Last row's third column value means that 14.2% of the fingerprints had exactly 2 PEs. Row A's last column shows that $0.433 * 141 = 61$ architectures had at least one A type PE. The table has zeroes due to area constraints. For example, an architecture with 4 B type PEs does not exist, because one B type PE takes $2mm^2$, and its associated interconnect area is $0.1mm^2$. Therefore, the total area is $4 * (2 + 0.1)mm^2 > 8mm^2$.

TABLE III

ATTRIBUTES AND LIMITS OF THE EXPERIMENT

Attribute	Values
Number of architectures	141
Maximum architecture area	$8mm^2$
Number of PEs in each architecture	2 to 5
Number of 300 node graphs	10
Objective function to optimize energy	$T(\sum A_i f_{max} + k \sum A_i f_i U_i)$, where i is from 1 to N PEs
k values	0, 1, 4
Optimization algorithm	Simulated annealing

E. Applications

The experiment uses ten random task graphs with 300 nodes from the Standard Task Graph set [9]. Random graphs are used to avoid bias in algorithms and results. 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. The edge weights were generated randomly from uniform distribution.

STGs are used because there exists well known efficient and near optimal scheduling algorithms for them [3]. This ensures that the observed differences in optimization results are due to mapping, not scheduling. More complex scheduling properties would diminish accuracy of mapping analysis. However, this experiment is agnostic of the STG structure, and so it could be done with general process networks like Kahn Process Networks (KPN).

F. Experiment Data

Table III shows attributes that were varied in the experiment. Each of the 10 task graphs was optimized and simulated against each architecture. This was done for three values of $k = 0, 1$ or 4 to change static versus dynamic energy balance. Thus, total of $10 * 141 * 3 = 4230$ simulations was run.

G. Software

The optimization software and simulator was written in C language and executed on a 9 machine Red Hat Enterprise Linux WS release 3 cluster, each machine having a single 2.8 GHz Intel Pentium 4 processor and 1 GiB of memory. Jobs were distributed to a cluster with *jobqueue* [10] (version control snapshot *2008-05-30*) by using OpenSSH [11] and rsync [12]. No special clustering software or configurations was used. A total of $8.48 \cdot 10^8$ mappings was evaluated in optimization in 27.4 computation days leading to average of $358 \frac{mappings}{s}$. Rapid mapping evaluation is a benefit of STGs.

IV. RESULTS

Figure 2 plots energy versus execution time for each architecture for $k = 1$ that emphasizes static energy. Figure 3 plots the same data for $k = 4$, i.e. bigger weight on dynamic energy. Energy values are summed and time values are summed for

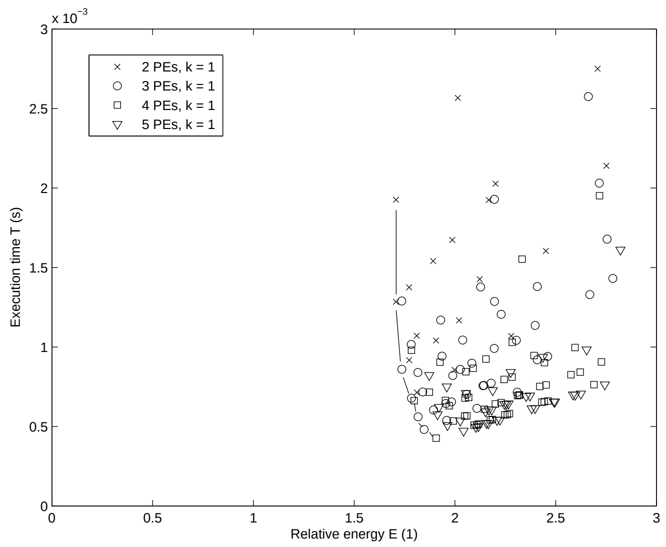


Fig. 2. Energy-time plot for different architectures with $k = 1$

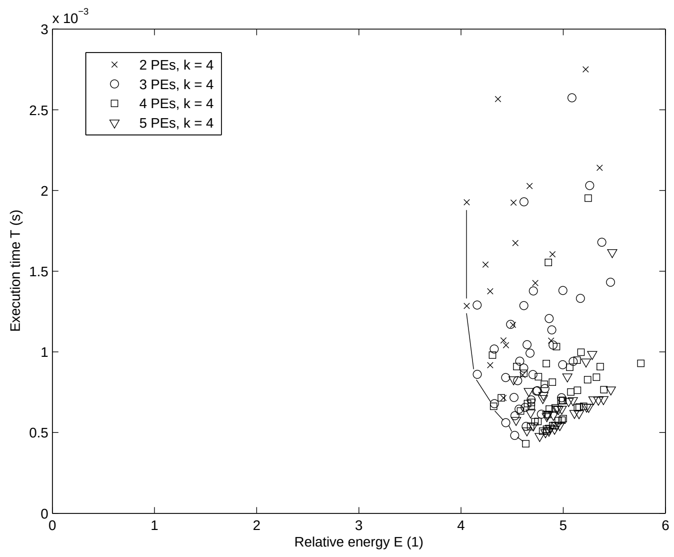


Fig. 3. Energy-time plot for different architectures with $k = 4$

each application. Energy E is the sum of objective function values (1) for a given architecture. Execution time T is the sum of execution times for a given architecture. Time is measured in seconds. Time is comparable even between different k values, but energy is not. A pair (E, T) presents a data point, or architecture, in the figure. E varied in range $[1.7, 3.1]$ for $k = 1$, and $[4.0, 5.8]$ for $k = 4$. Execution time sum T varied in range $[0.43, 3.85]$ms for both cases.

Pareto optimal solution boundary is marked with straight lines. These are not absolute Pareto optimums as not all possible architectures were evaluated. A Pareto optimal architecture is such that improving either energy or execution time leads to worsening the other factor. That is, in the Pareto optimal architectures, there are no two architectures where the other is better in terms of both energy and execution time.

TABLE IV

TOP 10 ARCHITECTURES TOGETHER WITH 3 PARETO OPTIMAL ARCHITECTURES FOR $k = 0$. ARCHITECTURES ARE LABELED WITH RESPECT TO THEIR ARCHITECTURE FINGERPRINTS. PARETO OPTIMAL ARCHITECTURES ARE MARKED WITH *. BEST VALUES IN EACH COLUMN ARE IN BOLD.

Arch. finger-print	E (1)	T (ms)	$\frac{E_{AA}}{E}$ (1)	$\frac{T_{AA}}{T}$ (1)	U_P (%)	U_I (%)	A (mm^2)
CC*	**0.925**	1.926	**1.999**	1.999	**100**	10	**2.4**
DD*	0.925	1.285	**1.999**	2.998	**100**	14	**2.4**
CCC	0.928	1.289	1.993	2.989	**100**	28	3.6
DDD*	0.928	0.860	1.991	4.481	99	41	3.6
CCE	0.935	1.016	1.977	3.789	99	35	4.6
CE	0.935	1.375	1.977	2.800	**100**	13	3.4
DF	0.936	0.917	1.976	4.199	**100**	20	3.4
DDF*	0.936	0.678	1.975	5.677	99	51	4.6
CCCC	0.941	0.980	1.965	3.931	98	52	4.8
CEE	0.941	0.840	1.964	4.583	99	42	5.6
DFF*	0.943	0.561	1.961	6.864	99	62	5.6
FFF*	0.953	0.482	1.939	7.999	98	73	6.6
DFFF*	1.001	**0.428**	1.847	**9.005**	90	**96**	7.8

A. *Top Architectures*

Table IV, Table V and Table VI show top 10 and all Pareto optimal architectures for cases $k = 0$, $k = 1$ and $k = 4$, respectively. $k = 0$ case is practically pure performance optimization, although measured in energy, but $k = 1$ and $k = 4$ are strictly energy optimization. Energy E and total execution time T are absolute values, and they are comparable to the slowest 2-PE architecture AA. $\frac{E_{AA}}{E}$ is the energy gain over AA architecture, the bigger the better. $\frac{T_{AA}}{T}$ is the speedup over AA architecture, the bigger the better. U_P is the mean PE utilization. U_I is the mean interconnect utilization. A is the area measured in square millimeters.

CC wins energy with all values of k. In $k = 1$ case, it is 1.7% more energy-efficient than the nearest 3 PE solution, CCC. It is $4,5\%$ more energy-efficient than the nearest 4 PE solution, CCCC. When the role of the dynamic energy increases in $k = 4$, the differences are larger: 2.5% and 6.2% against CCC and CCCC, respectively.

DFFF is the fastest architecture, and also a Pareto optimum. It has the highest performance processors given the area constraints. Note that 5 PE solutions do not have performance advantage over 4 PE solutions due to area constraints. For all values of k, DFFF runs at $4.5\times$ speed compared to the most energy-efficient architecture CC. However, it consumes only 8.2% ($k = 0$) to 14.3% ($k = 4$) more energy.

Most energy-efficient architectures have lower interconnect utilization U_I and higher processor utilization U_P than the fastest architectures. Lower processor utilization in high performance architectures can be explained with high peeks of performance demand that they can satisfy. Low performance architectures have longer task queues during peeks, which balances the load in time, but makes the critical path longer.

Approximately half the architectures are homogeneous in top 10.

Table VII and Table VIII show the proportion of how many times a given number and type of PE was in top 10 least

TABLE V

TOP 10 ARCHITECTURES TOGETHER WITH 3 PARETO OPTIMAL ARCHITECTURES FOR $k = 1$. FIGURE 2 SHOWS ARCHITECTURES GENERATED FOR $k = 1$ CASE.

Arch. finger-print	E (1)	T (ms)	$\frac{E_{AA}}{E}$ (1)	$\frac{T_{AA}}{T}$ (1)	U_P (%)	U_I (%)	A (mm^2)
CC*	**1.707**	1.926	**1.541**	1.999	**100**	10	**2.4**
DD*	1.708	1.285	**1.541**	2.998	**100**	14	**2.4**
CCC	1.736	1.289	1.516	2.988	99	27	3.6
DDD*	1.737	0.860	1.515	4.479	99	40	3.6
CE	1.773	1.376	1.484	2.800	**100**	13	3.4
DF	1.773	0.917	1.484	4.199	**100**	19	3.4
CCE	1.783	1.016	1.476	3.791	**100**	34	4.6
CCCC	1.784	0.980	1.475	3.929	98	50	4.8
DDF*	1.784	0.678	1.475	5.679	99	51	4.6
DDDD*	1.798	0.663	1.464	5.810	96	73	4.8
DFF*	1.817	0.561	1.448	6.864	99	61	5.6
FFF*	1.847	0.482	1.425	7.998	98	72	6.6
DFFF*	1.907	**0.427**	1.380	**9.028**	90	**95**	7.8

TABLE VI

TOP 10 ARCHITECTURES TOGETHER WITH 3 PARETO OPTIMAL ARCHITECTURES FOR $k = 4$. FIGURE 3 SHOWS ARCHITECTURES GENERATED FOR $k = 4$ CASE.

Arch. finger-print	E (1)	T (ms)	$\frac{E_{AA}}{E}$ (1)	$\frac{T_{AA}}{T}$ (1)	U_P (%)	U_I (%)	A (mm^2)
CC*	**4.055**	1.927	**1.228**	1.999	**100**	9	**2.4**
DD*	4.056	1.285	**1.228**	2.998	**100**	14	**2.4**
CCC	4.158	1.290	1.198	2.986	99	26	3.6
DDD*	4.159	0.861	1.197	4.476	99	40	3.6
CD	4.239	1.541	1.175	2.500	**100**	12	**2.4**
CE	4.285	1.376	1.162	2.800	**100**	13	3.4
DF	4.285	0.917	1.162	4.199	**100**	19	3.4
CCCC	4.308	0.980	1.156	3.930	98	49	4.8
DDDD*	4.319	0.663	1.153	5.812	96	71	4.8
CCE	4.325	1.018	1.151	3.785	99	33	4.6
DFF*	4.438	0.561	1.122	6.862	99	60	5.6
FFF*	4.526	0.482	1.100	7.990	98	70	6.6
DFFF*	4.633	**0.430**	1.075	**8.952**	91	**93**	7.8

energy consuming architectures for cases $k = 1$ and $k = 4$.

In $k = 1$ and $k = 4$ cases, 2 PE solutions filled 4 and 5 of the top 10 positions, respectively. 3 PE solutions filled 4 and 3 in those cases. 4 PE solutions filled 2 positions in both cases. 2 and 3 PE solutions seem suitable for low energy applications. However, 3 and 4 PE solutions have high performance.

There are no A and B type PEs in the top 10. This can be attributed to poor performance and energy inefficiency. f_{max} is a determining factor for static energy (1), and it puts processors with frequency less than f_{max} into disadvantage. The minimum value of f_{max} is 200MHz, because the interconnect is clocked at 200MHz. For this reason, A and B types are not favored. However, C and E types have the advantage of not increasing f_{max}. C type was the most common processor among low-energy architectures.

B. Pareto Optimal Solutions

Pareto optimal solutions are labeled with asterisk (*) in Table V and Table VI.

For the $k = 1$ case, static energy proportion for Pareto optimal architectures varies between 52% and 54%. For the

TABLE VII

PROPORTION OF HOW MANY TIMES A GIVEN NUMBER AND TYPE OF PE WAS IN TOP 10 ARCHITECTURES WITH $k = 1$. TABLE VALUES ARE EXPLAINED IN SECTION III-D.

PE type	Once (%)	Twice (%)	3 times (%)	4 times (%)	PE prop.
A	0	0	0	0	0
B	0	0	0	0	0
C	10	20	20	10	60
D	10	20	20	10	60
E	20	0	0	0	20
F	20	0	0	0	20

TABLE VIII

PROPORTION OF HOW MANY TIMES A GIVEN NUMBER AND TYPE OF PE WAS IN TOP 10 ARCHITECTURES WITH $k = 4$. TABLE VALUES ARE EXPLAINED IN SECTION III-D.

PE type	Once (%)	Twice (%)	3 times (%)	4 times (%)	PE prop.
A	0	0	0	0	0
B	0	0	0	0	0
C	20	20	10	10	60
D	20	10	10	10	50
E	20	0	0	0	20
F	10	0	0	0	10

$k = 4$ case, it varies between between 21% and 23%. Thus, the energy profile is rather uniform for both cases.

Pareto optimal solutions have 2, 3 and 4 PEs. 2-PE solutions do well due to low energy. 3 and 4 PE systems are do well due to a trade-off between energy and performance.

Figure 2 and Figure 3 show the clustering of solutions in the design space. Pareto optimal solutions constitute mere 5% and 6% of all solutions (7 and 8 out of 141) for $k = 1$ and $k = 4$, respectively. Therefore, automatic exploration is needed even when design space is limited to only 6 types of PEs and 2 to 5 PEs per architecture. It is not feasible to try out these solutions by manual work.

C. Optimization Convergence

Figure 4 and Figure 5 plot ratio $\frac{E_{AA}}{E}$ against mapping iterations for each Pareto optimal solution. The number of iterations it takes to win AA increases as the number of PEs increases. This comes from increased complexity of the mapping problem and the SA mapping algorithm that scales up iterations with respect to architecture complexity, the number of PEs. 4 PE architectures take over 20000 more iterations than 3 PE architectures to reach the level of AA (the gain value 1.0).

Our earlier work [8] presented an automated parameterization method for SA mapping. Originally it was only used for homogeneous architectures and performance optimization. The energy-time trade-offs presented in this paper indicate that the method can also be used for heterogeneous architectures and energy optimization.

In order to reach energy-efficiency of even AA architecture, it takes tens of thousands of mappings for 4 PE systems. Hence, it is a non-trivial problem in most cases. This creates demand for automated mapping (exploration). This may require behavior level simulation due to simulation time,

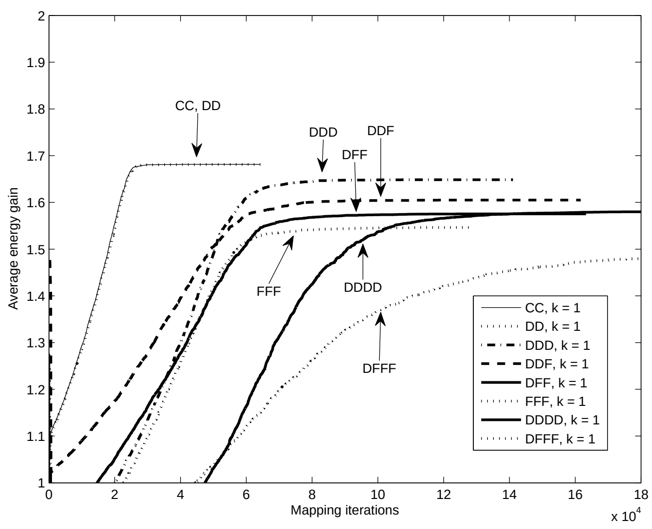


Fig. 4. Average energy gain plotted against mapping optimization iterations for the $k = 1$ case. Gain is computed as reference energy value divided by an energy value. Average gain is normalized to the average best objective value of the AA architecture. DD architecture's value 1.7 means AA consumed 1.7 times the energy of DD.

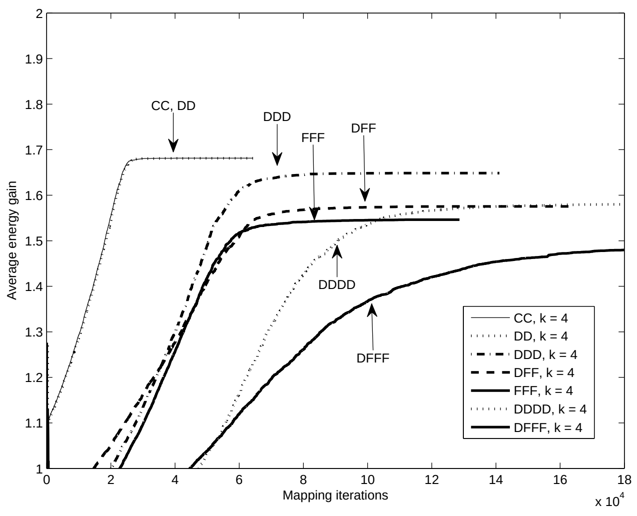


Fig. 5. Average energy gain plotted against mapping optimization iterations for the $k = 4$ case.

as was done in this paper. More accurate simulation may need thousands of CPUs. Fortunately, thousands of CPUs are reachable with current development budgets.

V. Conclusion

We evaluated heterogeneous architectures by optimizing both energy and performance for applications. The energy versus performance trade-off was analyzed by looking at Pareto optimal solutions. It was found that both energy-efficient and well performing solutions exist, and in general, performance is traded for energy-efficiency. Results indicated that automated exploration tools are needed when the mapping problem complexity grows, starting already with our experiment setup: 6 types of PEs to select from, and the system consists of 2 to 5 PEs.

Also, the results show that our Simulated Annealing method can be used in energy optimization for heterogeneous architectures, as well as in performance optimization for homogeneous architectures.

In the future, we plan to utilize SA to directly seek an optimal HW allocation and consider the bus or NoC energy more closely.

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, Proceedings. 41st, pp. 681-685, 2004.
[3] 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.
[4] 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, 2007.
[5] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Optimization by simulated annealing*, Science, Vol. 200, No. 4598, pp. 671-680, 1983.
[6] 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.
[7] G. Kahn, *The semantics of a simple language for parallel programming*, Information Processing, pp. 471-475, 1974.
[8] 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, 2006, pp. 73-76.
[9] *Standard task graph set*, ONLINE: http://www.kasahara.elec.waseda.ac.jp/schedule, 2003.
[10] *jobqueue*. Software. ONLINE: *http://zakalwe.fi/ shd/foss/jobqueue/*
[11] *OpenSSH*, software, ONLINE: *http://openssh.org*
[12] *rsync*, software, ONLINE: *http://rsync.samba.org*