Adaptive Mapping for Multiple Applications on Parallel Architectures

Ismail Assayad, Alain Girault

Adaptive Mapping for Multiple Applications on Parallel Architectures

Ismail Assayad[1] and Alain Girault[2]

[1] ENSEM, University Hassan II of Casablanca, Morocco,
i.assayad@ensem.ac.ma iassayad@gmail.com
[2] INRIA Rhône-Alpes and LIG, Grenoble, France
alain.girault@inria.fr

Abstract. We propose a novel adaptive approach capable of handling dynamism of a set of applications on network-on-chip. The applications are subject to throughput or energy consumption constraints. For each application, a set of non-dominated (Pareto) schedules are computed at design-time in the (energy, period, processors) space for different cores topologies. Then, upon the starting or ending of an application, a lightweight adaptive run-time scheduler reconfigures the mapping of the live applications according to the available resources (i.e., the available cores of the network-on-chip). This run-time scheduler selects the best topology for each application and maps them to the network-on-chip using the tetris algorithm. This novel scheduling approach is adaptive, it changes the mapping of applications during their execution, and thus delivers just enough power to achieve applications constraints.

Keywords : Adaptive Mapping, Multi-Objective Optimization, Energy, Throughput, Iterative Applications, Network-On-Chip.

1 Introduction

With an incessant technology improvement, we have witnessed a series of remarkable developments in systems on chip. One of them is the increasing processing capability of the system. The increase is not only achieved by the performance improvements between the generations of uniprocessors, but also comes from the advent of multi-core or many-core architectures where tens to hundreds of processors or cores can be integrated on a single chip. Examples of such architectures is presented in [1]. This introduces new big challenges.

The first challenge is the support for a variety of applications: mobile communications, networking, automotive and avionic applications, multimedia in the automobile and Internet interfaced with many embedded control systems. These applications may run concurrently, start and stop at any time. Each application may have multiple configurations, with different constraints imposed by the external world or the user (throughput constraints, deadlines and quality requirements, such as audio and video quality, output accuracy). The second challenge is to alleviate the power cost especially for battery powered devices. For

instance, the new generation of smart watches, led by Apple, have a recharging cycle measured in hours, some as low as eighteen [2]. It is clear that future wearables must deliver user functionality measured in days and weeks, not hours [3].

To address the previously mentioned challenges dynamic resource allocation and dynamic reconfiguration of applications must be supported. Also, resources must be scaled dynamically by operating frequency and voltage scaling (DVFS) in order to control the power consumption and to deliver just enough power. Power saving may be pushed even further by an adaptive scheduler strategy as performance constraints of applications and available resources to be used for each application may vary over time. This strategy is based on a careful evaluation of the power efficiency of combining DVFS with more application parallelism, as applications are launched or complete.

Since such a scheduler is intended for embedded platforms, only a lightweight implementation is acceptable at run-time. Therefore it is important to alleviate the run-time decision making on the one hand and to avoid the combinatorial complexity of the set of applications and over-approximations of fully pre-computed schedules on the other hand. For that we advocate an approach consisting of running applications in isolation, and computing off-line the set of optimal schedules for each application in isolation, before deciding on-line on the best combination of these schedules in function of the number of available processors and the performance constraints, whenever a change in the system configuration occurs. This approach is consistent because when aplications are run in isolation a strong pareto schedule of a combination of applications is necessarily a combination of weak or strong pareto schedules of individual applications.

The off-line part is performed by the tri-criteria optimization, to calculate 3D set of pareto schedules of individual applications. From a given software application graph, the optimization produces a static multiprocessor schedule that optimizes three criteria: its *schedule power*, its *period*, and its *processors number*. We target homogeneous mesh network-on-chip architectures. Our tricriteria scheduling uses DVFS to lower the power consumption. For a given number of processors all possible contiguous topologies are considered which results in a set of strong or weak pareto schedules.

The on-line part is performed by the run-time scheduler at each change to system configuration due to user requests. These changes include the start of a new application , the end of an application, a change of an application configuration, and a processor failure. The scheduler has then to adapt the schedules of the applicatiosn according to the new system configuration. for that it chooses, among the set of pre-computed 3D pareto solutions, one schedule per each application so that the set schedules is a optimal schedule for the current set of applications. To dot that, first the pareto schedules corresponding to the minimum number of processors and the maximum period are assigned to individual application. Then, if there are remaining processors, the on-line scheduler adapts the mapping by adding them to the applications that turn to achieve the most energy saving.

Selected solutions are then mapped on the NoC using a mapping technique based on an improved version of dellacherie algorithm (tetris) [4]. Since a set of schedules with given topologies may not be feasible due to topology constraint, tetris may fail to pack all topologies into the 2D mesh NoC. Hence, the run-time scheduler has also to explore the space of processor topologies when assigning processors to applications in decreasing order of energy saving. In that order when tetris fails it is possible for the run-time scheduler algorithm to backtrack to the last best set of schedules with no extra cost. This assignment problem is an instance of the multiple-choice knap-sack problem (MCKP), which is a variant of the 0-1 knapsack combinatorial optimization an NP-Hard problem [5]. Hence, we use a greedy linear heuristics which assigns one processor at a time to the set of applications in decreasing order of topology energy saving.

The main contribution of this paper is the adaptive scheduling method, the *first* adaptive scheduling heuristics able to produce, starting from applications algorithms graphs, NoC architecture graph, and throughput constraints, schedules with near-optimal energy savings based on pre-computed pareto schedules of individual applications.

The remainder of the paper is organized as follows. Section 2 introduces applications, architecture models, power and period definitions. Section 3 presents the scheduler part computed off-line while Section 4 formulates the schedules selection problem solved by our run-time scheduler heuristics. State-of-the-art on adaptive scheduling techniques for multiple applications on multi-core architectures is overviewed in Section 5. Finally, Section 6 reports the simulation results. Conclusions are drawn in Section 7.

2 Framework

In this section, we detail the application model, the platform model and the energy model. We end with the formal definition of the tri-criteria multiprocessor mapping problem.

2.1 Application model

We consider stream-based real-time application. Our model is therefore that of an application algorithm graph $\mathcal{A}lg$ which is executed repeatedly in a pipelined manner to achieve a better throughput.

$\mathcal{A}lg$ is an *acyclic oriented graph* $(\mathcal{X}, \mathcal{D})$ (See Figure 1(a)). Its nodes (the set $\mathcal{X}$) are software blocks called *operations*. Each arc of $\mathcal{A}lg$ (the set $\mathcal{D}$) is a *data-dependency* between two operations. If $X \triangleright Y$ is a data-dependency, then X is a *predecessor* of Y, while Y is a *successor* of X. The set of predecessors of X is noted $pred(X)$ while its set of successors is noted $succ(X)$. X is also called the *source* of the data-dependency $X \triangleright Y$, and Y is its *destination*. Operations with no predecessor (resp. successor) are called *input* operations (resp. *output*). Operations do not have any side effect, except for input/output operations: an input operation (resp. output) is a call to a sensor driver (resp. actuator).

Input and output operations read input data from the input drivers and write their output data to output drivers, respectively.

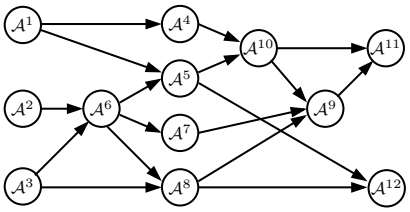

Fig. 1. An example of algorithm graph $\mathcal{A}lg$: I_1, I_2, and I_3 are input operations, O_1 and O_2 are output operations, A–G are regular operations.

2.2 Platform model

The target plateform is a homogeneous 2-D mesh-based NoC using deterministic X-Y communication strategy and providing computation and communication resources to implement multiple applications (Figure 2). A NoC is modeled as an architecture graph ($\mathcal{A}rc$). $\mathcal{A}rc =< \mathcal{P}, \mathcal{L} >$ is a graph where $\mathcal{P} = p_1, p_2, ..., p_q$ denotes the set of tiles on the NoC, corresponding to the set of $\mathcal{A}rc$ vertices, and $\mathcal{L} = \{(p_i, p_j, l_{ij})\}$ designates the set of communication links from nodes p_i to nodes p_j, corresponding to the edges of $\mathcal{A}rc$. l_{ij} represents the communication length from node p_i to node p_j. The number of nodes q in $\mathcal{A}rc$ is denoted as the size of the NoC.

The run-time scheduler schedules the given set of applications (e.g. $\mathcal{A}, \mathcal{B}, \mathcal{C}$ in figure 3) and manages the resources on the NoC. It runs on a dedicated processor and executes the proposed scheduling algorithms to map each application on a feasible region and loads all tasks on the tiles according to the mapping solution. This work deals with on-line scenarios, i.e., the scheduler does not know in advance when each application arrives or when it finishes. In this paper, we focus on the mapping algorithms of the scheduler.

2.3 Voltage, frequency, and power consumption

The maximum supply voltage is noted V_{max} and the corresponding highest operating frequency is noted f_{max}. For each operation, its WCET assumes that the processor operates at f_{max} and V_{max} (and similarly for the WCCT of the data-dependencies). Because the circuit delay is almost linearly related to $1/V$ [6], there is a linear relationship between the supply voltage V and the operating frequency f. From now on, we will assume that the operating frequencies are *normalized*, that is, $f_{max} = 1$ and any other frequency f is in the interval $(0, 1)$. Accordingly, the execution time of the operation or data-dependency X placed

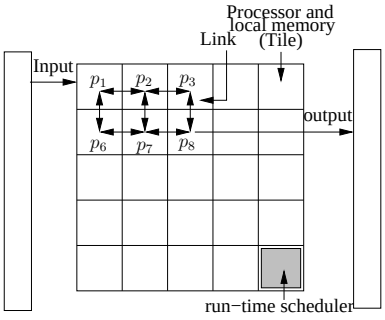

Fig. 2. NoC architecture.

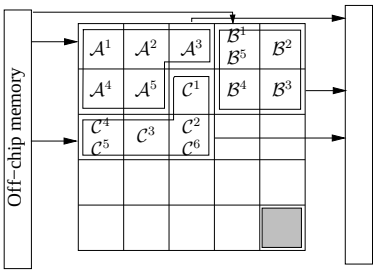

Fig. 3. Distributed mapping on NoC.

onto the hardware component C, be it a processor or a communication link, which is running at frequency f (taken as a scaling factor) is:

$$\mathcal{E}xe(X, C, f) = \mathcal{E}xe(X, C)/f \tag{1}$$

The power consumption P of a single operation placed on a single processor is computed according to the classical model of Zhu et al. [7]:

$$P = P_s + h(P_{ind} + P_d) \qquad P_d = C_{ef} V^2 f \tag{2}$$

where P_s is the static power (power to maintain basic circuits and to keep the clock running), h is equal to 1 when the circuit is active and 0 when it is inactive, P_{ind} is the frequency independent active power (the power portion that is independent of the voltage and the frequency; it becomes 0 when the system is put to sleep, but the cost of doing so is very expensive [8]), P_d is the frequency dependent active power (the processor dynamic power and any power that depends on the voltage or the frequency), C_{ef} is the switch capacitance, V is the supply voltage, and f is the operating frequency. C_{ef} is assumed to be constant for all operations, which is a simplifying assumption, since one would normally need to take into account the actual switching activity of each operation to compute accurately the consummed energy. However, such an accurate computation is infeasible for the application sizes we consider here.

For a multiprocessor schedule S, we cannot apply directly Eq (2). Instead, we must compute the total energy $E(S)$ consumed by S, and then divide by the schedule length $L(S)$:

$$P(S) = E(S)/L(S) \tag{3}$$

We compute $E(S)$ by summing the contribution of each processor, depending on the voltage and frequency of each operation placed onto it. On the processor p_i, the energy consumed by each operation is the product of the active power

$P^i_{ind} + P^i_d$ by its execution time. As a conclusion, the total consumed energy is:

$$E(S) = \sum_{i=1}^{|\mathcal{P}|} \left(\sum_{o_j \in p_i} (P^i_{ind} + P^i_d).\mathcal{E}xe(o_j, p_i) \right) \quad (4)$$

2.4 Period computation of a multiprocessor schedule

Figures 4 and 5 show an example of an application graph with three tasks, X which is mapped on P_2 and Y, Z which are mapped on P_1. Data dependencies are communicated through link L_{12}. The period P of the schedule is the time duration between two outputs of the application. Since Z is the output task, P is depicted as the duration of the time interval between ends of two occurence of Z. As suggested in the figure 5 by the rising arrow indicating the application second iteration movement toward the first one, the period can also be defined as the maximal utilization, including slack times, over processors and communication links. For this example for instance, this definition can be written as in Eq (5) where b and e denote the begin time and end time, respectively.

$$P = \max\left\{e(X,p_2) - b(X,p_2), e(Z,p_1) - b(Y,p_1), e(X \triangleright Z, l_{12}) - b(X \triangleright Y, l_{12})\right\} \quad (5)$$

It is worth noticing that this definition is interesting as it allows to reduce the number of MILP variables.

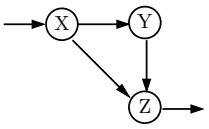

Fig. 4. Graph example.

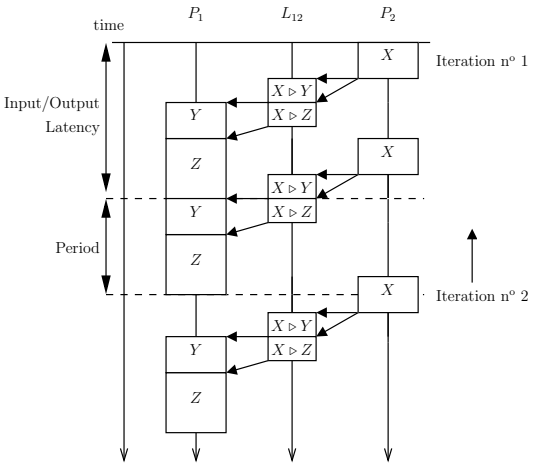

Fig. 5. A schedule and its period.

3 Off-line tri-criteria Optimization

3.1 Principle of the method and overview

With growing complexity of applications and embedded devices, deriving Pareto sets of application configurations is not trivial. In this paper we use the approach presented in [9] which involves transforming all the criteria except one into *constraints*, and then minimizing the last remaining criterion iteratively and which is inspired from the ϵ-constraint method [10]. Figure 6 illustrates the particular case of two criteria Z_1 and Z_2. To obtain the Pareto front, Z_1 is transformed into a constraint, with its first value set to $K_1^1 = +\infty$. The first run involves minimizing Z_2 under the constraint $Z_1 < +\infty$, which produces the Pareto point x^1. For the second run, the constraint is set to the value of x_1, that is $K_1^2 = Z_1(x_1)$: we therefore minimize Z_2 under the constraint $Z_1 < K_1^2$, which produces the Pareto point x^2, and so on. This process converges provided that the number of Pareto optima is bounded. Otherwise it suffices to slice the interval $[0, +\infty)$ into a finite number of contiguous sub-intervals of the form $[K_1^i, K_1^{i+1}]$. Another way is to slice the interval $[0, +\infty)$ into a finite number of contiguous sub-intervals of the form $[K_1^{i+1}, K_1^i]$.

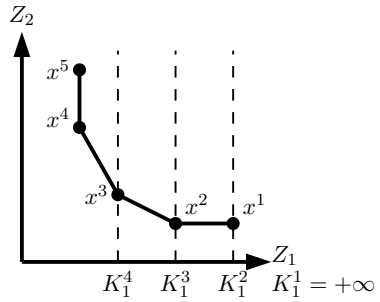

Fig. 6. Transformation method to produce the Pareto front.

The application algorithm graphs we are dealing with are small to medium The application algorithm graphs we are dealing with are small to medium (three to tens of operations, each operation being a software block), thereby making feasible exact scheduling methods such MILP or methods with backtracking, such as branch-and-bound.

For these reasons, our tricriteria scheduling technique minimizes the energy consumption under the double constraint that the throughput and the processors number remain below some thresholds, respectively T_{obj} and N_{obj}. By running it with decreasing values of T_{obj} and N_{obj}, starting with $(+\infty, +\infty)$, we are able to produce the Pareto front in the 3D space (Energy, Throughput, Processors number). This Pareto front shows the existing tradeoffs between the three

criteria, allowing the user to choose the solution that best meets his/her application needs. Finally, our method for producing a Pareto front could work with any other scheduling heuristics minimizing the schedule Energy under the constraints of both the throughput and the processors number.

3.2 Static optimal tricriteria scheduling Program - MILP

For each application, optimal schedules are computed off-line using an MILP tricriteria scheduling program. Inputs of the program are the application model $\mathcal{A}lg$, the NoC model $\mathcal{A}rc$ and the worst case execution-times WCETs of tasks and data communication. Outputs of the program is the pareto set composed of the optimal schedules of $\mathcal{A}lg$ on $\mathcal{A}rc$ s.t. in terms of Energy $E(S)$, Number of processors $N(S)$ and Throughput P.

DVFS is used to minimize the energy consumption of the architecture by exploiting the fact that a linear decrease of a processor frequency running a task, results in a cubic decrease in processor dynamic power at the price of only a linear decrease in execution-time of that task. DVFS can be done per island of tiles basis that is each island is optimized with its own supply voltage.

It is essential to note that for a given number of processors we might have one or several possible NoC sub-topologies whose pareto schedules may e. For example, for 4 processors there are 3 different topologies : the line topology, the T topology and the square topology. Except for the line topologies, we cannot guarantee at design-time that the other topologies will be feasible, i.e. may be successffully mapped on the NoC in the mapping phase of the run-time scheduler. Thus the pareto schedules for all NoC topologies alternatives with equal size must be computed and keeped by the program. For ease of presentation only we depict them on the same pareto set instead of separate ones by giving them different indexe values : the line topology, the T topology and the square topology are indexed with 4.1, 4.2, 4.3 in the processors axis. However, it is worth noticing that a total order is not defined on topologies with identical number of processors although it appears to be this way on the figure.

Figure 7 shows the grid with the constraints which is used to incrementally build the set of pareto schedules, and figure 8Â shows index values notation used for three sub-architectures of different topologies but same size, 4 processors.

4 Run-time scheduler

4.1 Working principle

The role of the run-time scheduler is that, given the current system configuration and the new set of active applications, it computes the next system configuration such that throughput constraints are met and power consumption is minimized. It should be noted, in passing, that although the throughput-case is adressed in this paper, the proposed approach is easily applicable to the power-mode case, i.e. satisfying all power constraints and minimizing period, on a simple condition that the pareto fronts are inverted.

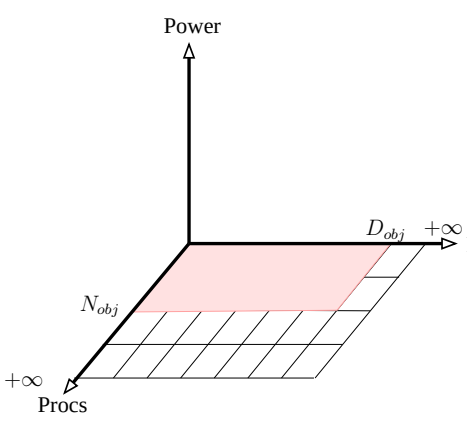

Fig. 7. Building a grid with the constraints.

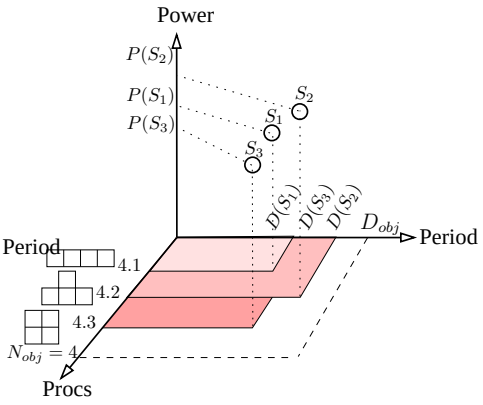

Fig. 8. Schedules for three tile topologies of $N_{obj} = 4$ processors under D_{obj} constraint.

After the off-line phase in which the static schedules are computed for each application individually comes the on-line phase. In this phase, at each change in the set of applications, the run-time scheduler has to choose the best schedule for each application so that the chosen schedules satisfy the following requirements :

- throughput constraints are still met for existing applications and the new one if any,
- power consumption of the architecture is minimized,
- schedules topologies best fit into the NoC architecture (a)
- applications reconfiguration cost will be amortized as the applications run repeatedly (b)

Requirement (a) means that the topologies should be mapped onto the NoC such that the number of Noc holes, i.e. unused tiles, is minimized. Requirement (b) means that the reconfiguration cost in term of energy consumption due to topologies re-mapping on the NoC should be compensated by the energy gain that would be achieved by the new configuration otherwise the re-mapping should be discarded to the extent possible.

Importance of these two requirements appears when many applications are competing for the NoC architecture. When the number of application increases and from a certain point, not all applications could benefit from the maximal number of processors as suggested by their respective pareto sets. This means that reconfigurations become inevitable and any unecessay hole may be prejudicial to the system in that it is a lost resource.

At run-time, to select the mapping of the next configuration we need to :

- select a schedule for each application,
- select NoC tiles for each schedule topology.

5 Related work

There has been a quite lot of research in multiple application design space exploration. some researchers focus on scenario-based approaches where multiple application mapping schenarios are explored at design time in order to handle dynamism in the number of active applications at runtime [11,12]. Others focus on one application with many configurations [13]. The scenario-based approaches, however, are not scalable even for small-sized applications as the number of scenarios increases exponentially with the number of applications and their configurations and become intractable; not to mention that these approaches forces the designer to recompute the whole schedules form scratch even after the addition of single new application or configuration.

In [14], authors proposed a mapping method whereby multiple applications can be simultaneously mapped on the many-core NoCs. However the proposed mapping is not adaptive and applications do not have constraints and are not reactive or iterative. In [15], authors include only a single mapping having minimum average power consumption. In [16] authors perform a mapping of tasks based on dynamically computed weighted sum of resources usage including processors, memories amd bandwidth utilization with the objective of optimizing resources utilization by minimizing the latter sum.

[17,18] presented an hybrid mapping technique where performance and energy schedules are computed at design-time and one schedule is selected at runtime to minimize energy consumption while satisfying performance constraints. in [17] constraint on end-to-end execution times of applications are considered rather than throughput of iterative applications. In [18] applications are iterative and throughput is optimized but the proposed technique is not adaptive. For instance it does not take advantage from system configuration change to minimize the energy consumption, neither by a combined re-parallelization and DVFS when additional ressources become available nor by some schedule sequentialization when throughput constraint is lowered. This can be explained by the fact that at design-time it generates a set of schedule by minimizing the throughput for each application for all possible number of processors rather than constraining the throughput. [19] work includes mappings having trade-offs between power consumption and performance but throughputs constraints are not included in the optimization. Moreover proposed approach adaptivity is limited as one schedule for each number of processors is retained for each application. In this case for instance the mapping of an application when a constraint is relaxed by the user can be changed only by assigning more processors and not by modifying current schedule.

Furthermore these works do not consider adaptivity regarding architecture topologies. They either over-approximate communication latencies using values computed assuming a (virtual) topology with max-hop links between every pair of processors like in [18], or use latencies computed by simulation for one fixed communication architecture (one schedule per a given number of processors) like in [17] [19].

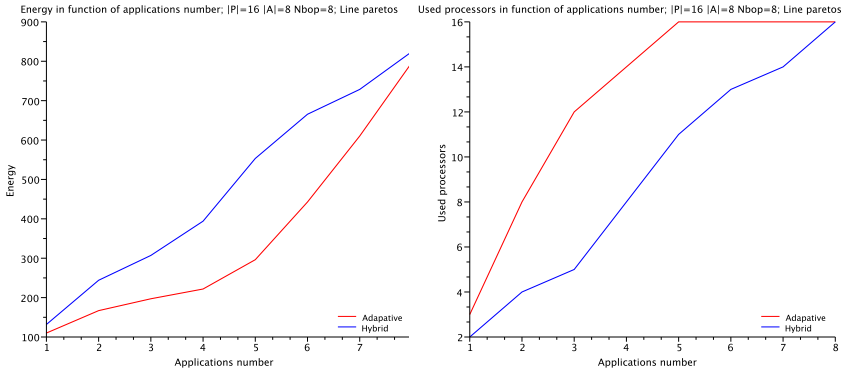

Fig. 9. Configurations computed by our approach and the ones computed by hybrid algorithm.

In our approach this level of adaptivity is also considered because NoC pareto schedules with less processors and better topologies may strongly dominate schedules with more processors but inefficient topologies.

6 Simulations

We have compared the performance of our approach called adaptive approach with the approach proposed in [18], called hybrid approach.

We have plotted in Figures 9(left), and 9(right), respectively the total energy consumption, and the total number of used processors of the schedules computed by adaptive and by hybrid. The values have been evaluated for a workload scenario starting from 1 up to 8 application graphs $\mathcal{A}lg$ of 8 operations each on a 4x4 $\mathcal{A}rc$ with throughput constraints equal to 100 units for all applications. Notice that in order to make comparisons under identical conditions, we use the results obtained when using only line topologies for the adaptive approach, i.e. without exploiting topology-level adaptability.

Simulation results in figure 9(left) show that adaptive approach performs systematically better than hybrid approach and that the energy saving reaches 36,33% when 5 applications are running on the NoC. This is explained by the fact that the adaptive approach better exploits the available resources in the NoC thanks to the parallelism-level adaptability as shown in the results of figure 9(right). For instance, when 3 applications are running on the NoC, adaptive approach is able to use twice as many processors as the hybrid approach. This can be explained by the fact that the schedule algorithmics used in the former approach combines parallelism with DVFS to achieve more energy saving while being below the throughput constraints, whereas the algorithmics of the later appraoch exploits parallelism to achieve the maximal throughput.

7 Conclusion

We have presented the adaptive energy throughput scheduling heuristics, called **adaptive**, to minimize the energy consumption and to satisfy throughput constraints for multiple applications on NoC. Our run-time scheduler goes beyond the traditional voltage scaling and power management level of adaptability of related scheduling approaches. It uses three levels of adaptability to achieve better energy savings : schedule-level adaptability, DVFS-level adaptability, and topology-level adaptability.

These advanced, and needed, levels of adaptability were possible for the run-time scheduler thanks to the multi-curve form of pareto sets derived off-line for each application individually. Both the throughput and the processors number are taken as *constraints*, so adaptive attempts to minimize the power while satisfying these constraints. By running the off-line part of adaptive with several values of these constraints, we are able to produce a set of pareto solutions taking care to not exclude weak pareto solutions dominated by others having same processors number but different topologies.

The run-time scheduler is able to take advantage from these pre-computed schedules by efficiently adapting applications schedules for the upcoming system configurations by reacting to applications starts or stops, application configurations change, throughputs constraints change, and processor failures.

To the best of our knowledge, this is the *first* reported adaptive method that allows user to compute schedules at run-time delivering just enough power to deliver the required functionalities. Moreover, because the pareto fronts computed off-line minimize both power consumption and throughput, the scheduling heuristics is also applicable without modification to the case of systems demanding high throughput under power constraints. This advance comes at the price of system re-configuration costs in transcient regimes which has to be minimized, a detailed analysis of this is beyond the scope of this paper and is a subject of future work.

8 Bibliography

References

1. S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, D. F. J. Tschanz, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w tera-ops processor in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 43(1), pp. 29–41, 2008.
2. "Apple watch battery life: how many hours does it last?" http://www.techradar.com/news/wearables/apple-watch-battery-life-how-many-hours-does-it-last--1291435 (visited on 09/01/2015).
3. "The battery is dead; long live power management," http://www.design-reuse.com/industryexpertblogs/38079/the-battery-is-dead-long-live-power-management.html (visited on 09/01/2015).

4. C. Thierry and B. Scherrer, "Building controllers for Tetris," *International Computer Games Association Journal*, vol. 32, no. 1, pp. 3–11, March 2010. [Online]. Available: http://hal.archives-ouvertes.fr/docs/00/41/89/54/PDF/article.pdf
5. R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, ser. The IBM Research Symposia Series, R. Miller, J. Thatcher, and J. Bohlinger, Eds. Springer US, 1972, pp. 85–103. [Online]. Available: http://dx.doi.org/10.1007/978-1-4684-2001-2_9
6. T. Burd and R. Brodersen, "Energy efficient CMOS micro-processor design," in *Hawaii International Conference on System Sciences, HICSS'95*. Honolulu (HI), USA: IEEE, Los Alamitos, CA, 1995.
7. D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *International Conference on Computer Aided Design, ICCAD'04*, San Jose (CA), USA, Nov. 2004, pp. 35–40.
8. E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Workshop on Power-Aware Computing Systems, WPACS'02*, Cambridge (MA), USA, Feb. 2002, pp. 179–196.
9. I. Assayad, A. Girault, and H. Kalla, "Tradeoff exploration between reliability, power consumption, and execution time for embedded systems - the TSH tricriteria scheduling heuristic," *STTT*, vol. 15, no. 3, pp. 229–245, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10009-012-0263-9
10. V. T'kindt and J.-C. Billaut, *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag, 2006.
11. P. van Stralen and A. Pimentel, "Scenario-based design space exploration of mpsocs," in *Computer Design (ICCD), 2010 IEEE International Conference on*, Oct 2010, pp. 305–312.
12. S. Stuijk, M. Geilen, and T. Basten, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, Sept 2010, pp. 548–555.
13. S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, ser. DATE '06. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006, pp. 118–123. [Online]. Available: http://dl.acm.org/citation.cfm?id=1131481.1131519
14. B. Yang, L. Guang, T. Xu, A. Yin, T. Santti, and J. Plosila, "Multi-application multi-step mapping method for many-core network-on-chips," in *NORCHIP, 2010*, Nov 2010, pp. 1–6.
15. A. Schranzhofer, J. Chen, and L. Thiele, "Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms," *IEEE Trans. Industrial Informatics*, vol. 6, no. 4, pp. 692–707, 2010. [Online]. Available: http://dx.doi.org/10.1109/TII.2010.2062192
16. J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous mpsocs," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.
17. G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria, "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 196–201.

18. A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 1, pp. 9:1–9:29, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2390191.2390200
19. C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *Computers Digital Techniques, IET*, vol. 5, no. 2, pp. 123–135, 2011.