A Divide and Conquer based Distributed Run-time Mapping Methodology for Many-Core platforms

Iraklis Anagnostopoulos, Alexandros Bartzas, Georgios Kathareios, Dimitrios Soudris
School of Electrical and Computer Engineering, National Technical University of Athens, Greece

***Abstract*—Real-time applications are raising the challenge of unpredictability. This is an extremely difficult problem in the context of modern, dynamic, multiprocessor platforms which, while providing potentially high performance, make the task of timing prediction extremely difficult. In this paper, we present a flexible distributed run-time application mapping framework for both homogeneous and heterogeneous multi-core platforms that adapts to application's needs and application's execution restrictions. The novel idea of this article is the application of autonomic management paradigms in a decentralized manner inspired by Divide-and-Conquer (D&C) method. We have tested our approach in a Leon-based Network-on-Chip platform using both synthetic and real application workload. Experimental results showed that our mapping framework produces on average 21% and 10% better on-chip communication cost for homogeneous and heterogeneous platform respectively.**

I. Introduction and Motivation

Future integrated systems will contain billion of transistors [1], composing tens to hundreds of IP cores. Modern embedded platforms take advantage of this manufacturing technology advancement and are moving from Multi-Processor Systems-on-Chip (MPSoC) towards Many-Core architectures employing high numbers of processing cores. Intel has already created platforms with 80 and 48 general purpose processing cores [2], [3], [4], while Networks-on-Chip (NoC) are already supported by the industry (such as the Æthereal NoC [5] from NXP and the STNoC [6] from STMicroelectronics). The industrial vision goes as far as thousand core chips [7]. The development of such many-core architectures is driven by the development of highly parallel/multi-threaded demanding applications. A big challenge in using such a complex system is among others to efficiently map the various applications on the many-core platform.

Resource management is a key technology for the successful use of computing platforms. The run-time resource management paradigm has become prominent recently because it can deal with the run-time dynamics of applications and platforms. Thus, the efficient run-time application mapping enables the efficient usage of the platform resources, minimizing mapping time, interconnection network communication load and energy budget. Existing approaches to run-time mapping algorithms on many-core platforms, even if they expose some autonomic properties, are typically centralized [8]. Traditionally, a central core periodically analyzes available information and computes a global configuration for the whole platform. It pushes this configuration out to the individual cores in a piecemeal fashion; alternatively these cores pull their respective configurations from the central one. However, such a centralized approach has several disadvantages. First, it creates a central point of failure, which renders the whole system unusable when the central core fails. Second, a central core limits scalability, because it represents a bottleneck for processing and communication functions, especially in environments that require frequent configuration changes. In this paper, we present a flexible distributed run-time mapping framework based on the Divide-and-Conquer strategy.

Divide-and-Conquer (D&C, derived from Latin: divide et impera) is a combination of political, military and economic strategy. Its goal is to maintain control by breaking up larger concentrations of power into smaller ones. These smaller concentrations of power individually have less power than the one implementing the strategy. Historically, it has been used by great empires such as ancient Rome, by dividing the land to smaller regions and setting local Kings.

In computer science, the D&C strategy consists in breaking a problem into simpler subproblems of the same type, next to solve these subproblems, finally to amalgamate the obtained results into a solution to the problem. Thus, it is primarily a recursive method for finding solutions to a big problem. The algorithms of this type display two parts; the first one breaks the problem into subproblems, the second one merges the partial results into the final result.

In this paper, we adopt the concept of the D&C method and use it so as to perform *distributed run-time mapping* on both homogeneous and heterogeneous many-core platforms. An abstract example of our approach implementing the D&C concept is presented in Figure 1. When a new application arrives, the "Emperor" task is triggered and it gets all the appropriate applications' information. Then, it chooses the most suitable region, in terms of available cores, that application can fit on and it sends a signal to a suitable core (making him "Local King") in that region to perform the run-time mapping. The novel ideas of this paper are:

- We propose a flexible distributed method for run-time mapping on *both* homogeneous and heterogeneous many-core platforms
- The flexibility of our approach is based on the fact that our run-time mapping framework can achieve different

This work is partially supported by the E.C. funded FP7-248716 2PARMA Project, www.2parma.eu
978-3-9810801-8-6/DATE12/©2012 EDAA

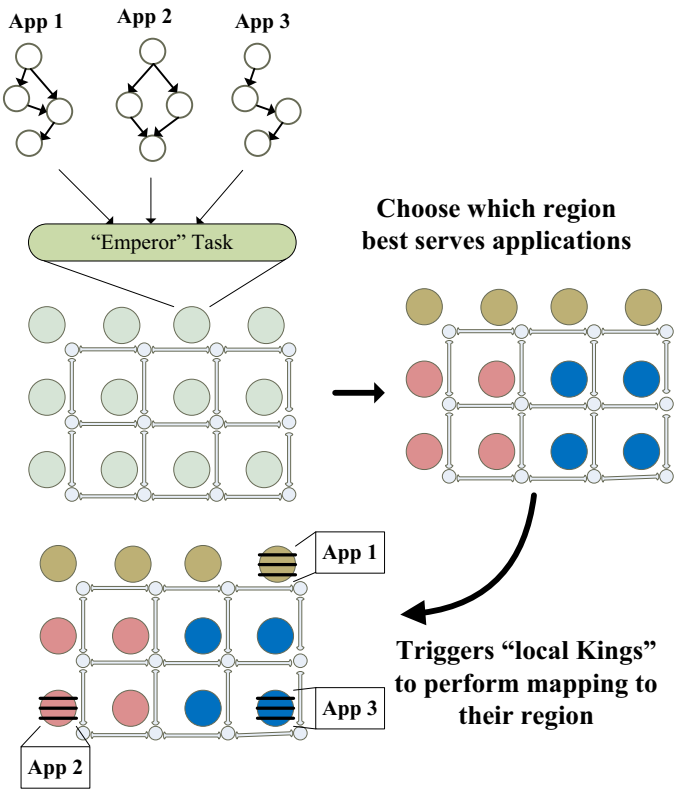

Fig. 1. Divide and Conquer on many-core platform.

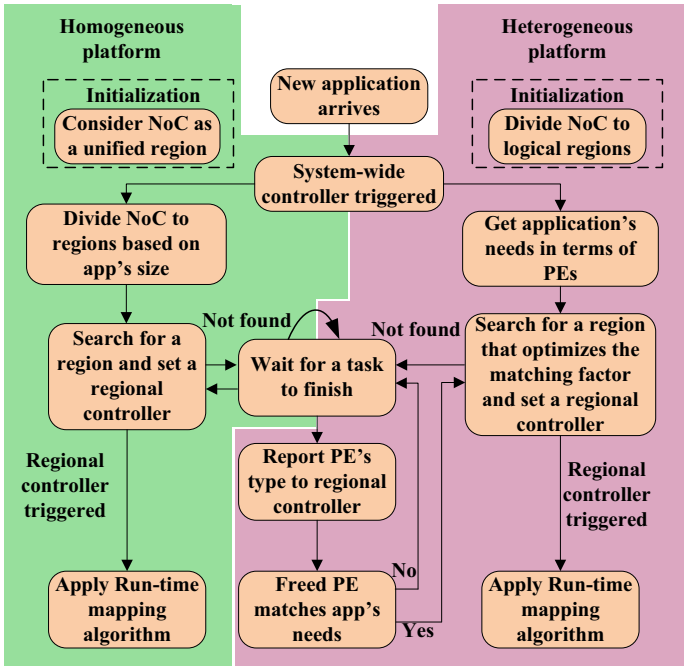

Fig. 2. Flow of our D&C methodology.

levels of platform's resources utilization depending on application's needs in comparison with other state-of-the-art distributed algorithms [9].

- We employ a fast node swapping procedure at the final step of the run-time mapping producing even better results in terms of on-chip communication cost.

The rest of the paper is organized as follows. The related work is presented in Section II. The adoption of the D&C approach in the context of our work is presented in more detail in Section III, where the proposed methodology framework is presented. The evaluation of the proposed approach and the simulation results are presented in Section IV and finally, conclusions are drawn in Section V.

II. Related Work

The authors of [10] present a mapping and scheduling strategy for hard real-time embedded systems, which communicate over a shared medium (i.e., bus) aiming at minimizing the system modification cost. A run-time application mapping onto homogeneous NoC platforms with multiple voltage levels is presented in [11]. That technique consists of a region selection algorithm and a heuristic for run-time application mapping. Broersma et al. [12] propose the MinWeight algorithm for solving the minimum weight processor assignment problem but only for task graphs with maximum degree at most two. Smit et al. in [13] extend the aforementioned algorithm by solving the problem of run-time task assignment on heterogeneous processors with task graphs restricted to a small number of vertices or a large number of vertices with degree of no more than two [12]. A unified single-objective algorithm, called UMARS, couples path selection, mapping of cores and TDMA time-slot allocation, such that the network required to meet the constraints of the application is minimized [14].

Faruque et al. [9] present a runtime application mapping in a distributed manner using agents targeting for adaptive NoC-based heterogeneous multi-processor systems. Authors claim that a centralized run-time resource management may bear a series of problems such as single point of failure and large volume of monitoring-traffic. However, Nollet et al. [15] present a centralized runtime resource management scheme that is able to efficiently manage a NoC containing fine grain reconfigurable hardware tiles and two task migration algorithms. The resource management heuristic consists of a basic algorithm completed with reconfigurable add-ons. The basic heuristic contains ideas from multiple resource management approaches. A greedy non-iterative algorithm is presented in [16]. Mapping is done based on core clustering where communication is routed by static xy routing. In this paper, we present a run-time distributed application mapping for homogeneous and heterogeneous many-core platforms.

III. Proposed Methodology Framework

The goal of the proposed methodology framework is to perform bandwidth-aware run-time mapping of application(s) described by their application task graphs in many-core network-on-chip architectures described by their application graphs.

An overview of our methodology framework is presented in Figure 2. Once a new application arrives the system-wide controller is invoked, the so called "Emperor" task is triggered, and according to the kind of many-core platform, the Homogeneous (Section III-B) or the Heterogeneous (Section III-C)

flow is followed. The system-wide controller is a light-weight task that performs the initial steps of the run-time mapping. It is responsible for:

1) getting the application's requirements,
2) selecting an appropriate region to map that application on and
3) triggering other cores in the region (the regional controllers, the so called "Local Kings" of Section I) so as to perform the run-time mapping algorithm.

A. Definitions

An application task graph (ATG) is used to capture the traffic flow characteristics. The ATG $G(T, D)$ is a directed acyclic graph, where each vertex t_i represents a computational module in the application. $K[t]$ $\forall t_i \in T$ specifies task's class type (e.g. logic task, computational task, memory task etc). Each directed arc $d_{i,j} \in D$ between tasks t_i and t_j characterizes data and communication dependencies. Each $d_{i,j}$ has an associated value $b(d_{i,j})$, which stands for the communication volume exchanged between tasks t_i and t_j.

A many-core platform topology and its communication infrastructure can be uniquely described by a strongly connected directed graph $A(I, N)$. The set of vertices N is composed of two mutually exclusive subsets N_{PE} and N_C containing the available platform's Processing Elements (PEs) and the platform's on-chip interconnection elements (such as routers in Network-on-Chip technology). $C[pe_i]$ $\forall pe_i \in N_{PE}$ specifies the class of the PE pe_i. The set of edges I contains the interconnection information (both physical and virtual) for the N set.

The mapped cores define the M_{PE} set. We also define a mapping function $map : T \rightarrow N_{PE}$ that maps the application's task (T set) to the available PEs (N_{PE} set). Let the set of unmapped nodes $\overline{M_{PE}}$ such as $pe \in \overline{M_{PE}}$ if $pe \notin M_{PE}$. From our definition it follows that $M_{PE} \cap \overline{M_{PE}} = \emptyset$.

We define the set R which describes the logical regions on the platform. R is composed of k ($k \geqslant 1$) subsets $R_1, R_2, ..., R_i, ..., R_k$ such that $\bigcap_{i=1}^{k} R_i = \emptyset$ and $\bigcup_{i=1}^{k} R_i = R$. $M_{R_i}[]$ is a list that defines the one to one result of the map mapping function in the R_i region. A region R_i is considered occupied iff $\exists pe_i \in R_i : pe_i \in M_{PE}$.

In an heterogeneous platform, the $C[pe_i]$ varies for each PE. Also, each t_i may require a special class of PE to run on, e.g. a DSP PE class. In order to comply with application's requirements and platform's resources and have the best $T \rightarrow N_{PE}$ correspondence, we define $\forall t_i \in T$ the parameter Matching Factor (MF) such that:

$$1 \geq \frac{C[pe_i]}{K[t_i]} \geq MF_{t_i}, \forall pe_i \in N_{PE} \quad (1)$$

MF is a designer specified parameter and defines the classes of PEs that the t_i can sit on. MF implies how good the class $C[pe_i]$ of pe_i element matches the specific t_i task and it defines a priority type on which core the task should be mapped on first. Different values and different decisions for MF result to different $M_{R_i}[]$ lists. In a homogeneous platform $MF_{t_1} = ... = MF_{t_N}, \forall t_i \in T$ because $C[pe_1] = ... = C[pe_N]$, $\forall pe \in N_{pe}$.

Algorithm 1 Homogeneous platform

```
   // Step 1: Check availability
 1: If |T| ⩽ |M_PE|
 2:     define new R_i ∈ R|∀pe_i ∈ R_i, pe_i ∈ M_PE
 3:     signal(R_i)
 4:     jump(Step 2)
 5: Else
 6:     wait() // for a task to release its PE
 7:     jump(Step 1)
   // Step 2: Run time mapping procedure
 8: ∀d_{i,j} ∈ D
 9:     ∀pe_i ∈ R_i
10:         src = min{F_HOM(d_i, pe_i)} // equation 2
11:         dst = min{F_HOM(d_j, pe_i)}
12:     M_PE, M_{R_i}[] ← src
13:     M_PE, M_{R_i}[] ← dst
   // Step 3: Swapping procedure
14: bestCost = bwCost{M_{R_i}[]} // equation 3
15: ∀t_i ∈ R_i
16:     ∀t_j ∈ R_i, t_j ≠ t_i
17:         If (MD(t_i, t_j) ⩽ MAX_MANH_DST)
18:             swap(t_i, t_j)
19:             tmpCost = bwCost{M_{R_i}[]}
20:             If tmpCost < bestCost
21:                 bestCost = tmpCost
22:                 M_{R_i}[] ⇐ new M_{R_i}[]
23:             Else
24:                 swap(t_i, t_j)
```

B. Homogeneous Platform

The starting point of the methodology is the annotated graphs of the many-core platform $A(I, N)$ and of the application(s) $G(T, D)$ to be mapped. The mapping algorithm utilizes this information and proposes the solution without violating the bandwidth constraints of the platform and the requirements of the application(s). The mapping procedure is presented in Alg. 1.

- Step 1 (lines 1-7): In the first step, we check the total number of tasks $|T|$. If platform is capable for serving the application a new region R_i is created and a signal is sent to a core inside the R_i region. This core plays the role of the regional controller ("Local King" as described in Section I) and it performs the run-time mapping algorithm. If this is not the case, we wait for a task to finish and free its PE. Due to fact that the set R is composed of mutually exclusive subsets, pe_i cannot belong to other subsets of R.
- Step 2 (lines 8-13): For every communication flow ($d_{i,j}$) in G, we find for the source (i) and the destination (j) the min value of cost function 2 for the selected PEs.

$$F_{HOM} = \sum_j (b(d_{i,j}) + MD_{i,j}) + \sum_i \sum_j b(d_{i,j}) \times MD_{i,j} \quad (2)$$

  where $MD_{i,j}$ is the distance (measured in hops) between pe_i and pe_j. This cost function combines the communication cost of the neighborhood of pe_i (first term) and the total communication cost of the platform (second term).
- Step 3 (lines 14-24): After the initial mapping has been performed we employ an iterative application node swapping process (similar to the one used in [17]) trying to

further reduce the total communication cost. During this process a pair of application nodes (mapped on platform nodes) is chosen and their position on the platform is swapped. After each swap the total communication cost (equation 3) is evaluated and if it is smaller than the previous value the swap is kept, otherwise the swap is not valid. The number of iterations of this swapping procedure is defined by the value MAX_MANH_DST.

$$bwCost = \sum_{M_{R_i}[]} b(d_{i,j}) * (MD_{i,j}) \quad (3)$$

C. Heterogeneous Platform

The starting point of the methodology for heterogeneous platforms is the annotated graphs of the many-core platform $A(I, N)$, of the application(s) $G(T, D)$ to be mapped on and the designer specified parameter MF. The algorithm searches for regions able to serve application's task as best as possible in terms of available classes. The algorithm for the heterogeneous platforms is presented in Alg. 2.

- Step 1 (lines 1-8): In the first step, we try to find a region in which all cores' classes match perfect with all application's tasks ($\frac{C[pe_i]}{K[t_i]} = 1$). If such region exists, a signal is sent to a core inside the region in order to perform the distributed run-time mapping algorithm.
- Step 2 (lines 9-13): If the first matching does not yield a result and application's requirements are not so strict ($MF < 1$) we change the binding according to MF_{t_i} and try to find regions that are as close as possible to the required MF_{t_i} $\forall t_i \in T$. If such a region exists this region is selected and the mapping algorithm is performed.
- Step 3 (lines 14-20): If still no matching region has been found, we search for any unoccupied R_i and we add to that R_i any $pe_i \in \overline{M_{PE}}$. If the new region $R_i^{'}$ is not able to serve the application, all PEs that were previously attached, they are now restored to their previous regions.
- Step 4 (lines 21-28): If still no matching region has been found or all regions are occupied, a new region R_i is created and any $pe_i \in \overline{M_{PE}}$ is added to that region. And in this case if the new region R_i is not able to serve the application, all PEs that were attached , they are now restored to their previous regions and we wait for a task to finish and free its PE.
- Step 5 (lines 29-37): In this step, we define the set S that contains all the flows $d_{i,j}$ whose either source's class ($K[t_i]$) or destination's class ($K[t_j]$) is bound to $K[t_k]$. This set is then sorted by $b(d_{i,j})$. We sort flows by bandwidth requirements as it helps in reducing bandwidth fragmentation and it important from a resource conservation perspective since the benefits of a shorter path grows with communication demands. Then, for every communication flow ($d_{i,j}$) in S, we find for the source (i) and the destination (j) the min value of cost function 4 for the selected pe_i.

$$F_{HET} = F_{HOM} + Q(C[pe_i]) \quad (4)$$

Algorithm 2 Heterogeneous platform

```
    // Step 1: Region selection step
 1: ∀t_i ∈ T
 2:   ∀pe_i ∈ N_PE
 3:     sort{MF_{t_i}}
 4: ∀R_i ∈ R
 5:   If (|T| ⩽ |M_PE|) && (∀t_i ∈ T, ∃pe_i ∈ M_{R_i} : C[pe_i]/K[t_i] = 1)
 6:     select(R_i)
 7:     jump(Step 5)
 8:
    // Step 2: if the first matching doesn't yield a result
 9: ∀R_i ∈ R
10:   If (|T| ⩽ |M_PE|) && (∀t_i ∈ T, ∃pe_i ∈ M_{R_i} : 1 > C[pe_i]/K[t_i] ≥ MF_{t_i} )
11:     select(R_i)
12:     jump(Step 5)
13:
    // Step 3: no matching region has been found
14: ∀ unoccupied R_i ∈ R
15:   ∀pe_i ∈ M_PE, pe_i ∉ R_i
16:     {R_i} = {R_i} + pe_i
17:   repeat(Steps 1-2) for R_i
18:   If R_i not selected
19:     {R_i} = {R_i} − pe_i, restore(R_i)
20:
    // Step 4: no region was found, or all regions are occupied
21: define new R_i = ∅ ∈ R
22: ∀pe_i ∈ M_PE
23:   {R_i} = {R_i} + pe_i
24: repeat(Steps 1-2) for R_i
25: If R_i not selected
26:   {R_i} = {R_i} − pe_i, restore(R_i)
27: wait() // for a task to release its PE
28: jump(Step 1)
    // Step 5: Run time mapping procedure
29: ∀K[t_k] ∈ G
30:   {S} = d_{i,j} iff (K[t_k] = K[t_i])||(K[t_k] = T[t_j])
31: sort(S) //by b(d_{i,j})
32: ∀d_{i,j} ∈ S
33:   ∀pe_i ∈ R_i
34:     src = min{F_HET(d_i, pe_i)} // equation 4
35:     dst = min{F_HET(d_j, pe_i)}
36:   M_PE, M_{R_i}[] ← src
37:   M_PE, M_{R_i}[] ← dst
    // Step 6: Swapping procedure
38: bestCost = bwCost{M_{R_i}[]} // equation 3
39: ∀t_i ∈ R_i
40:   ∀t_j ∈ R_i, t_j ≠ t_i
41:     If (MD(t_i, t_j) ⩽ MAX_MANH_DST)
42:       swap(t_i, t_j)
43:       tmpCost = bwCost{M_{R_i}[]}
44:       If tmpCost < bestCost
45:         bestCost = tmpCost
46:         M_{R_i}[] ⇐ new M_{R_i}[]
47:       Else
48:         swap(t_i, t_j)
```

where $Q(C[pe_i])$ is the requirements of class $C[pe_i]$ in terms of execution utilization and defines the PE's utilization by the task to be mapped on.

- Step 6 (lines 38-48): After the initial mapping has been performed we try to further reduce the total communication cost by employing the swapping technique. We swap a pair of mapped nodes and after each swap the total communication cost (equation 3) is evaluated. If it is better that the current communication cost the swap remains, otherwise, we restore it back.

IV. Experimental Results

We have performed extensive simulations of the behavior of several applications (a) MPEG-4, (b) Multi-Window Display (MWD) [18], (b) Picture-In-Picture (PIP) [18] (d) MultiMedia System (MMS) [19], (e) Digitale Radio Mondiale (DRM) [20]

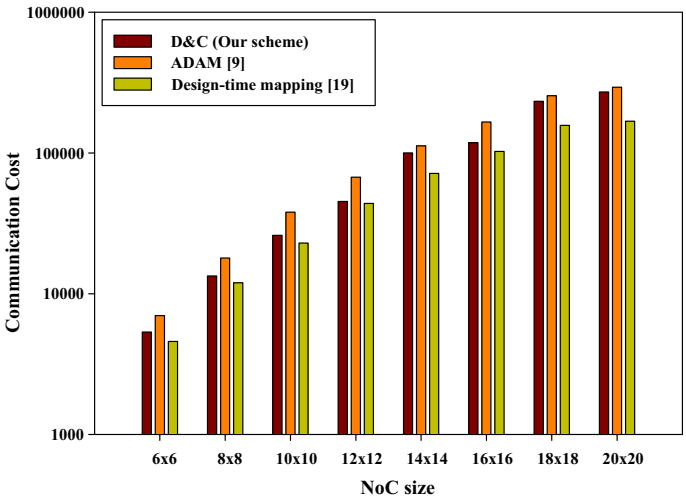

Fig. 3. Communication Cost comparison in homogeneous platforms.

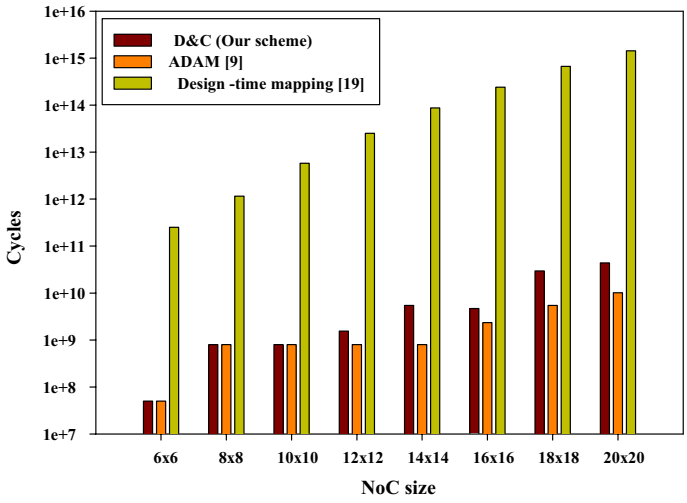

Fig. 4. Mapping computational effort in homogeneous platforms.

and (f) applications from TGFF [21] to validate our approach. An architecture that is able to accommodate a high number of cores, satisfying the need for high on-chip communication and data transfers, is the Network-on-Chip (NoC) architecture. The Network-on-Chip (NoC) model is emerging as a revolutionary architecture in solving the performance limitations arising out of long interconnects, outperforming more mainstream bus architectures and as a very good architecture template for many-core platforms.

In Figures 3 and 4 we compare for various homogeneous platforms, using TGFF application graphs, our D&C approach to the state-of-the-art distributed run-time mapping algorithm [9] and to an exhaustive design-time mapping [19], in terms of on-chip communication cost. The on-chip communication cost is a part of the mapping cost function for all these three mapping algorithms. Figure 3 shows that our D&C method achieves on average 21% better result in terms of final on-chip communication cost compared to the run-time algorithm presented in [9]. Although the optimal result is achieved by the exhaustive design-time mapping algorithm, our D&C method requires significant less cycles (improved scalability) in order to make the mapping decisions, as shown in Figure 4.

For the validation of our approach on heterogeneous platforms we used the platform presented in [22]. The platform is composed of Processor-Memory (PM) nodes interconnected via a packet-switched mesh network. A node can also be a memory node without a processor, pure logic or an interface node to off-chip memory. Each PM node contains a LEON3 processor, hardware modules connected to the local bus, and a local memory. The system uses a virtual-to-physical translation and all shared memories are globally visible to all nodes and organized as a single virtual addressing space. The communication of cores inside the platform is done using message-passing instructions and by using the shared memory interface. Whenever there is a need for the system-wide controller to trigger another core, the hardware's synchronization safe-lock memory mechanism is used. Shared memory environment

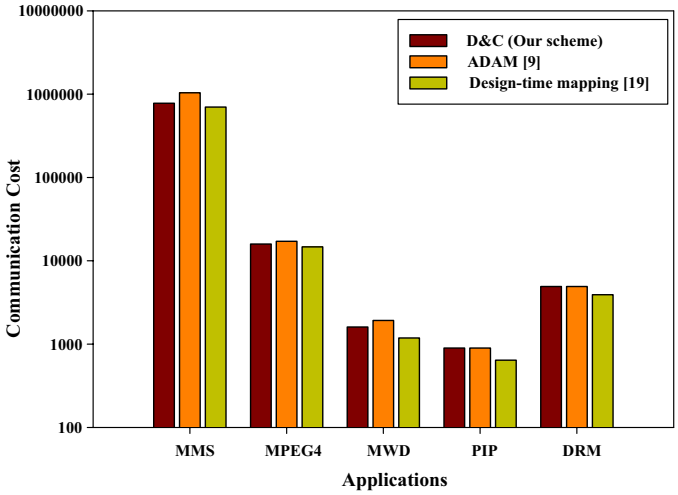

Fig. 5. Communication cost comparison for the five selected applications.

allows the ease use of such mechanisms. The lock is acquired by the system-wide controller and it propagates information to shared memory. Then the lock is freed and the region controller loads the data from the memory and performs the required mapping operations. The execution of code on a regional controller is also possible with the usage of message passing instructions.

Figure 5 presents a comparison of the on-chip communication cost for the five selected applications. We compare the on-chip communication cost of our D&C approach ADAM and with the exhaustive design-time mapping algorithm [19]. As Figure 5 depicts, our proposed algorithm has on average 10% better result in terms of on-chip communication that the ADAM run-time mapping algorithm. However, the best result is achieved, as expected, by the exhaustive design-time mapping algorithm. The cycles required for the result extraction are the same for both the run-time algorithms but for the exhaustive one is 100$\times$ bigger.

Several run-time scenarios were built on the NoC platform. The difference between the different scenarios is the number

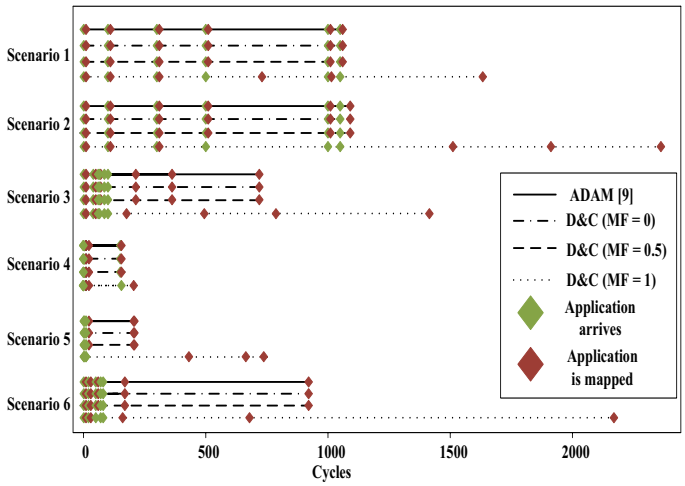

Fig. 6. Run-time mapping scenarios on an heterogeneous many-core platform.

TABLE I
UTILIZATION OF PLATFORM'S RESOURCES

	D&C ($MF = 1$)	D&C ($MF = 0.5$)	D&C ($MF = 0$)	ADAM [9]
Scenario 1	100%	92%	92%	91%
Scenario 2	100%	88%	88%	87%
Scenario 3	100%	87%	87%	88%
Scenario 4	100%	86%	86%	84%
Scenario 5	100%	78%	78%	79%
Scenario 6	100%	87%	87%	88%

of applications and the arrival time of each one of them. The arrival time was randomly generated. We compared our D&C scheme, using different matching factor values, with the ADAM [9] one. Figure 6 depicts all the implemented scenarios. The green diamond represents the arrival time of the application while the red one represents the time that the mapping result was taken. Three MF values are chosen to match what is provided by the many-core platform [22], [23]. The picture shows that both our D&C scheme (with $MF = 0$ and $MF = 0.5$) has the same run-time behavior with the ADAM approach. D&C scheme with $MF = 1$ has a different behavior because under the $MF = 1$ restriction a task can be mapped only on a core that has the same class type with the task. In this case, the algorithm takes more time because it waits for desired cores to be freed after finishing their tasks. D&C scheme with $MF = 1$ has the best task to core mapping decision resulting to best platform's resources utilization as depicted in Table I. Table I shows that with $MF = 1$, we can have 100% utilization of platform resources at run-time with a penalty cost at performance. If our application needs are not so strict we can choose other values of matching factor, thus relaxing the strictness of the matching.

V. CONCLUSION

In this paper we presented a D&C based distributed run-time application mapping framework for both homogeneous and heterogeneous many-core platforms. Our framework adapts to application's needs and application's execution restrictions by using the matching factor parameter. Our mapping framework produces on average 21% and 10% better on-chip communication cost for homogeneous and heterogeneous platforms respectively, compared to the ADAM [9] scheme with almost the same computational effort. The random implemented run-time scenarios showed that our algorithm can have different behavior according to the selected matching factor and resulting to different platform's resources utilization. Future extensions will include multi-objective mapping cost functions and multi-layered run-time controller.

REFERENCES

[1] Semiconductor Industry Association, "International technology roadmap for semiconductors," 2006. [Online]. Available: http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm

[2] J. Howard *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *Proc. of ISSCC*, feb. 2010, pp. 108 –109.

[3] L. Seiler *et al.*, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, pp. 18:1–18:15, August 2008.

[4] S. Vangal *et al.*, "An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS," in *Proc. of ISSCC*. IEEE, 2007, pp. 98–589.

[5] K. Goossens *et al.*, "Æhereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test*, vol. 22, no. 5, pp. 414–421, 2005.

[6] STMicroelectronics, "STNoC: Building a new system-on-chip paradigm," White Paper, 2005.

[7] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. of DAC*. ACM, 2007, pp. 746–749.

[8] E. Carvalho *et al.*, "Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs," in *Proc. of IWRSP*. IEEE Computer Society, 2007, pp. 34–40.

[9] M. A. Al Faruque *et al.*, "Adam: run-time agent-based distributed application mapping for on-chip communication," in *Proc. of DAC*. ACM, 2008, pp. 760–765.

[10] P. Pop *et al.*, "An approach to incremental design of distributed embedded systems," in *Proc. of DAC*. ACM, 2001, pp. 450–455.

[11] C.-L. Chou and R. Marculescu, "Incremental run-time application mapping for homogeneous nocs with multiple voltage levels," in *Proc. of CODES+ISSS*. ACM, 2007, pp. 161–166.

[12] H. Broersma *et al.*, "The computational complexity of the minimum weight processor assignment problem." in *Proc. of WG*, 2004, pp. 189–200.

[13] L. T. Smit *et al.*, "Run-time assignment of tasks to multiple heterogeneous processors," in *Progress 2004 Embedded Systems Symp., the*, 2004, pp. 185–192.

[14] A. Hansson *et al.*, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *Proc. of CODES+ISSS*. ACM, 2005, pp. 75–80.

[15] V. Nollet *et al.*, "Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles," in *Proc. of DATE*. IEEE Computer Society, 2005, pp. 234–239.

[16] K. Goossens *et al.*, "A design flow for application-specific networks on chip with guaranteed performance to accelerate soc design and verification," in *Proc. of DATE*, 2005, pp. 1182–1187.

[17] S. Murali and G. D. Micheli, "Bandwidth-constrained mapping of cores onto NoC architectures," in *Proc. of DATE*. IEEE Computer Society, 2004, p. 20896.

[18] D. Bertozzi *et al.*, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE TPDS*, vol. 16, no. 2, pp. 113–129, Feb 2005.

[19] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *IEEE TCAD*, vol. 24, no. 4, pp. 551–562, 2005.

[20] L. T. Smit *et al.*, "Run-time mapping of applications to a heterogeneous soc," in *Proc. of SoC*, 2005.

[21] R. P. Dick *et al.*, "Tgff: task graphs for free." in *CODES'98*, 1998, pp. 97–101.

[22] I. Anagnostopoulos *et al.*, "Custom microcoded dynamic memory management for distributed on-chip memory organizations," *Embedded Systems Letters, IEEE*, vol. 3, no. 2, pp. 66 –69, june 2011.

[23] X. Chen *et al.*, "Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller," in *Proc. of DATE*, 2010, pp. 39–44.