Microelectronics Journal 60 (2017) 129–143

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Models of computation for NoC mapping: Timing and energy saving awareness

César Marcon[a,*], Thais Webber[b], Altamiro Amadeu Susin[c]

[a] Computer Science Department, Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS, Brazil
[b] Computer Science Department, Universidade de Santa Cruz do Sul, UNISC, Brazil
[c] Computer Science Department, Universidade Federal do Rio Grande do Sul, UFRGS, Brazil

ARTICLE INFO

Keywords:
Application modeling
Mapping
Network-on-Chip (NoC)
Energy minimization
Performance analysis

ABSTRACT

A complex application implemented as a System-on-Chip (SoC) demands extensive system level modeling. Its implementation encompasses a large number of cores and an advanced interconnection scheme such as a Network-on-Chip (NoC). This type of application normally requires energy efficiency and execution time minimization, which implies high-level exploration for cores/tasks placement into the target architecture. A Model of Computation (MoC) captures some characteristics of the applications aiming to fulfill high-level explorations. This work analyzes MoCs employed on the static and dynamic mapping of applications onto regular NoCs, providing a classification based on aspects of computation and communication. Additionally, this paper discusses advantages and drawbacks of these MoCs, such as the complexity of capturing application aspects, as well as the mapping quality. Finally, this work implements the five MoCs more applied on the mapping and compares them applying a benchmark composed of synthetic and embedded applications running on various NoC sizes.

1. Introduction

Deep submicron technologies allow billions of transistors integrated into a single chip performing a complex System-on-Chip (SoC). These technologies pose formidable physical design challenges for long wires and global on-chip connections. Therefore, several designers have proposed to change from the fully synchronous design paradigm to the Globally Asynchronous, Locally Synchronous (GALS) design paradigm [1,2] enabling to split the application into synchronous domains, placing each domain inside a limited region called tile.

A Network-on-Chip (NoC) is an on-chip communication architecture composed of routers interconnected by point-to-point communication channels. The NoC channels may be adapted to GALS paradigm with asynchronous communication among synchronous domains. Besides, a NoC enables high communication rates and more than one parallel/concurrent communication [3].

The application-mapping problem consists in finding associations (mappings) among elements to minimize some cost function. Let an application be a set of m tasks or n cores running on a SoC based on NoC. Thus, the application-mapping problem takes into account two distinct mappings: (i) tasks into a core, i.e., *task mapping*; and (ii) core, such as processor or memory, into a NoC's tile, i.e., *core mapping*. The core-mapping problem allows $n!$ solutions, whereas the task-mapping problem allows much more possibilities since one or more tasks may be mapped onto a single core. Given a SoC with hundreds of cores, and possibly thousands of tasks, an exhaustive search of the mapping problem on the solution space is unfeasible. Thus, the optimal implementation of such application requires efficient mapping strategies. However, some strategies do not fulfill all design requirements due to the poor quality of the estimation tools, where an underlying Model of Computation (MoC) [4] that does not capture relevant aspects of applications, represents the application.

We analyze several works addressing the mapping problem for NoC-based SoCs, considering the communication and/or processing of an application in order to minimize energy consumption and execution time. A scheme is proposed to classify each work according to the underlying MoC, which determines the application complexity that affects the time and memory usage for mapping computation as well as for mapping quality. The most found classes of MoCs were formalized enabling to discuss the complexity in capture the application MoC. Besides, we show a set of mapping algorithms that implements each application MoC and the corresponding quality of mappings. These reasons are fundamental to found a tradeoff between modeling complexity, computational effort, and quality of the computational

* Corresponding author.
E-mail addresses: cesar.marcon@pucrs.br (C. Marcon), thaiscs@unisc.br (T. Webber), altamiro.susin@ufrgs.br (A.A. Susin).

http://dx.doi.org/10.1016/j.mejo.2016.09.005
Received 13 January 2016; Received in revised form 20 August 2016; Accepted 30 September 2016
Available online 04 January 2017

result.

The remaining of the paper is organized as follows. Section 2 explains MoCs composition applied to the mapping, proposing a classification scheme based on application aspects. Section 3 summarizes relevant works about MoCs targeting the mapping problem. Section 4 defines the mapping problem and architectural aspects. Section 5 explains the basis of mapping algorithms concerning each underlying MoC. Section 6 presents experimental results highlighting the mapping quality of each case study, and Section 7 presents conclusions and further discussions.

2. Composition and classification of MoCs

[5] proposed evaluating the application aspects separately to allow efficient design space exploration. The most important aspects to be considered in a computational system are *processing*, *communication* and *storing*, because any information (e.g., data or code) is in one of these states. Additionally, the majority of works ignores the storing state or groups it with processing in order to build a single aspect – this work follows this last approach.

Processing and communication are expressed regarding quantities and relations. *Quantity* is a value, such as the number of bits enclosed inside a message or as the number of milliseconds necessary to perform a task. *Order* is a relation that connects events, where each event is a quantity associated to a time stamp; and a set of events perform a total order [6], e.g., the order of tasks executed during an application simulation. *Dependence* is a relation of events relations establishing a *partial ordering*. Total order comprises a unique ordering whereas dependence comprises all possible ordering.

The complexity of extracting an application quantity or relation is an important element to consider in choosing the MoC. The designer may associate counters to each element of processing or communication to capture quantities (e.g., execution time, data volume). To capture the total order, the designer needs additional queuing of events often associated to time stamps. The application simulation enables to obtain both, quantity and order (i.e., total order). However, the capture of partial ordering requires analyzing application dependencies, respecting all possible total orders, which implies to comprehend the application semantic. Additionally, it is hard to automate the capture of partial ordering and, when done manually, it is an error prone task.

MoCs enable to specify the semantics of computation and concurrency of systems [6]; [4]. They have complexities that define the easiness of capturing the application behavior, the time and memory spent to perform the associated algorithms, the capacity of dealing with different requirements, and the quality of achieved results.

MoCs used in the application-mapping problem are composed of order or dependence, associated (or not) to the quantity of communication and/or processing. For instance, one can model the system communication by a directed graph, where each vertex is a communication channel and the directed edges are communication dependences. Depending on the graph's connectivity or on the information associated to each vertex, different mapping strategies are applied, and thus the MoC is classified accordingly. The classification and composition of MoCs for task/core mapping (Fig. 1) consider the evaluation of application aspects (communication and processing) and quantities or relations (quantity, order and dependence)..

Simple models are those where a single application aspect is associated to a single quantity or relation, while *composite models* are those that associate more than one quantity or relation to a single aspect. Both, single and composite models are classified as *homogeneous models*. On the other hand, *heterogeneous models* deal with communication and processing aspects together, with one or more quantity or relation enabling to derive classes of MoCs. Since for the same application aspect it is not possible to merge total and partial ordering in the same model, Fig. 1 presents all possible combinations of homogeneous models. However, Fig. 1 does not show all heterogeneous models; it only shows how they could be composed. The related work section details homogeneous and heterogeneous models that are used specifically on mapping problem (refer to Table 1).

3. Related work for MoCs for mapping

The designer may map an application onto a NoC during the design time (statically) or during application execution (dynamically). Sahu and Chattopadhyay [7] published a thorough study of mapping approaches focusing on the underlying mapping algorithm. However, their work does not discuss the underlying target application MoC, which directly influences the mapping quality. Thus, this section evaluates a comprehensive subset of mapping works to classify their underlying MoC according to the scheme proposed in Section 2.

The communication aspects play a crucial role in the mapping problem for NoCs with several cores, so the designers use MoCs mainly to model communication behavior instead of modeling the processing. Consequently, almost all MoCs presented in this section contain communication aspects, and Cq is the most used model. Additionally, just a half part of models contains processing aspects. Table 1 explains eight classes of MoCs used on the related works.

Table 2 summarizes requirements, communication architecture, MoC name and type of related works.

It displays the following evidences: (i) MAP (i.e., mapping type) shows that the majority of works focus on static (ST) mapping. However, there is a balance between dynamic (DY) and static mapping in recent years; (ii) Design requirements displays that energy reduction is the most explored requirement, followed by timing/latency reduction. Only few works do not focus on one of these requirements; and (iii) practically all communication architectures are NoC mesh or at least regular tile-based NoCs. Additionally, column *MoC name (type)* shows the name of the MoC used in the related work associated with our classification scheme.

4. Problem formulation

This section formalizes and exemplifies the five most cited MoCs and communication architecture of Section 3, as well as the energy and timing models for further discussions. There are some possible variations inside each class of MoC. However, this work addresses only the core-mapping problem; nonetheless, one may apply analogous definitions to the task-mapping problem.

4.1. MoCs definitions

Definition 1. *Communication Quantity* (Cq) model is a directed graph $< C, W >$. Let $C = \{c_1, c_2..., c_n\}$ be the set of vertices or application cores, and $w_{ij} \in \mathbb{N}^*$ be the number of bits of all packets sent from a core c_i to a core c_j, then W is $\{(c_i, c_j) \mid c_i, c_j \in C \text{ and } w_{ij} \in \mathbb{N}^*\}$, the set of edges. Each edge is labeled with the value w_{ij}, which denotes the quantity of bits of all communications between application cores. Cq is a homogeneous MoC based on communication aspects of the application, i.e., on the quantity of bits exchanged between application cores.

Aiming to show a possible variation of a given MoC, Definition 2 formalizes Cqe, which extends Cq through aggregating traffic pattern evaluation.

Definition 2. Cqe extends Cq model, where the set of edges $W = \{(c_i, c_j) \mid c_i, c_j \in C, w_{ij} \in \mathbb{N}^*, f_{ij} \in \mathbb{N}\}$ represents all communications containing both, the amount of bits of all packets sent from core c_i to core c_j (w_{ij}) and the number of bit transitions occurred during the transmission of all packets from c_i to core c_j (f_{ij}).

Definition 3. *Communication Quantity and Ordering* (Cqo) model is a list of message sets. Let $w_{ijq} \in \mathbb{N}^*$ be the number of bits of the *q-th*

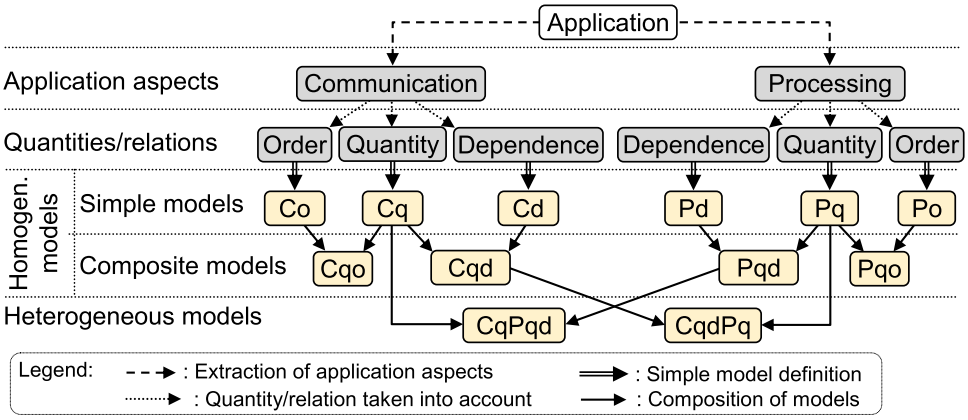

Fig. 1. Examples of some compositions of MoCs for application on mapping problems.

Table 1
Summary of MoCs used on mapping problem.

MoC	MoC explanation
Cq	Untimed model capturing only communication quantity of application, i.e., number of bits transmitted between tasks/cores
Cqo	Discrete event model improving Cq by capturing communication instants, i.e., each packet has a time tag performing total ordering
Cqd	Message dependence model capturing the quantity of communication and partial order of messages
Pqd	Processing dependence model that captures the deadlines of tasks
CqPq	Untimed model capturing communication and processing quantities
CqPqd	Improvement of Pqd by the capture of communication quantity
CqdPq	Improvement of Cqd, since it also captures some computation aspects
CqoPqo	Timed model that improves Cqo by capturing processing ordering

message sent from core c_i to a core c_j. Then $m_q = (c_i, c_j, w_{ijq}) \mid c_i, c_j \in C$ is the *q-th* message from core c_i to core c_j with w_{ijq} bits. Let $M = \{m_q \mid q \in \mathbb{N}^*\}$ be the set of all messages between application cores and δ be a subset of M. Thus, Cqo $= \{T = \{(t_i, \delta_k) \mid \delta_k \neq \emptyset, t_i \in \mathbb{N}^*, i \in \mathbb{N} \text{ and } 1 \leq k \leq q\} \cup \{Start = (0, \emptyset), End = (\infty, \emptyset)\}\}$ represents an ordered list of message sets, such that t is a time tag that marks the start time of the message $\boldsymbol{m}$. Two special vertices (*Start* and *End*) indicate the beginning and the end of the application. *Start* and *End* are 2-tuples, and (t_r, t_s) with $r \neq s$ implying $t_r < t_s$ is the set ordering. Cqo is a discrete event model [6] that is classified as a homogeneous and composed model based on the communication aspects of the application, more specifically quantity and total order together.

Definition 4. *Communication Quantity and Dependence* (Cqd) model is an acyclic directed graph $< P, D >$. The set P of vertices contains all packets exchanged between each pair of communicating cores. D is a set of edges containing all dependences of communication. The elements of the set P are 3-*tuples* in the form $p_{ijq} = (c_i, c_j, w_{ijq})$, where c_i and $c_j \in C$, and $w_{ijq} \in \mathbb{N}^*$ is the number of bits transmitted from core c_i to core c_j into the *q-th* packet p_{ijq} with $q \in \mathbb{N}^*$. Additionally, P_{ij} is the set of all packets sent from core c_i to core c_j. Cqd represents the communication of an application composed of an arbitrary number of cores. The direction of the edges in this graph indicates that the destination vertex computation depends on the communication provided by the origin vertex. In other words, the destination vertex presents a *communication dependence* regarding the origin vertex.

Definition 5. The *Communication Quantity and Dependence, and Processing Quantity* (CqdPq) improves Cqd by aggregating processing quantities. The sets of vertices and edges (P and D) are equivalent to the ones of Cqd – Definition 4. However, elements of P on CqdPq are 4-tuples in the form $p_{ijq} = (c_i, c_j, t_{iq}, w_{ijq})$, where only c_i, c_j and w_{ijq} are the same of Cqd, and $t_{iq} \in \mathbb{N}^*$ is the computation time elapsed, since all vertex dependences are solved until this vertex transmits its communication.

Fig. 2 shows a synthetic concurrent application, described with MPI (Message Passing Interface) primitives, which is distributed into four cores {A, B, E, F}..

Fig. 3 illustrates how the five MoCs defined in this section model the synthetic application of Fig. 2..

Fig. 3(a) shows the Cq, where C ={A, B, E, F}, and the edge labels are w_{AB}=15, w_{AF}=15, w_{BF}=40, w_{EA}=35, w_{FB}=15. Fig. 3(b) displays the Cqe, where the edge labels are (w_{AB}, f_{AB})=(15, 8), (w_{AF}, f_{AF})=(15, 0), (w_{BF}, f_{BF})=(40, 20), (w_{EA}, f_{EA})=(35, 15), (w_{FB}, f_{FB})=(15, 10). Fig. 3(c) shows a possible Cqo for the application, where the set of messages is M={m_1=(A, B, 15), m_2=(E, A, 20), m_3=(B, F, 40), m_4=(A, F, 15), m_5=(E, A, 15), m_6=(F, B, 15)}, which is partitioned in subsets δ_1={m_1, m_2}, δ_2={m_3}, δ_3={m_4}, δ_4={m_5, m_6}. Then, T={(10, δ_1), (20, δ_2), (60, δ_3), (90, δ_4)}. Fig. 3(d) depicts Cqd modeling the application, where P={p_{BF1}=(B, F, 40), p_{AB2}=(A, B, 15), p_{EA3}=(E, A, 20), p_{AF4}=(A, F, 15), p_{FB5}=(F, B, 15), p_{EA6}=(E, A, 15)}, and D={(Start, p_{BF1}, p_{AB2}, p_{EA3}), (p_{BF1}, p_{AF4}), (p_{AB2}, p_{AF4}), (p_{EA3}, p_{AF4}, p_{EA6}), (p_{AF4}, p_{FB5}, p_{EA6}), (p_{FB5}, End), (p_{EA6}, End)}. Fig. 3(e) illustrates CqdPq modeling the application, where P={p_{BF1}=(B, F, 20, 40), p_{AB2}=(A, B, 10, 15), p_{EA3}=(E, A, 10, 20), p_{AF4}=(A, F, 6, 15), p_{FB5}=(F, B, 8, 15), p_{EA6}=(E, A, 8, 15)}, and D is the same of Cqd.

In this section, all defined MoCs can capture bits traffic, which is one of the most important elements that contribute to the NoC dynamic energy consumption. However, models that capture bits transition and bits volume separately can perform better estimations, which is the case of Cqe. The total time spent by the application execution also affects the dynamic energy consumption, increasing the importance of models that capture this information, for instance, Cqo and CqdPq.

Among the models previously defined, only Cqd, Cqo, and CqdPq can estimate the application execution time. However, Cqd is appropriate for modeling I/O bounded applications since these applications are continuously transmitting packets and the processor time is less relevant to predict packets contention. On the other hand, Cqo may produce mappings that avoid packets contention for CPU bounded applications, since the communication is largely spaced and errors, inserted by a false total order provided by the model, do not affect the estimation results. Finally, CqdPq is suitable for any application because this model enables to predict packets contention independent of communicating and processing features.

4.2. Communication architecture definition

Mapping approaches are useful for all communication architectures where the cores/tasks position may affect the overall performance. However, this paper explores the generic model of 2D-mesh NoC with

Table 2
Related work and their underlying MoC classification.

Work	MAP	Design requirements	NoC	MoC name (type)
[8,9]	ST	Bandwidth minimization	Mesh, torus	Core graph (Cq)
[10]	ST	Energy and time reduction	Mesh	CTG (CqPqd)
[11]; [9]	ST	Energy, area and time reduction	Clos, torus, mesh	Core graph (Cq)
[12]	ST	Energy saving and bandwidth reservation	Mesh	APCG (Cq)
[13]	ST	Energy and time reduction	Mesh	CWM, CDM (Cq, Cqd)
[14]	ST	Energy and time reduction	Mesh	CDM, CDCM (Cqd, CqdPq)
[13]	ST	Energy and latency reduction	Mesh, torus, fat-tree	ACP (Cqo)
[15]	ST	Energy saving	Mesh	ECWM (Cq)
[16]; [17]; [18]	ST	Energy and area reduction	Mesh, torus	Core graph (Cq)
[19]	ST	Hops minimization and thermal balance	Mesh	APCG (Cq)
[20]	ST	Energy and area reduction, QoS	Ad hoc	Multigraph (Cq)
[21]	ST	Energy saving	Mesh	Task graph (Pqd)
[22]	DY	Energy and time minimization	Mesh	ACG (CqPq)
[23]	ST	Energy saving	Mesh	WCTG (Cq)
[24]	DY	Energy, channel occupation, latency reduction	Mesh	CDCM (CqdPq)
[25]	ST	Energy reduction	Regular tile	MACTG (CqPqd)
[26]	DY	Energy and congestion reduction	Mesh	CTG (CqPqd)
[27]	ST	Energy saving	Regular tile	ACG (CqdPq)
[28]	ST	Latency minimization and throughput increase	Mesh	Task graph (Cq)
[29]	ST	Energy and latency reduction	Mesh	ACG (CqPq)
[30]	ST	Energy saving	Mesh	Task graph (CqPqd)
[31]	DY	Load balance and comm. overhead reduction	Mesh	ATG (CqoPqo)
[32]	DY	Hotspot avoidance, uniform energy consumption	Mesh	CTG (Pqd)
[33]	ST	Hops minimization	Mesh	TFG (Cqo)
[34]	DY	Energy and time saving	Mesh	ATG (Cqd)
[35]	ST	Energy and latency reduction	Ad hoc	Core graph (Cq)
[36]	DY, ST	Time saving and real-time constraints fulfilling	Abstract model	KPN (CqoPqo)
[37]	DY, ST	Energy efficiency	Mesh	ACG (Cq)
[38]	DY	Time and energy saving	Mesh	DAG (Cqd)
[39]	DY, ST	Energy saving, load balancing	Mesh	TCG (Cq)
[4]	DY	Time saving	Mesh	ATG (CqdPq)
[40]	ST	Time and energy saving, and reliability	Abstract model	APCG (Cq)
[41]	DY	Time and energy saving	Mesh	App. Graph (CqdPq)
[42]	DY, ST	Load balance, time and energy saving	Mesh	CWM (Cq)
[43]	ST	insertion loss and crosstalk noise minimization	Mesh, torus	CG (Cq)
[44]	ST	Energy efficiency, thermal hotspot avoidance	3D mesh	Core graph (CqdPq)

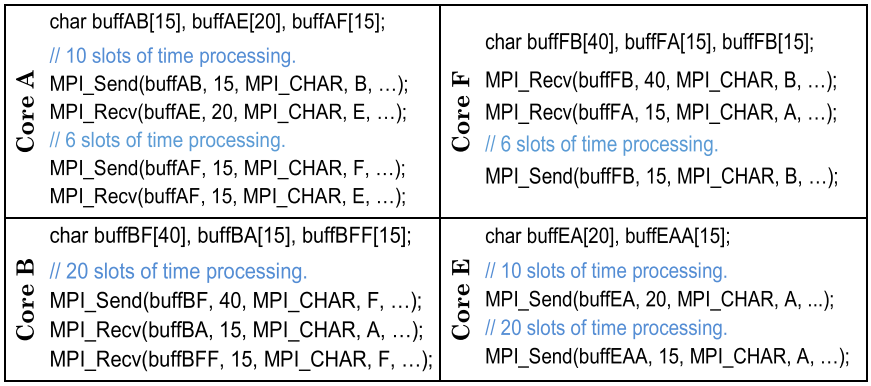

Fig. 2. Synthetic application with {A, B, E, F} cores. Each core shows a comment of processing time and subsequent actions of send/receive. The cores operation is supposed to occur in parallel, with synchronizing activities defined by send/receive MPI directives.

XY routing algorithm and wormhole control flow as target architecture.

Definition 6. A *Communication Resource Graph* (CRG) is a directed graph $CRG = < \Gamma, L >$, where the vertex set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ is the set of tiles, and the edge set $L = \{(\tau_i, \tau_j), \forall \tau_i, \tau_j \in \Gamma\}$ gives the set of links from τ_i to τ_j. $|\Gamma|$ is the total number of tiles. CRG vertices and edges represent routers and physical links of the NoC, respectively. The CRG definition is equivalent to several target architectures definitions, e.g., the *NoC topology graph* [8,9] and *architecture characterization graph* [10]. Fig. 4 depicts Definition 6 using a synthetic application with four cores arbitrarily mapped onto a CRG..

4.3. Energy consumption and timing models

Dynamic energy consumption is proportional to switching activity, arising from packets moving across the NoC, dissipating energy on the links and on the circuits and buffers of each router. Moreover, circuits for monitoring incoming packets also consume dynamic energy even when the router is in the idle state. This last parcel is named *idle energy* to distinguish both dynamic energies. Leakage current originates the static energy consumption that is proportional to the application execution time and the number of transistors. Normally, static energy is responsible for the smallest part of the energy consumption; however, for deep-submicron technologies, the designer cannot neglect the leakage current effect.

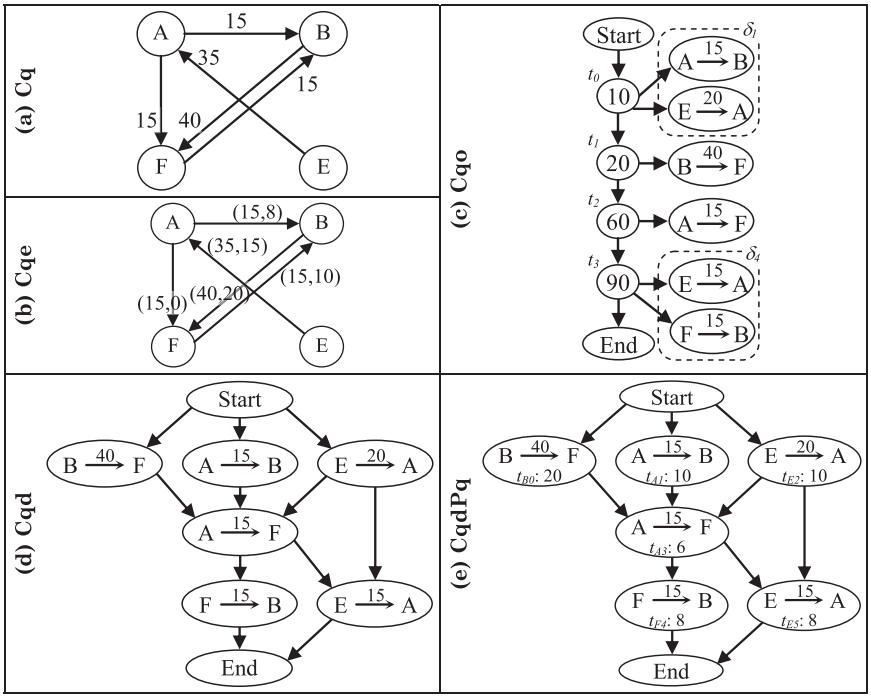

Fig. 3. MoCs based on the synthetic application of Fig. 2.

This work computes the dynamic energy similarly to [45], but some concepts are extended to consider static energy consumption and the *idle energy* consumed by a router. The concept of bit energy *Ebit* is used to estimate the dynamic energy consumption for each bit of each packet. Two components split *Ebit*: bit dynamic energy dissipated on the router and link between a router and the local core (*ERbit*); and bit dynamic energy dissipated on links between tiles (*ELbit*) considering regular mesh NoCs with square tiles.

Eq. (1) computes the dynamic energy consumed by a single bit traversing the NoC, from tile τ_i to tile τ_j, where η is the number of routers through which the bit passes. Let (x_i, y_i) and (x_j, y_j) be the coordinates of tiles τ_i and τ_j, respectively, then $\eta=(|x_j - x_i| + |y_j - y_i| + 1)$ is the number of routers of a given communication.

$$Ebit_h = \eta \times ERbit + (\eta - 1) \times ELbit \tag{1}$$

Let τ_i and τ_j be the tiles to which core c_a and core c_b are respectively mapped, represented by the tuple $(c_a \rightarrow \tau_i,\ c_b \rightarrow \tau_j)$. Thus, for the Cq (Definition 1), Eq. (2) is the dynamic energy consumed by communication. Furthermore, let $\check{C}$ be the set of all communicating pairs, then Eq. (3) expresses the total amount of *NoC dynamic energy consumption* (*EdNoC*) for Cq, computing the energy for all bits of all NoC communications.

$$Ep_{ab\eta(Cq)} = w_{ab} \times Ebit_\eta \tag{2}$$

$$EdNoC_{(Cq)} = \sum_{a=1}^{|C|} \sum_{b=1}^{|C|} Ep_{ab\eta(Cq)} \ \forall\ a \rightarrow b \in \check{C} \tag{3}$$

Cqe (Definition 2) considers not only bit quantity but also considers the bit flips, which implies to extend Eq. (1) by separating the energy consumed by consecutive bits that flip ($Ebit_\eta^F$) or not flip ($Ebit_\eta^N$). Therefore, when using Cqe, Eq. (4) is the dynamic energy consumed by a $c_a \rightarrow c_b$ communication ($Ep_{ab\eta}$), and Eq. (5) expresses *EdNoC*.

$$Ep_{ab\eta(Cqe)} = (f_{ab} \times Ebit_\eta^F) + (w_{ab} \times Ebit_\eta^N) \tag{4}$$

$$EdNoC_{(Cqe)} = \sum_{a=1}^{|C|} \sum_{b=1}^{|C|} Ep_{ab\eta(Cq*)} \ \forall\ a \rightarrow b \in \check{C} \tag{5}$$

Regarding the model Cqo, Eq. (6) computes the dynamic energy consumed by the qth packet of a $c_a \rightarrow c_b$ communication ($Ep_{abq\eta}$), and Eq. (7) gives the total amount of dynamic energy consumed on NoC operation (*EdNoC*), which takes into account the summation of all set of packets δ_t transmitted during the instant t.

$$Ep_{abq\eta} = w_{abq} \times Ebit_\eta \tag{6}$$

$$EdNoC_{(Cqo)} = \sum_{t=1}^{|T|} \sum_{q=1}^{|\delta_t|} Ep_{abq\eta} \ \forall\ a \rightarrow b \in \check{C} \tag{7}$$

The models Cqd and CqdPq (Definitions 4 and 5) employ the same of Eq. (6) to compute the dynamic energy consumed by the $q$thpacket of a $c_a \rightarrow c_b$ communication. Eq. (8) computes the *EdNoC* for both models. The innermost summation implements the quantity of packets transmitted for each pair of communicating cores.

$$EdNoC_{(Cqd,\ CqdPq)} = \sum_{a=1}^{|C|} \sum_{b=1}^{|C|} \sum_{q=1}^{|P|} Ep_{abq\eta} \ \forall\ a \rightarrow b \in \check{C} \tag{8}$$

Every router dissipates static and dynamic power (*PiRouter*) even in idle state. Let $|\Gamma|$ be the number of tiles, each one containing a

C = (A, B, E, F)

CRG = $<\{\tau_1, \tau_2, \tau_3, \tau_4\}, \{(\tau_1, \tau_2), (\tau_2, \tau_4), (\tau_4, \tau_3), (\tau_3, \tau_1)\}>$

mapping = $(\tau_1, B), (\tau_2, A), (\tau_3, F), (\tau_4, E)$

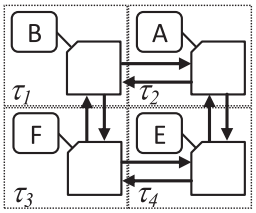

Fig. 4. Example of a 2×2 mesh NoC with an arbitrary mapping.

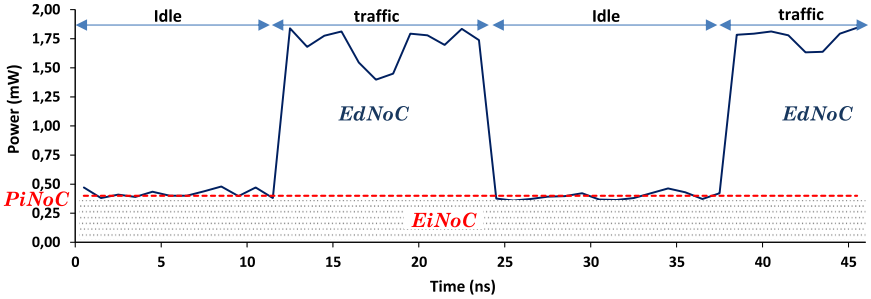

Fig. 5. Example of traffic periods for NoC energy calculation.

single router. Thus, Eq. (9) computes *static and idle power dissipation of a NoC* (*PiNoC*).

$$PiNoC = |\Gamma| \times PiRouter \tag{9}$$

Fig. 5 exemplify the router energy consumption on idle and traffic periods. For energy calculation, *EiNoC* is used as a minimum offset of energy consumed during the entire application execution time, instead of only the period when the router is idle. Thus, *EdNoC* computes only the increase of dynamic energy consumption caused by the traffic instead of the absolute value of energy consumption..

The *routing delay* and the *packet delay* compose the *total packet delay* of the wormhole routing algorithm. The routing delay is the time spent to create the communication path that is determined during the traffic of the packet header. The packet delay depends on the number of the remaining flits. Let n_{abq} be the number of flits of the *q-th* packet from core c_a to core c_b, which is obtained through the division of w_{abq} by the link width. Let λ be the clock cycle, and let t_r be the number of cycles needed for taking a routing decision, which is normally dependent on the arbitration mechanism. Furthermore, let t_l be the number of cycles to transmit a flit through a link. The routing delay ($d_{R\eta q}$) and the packet delay ($d_{P\eta q}$) of the qth packet from τ_i to τ_j, are represented in Eqs. (10) and (11), considering that a packet goes through η routers without contention.

$$d_{R\eta q} = (\eta \times (t_r + t_l) + t_l) \times \lambda \tag{10}$$

$$d_{P\eta q} = (t_l \times (n_{abq} - 1)) \times \lambda | a \rightarrow b \in \check{C} \tag{11}$$

Eq. (12) is the total packet delay ($d_{\eta q}$) obtained from the sum of $d_{R\eta q}$ and $d_{P\eta q}$.

$$d_{\eta q} = (\eta \times (t_r + t_l) + t_l \times n_{abq}) \times \lambda | a \rightarrow b \in \check{C} \tag{12}$$

Cq and Cqe may only estimate the *application execution time* (*texec*) as a minimum time spent on cores communication, which is achieved considering n_{abq} containing all flits transmitted from core c_a to core c_b in Eq. (12). Then, *texec* is equal to the greatest $d_{\eta q}$ of all cores communications. The model Cqo enables to estimate *texec* by the time tag of the last set of packets ($t_{|T|}$) added to the d_η of the latest packet. The computation of all $d_{\eta q}$ of all messages enables to obtain the *texec* of Cqd. Finally, CqdPq improves the *texec* estimated with Cqd by computing all t_{iq}.

Eq. (13) computes EiNoC, which represents the static and idle energy consumption of a NoC during texec.

$$EiNoC = PiNoC \times texec \tag{13}$$

Finally, Eq. (14) expresses the overall energy consumption on the NoC (*ENoC*) for all defined MoCs.

$$ENoC = EdNoC + EiNoC \tag{14}$$

Since Cq, Cqe and Cqd do not carry processing time information, which is normally the most significant portion of time that composes *texec*; these models enable to compute only the *EiNoC* originated from communications. Then, for many applications, the *EiNoC* calculated by these models is negligible compared to *EdNoC*. Consequently, only Cqo and CqdPq provide suitable *EiNoC* estimations.

5. MoC influence on mapping algorithm

This section evaluates the MoC influence on the mapping algorithms, regarding dynamic energy and execution time minimization. Although this section describes the core-mapping problem, the task mapping has similar formalization.

For all MoC evaluated, two parts divide the mapping algorithms: (*i*) *ObjFn* is the internal part representing the mapping objective that is an algorithm dependent on the target architecture and MoC; and (*ii*) an external part that encloses *ObjFn*, which is an algorithm independent of the application and the target architecture (e.g., Simulated Annealing). Since the external part is not the focus here, the implementation of these algorithms is not discussed.

Although this work addresses the minimization of dynamic energy consumption and execution time, the *ObjFn* is simplified using only the total energy consumption, as in Eq. (14). Two parcels that represent such requirements compose this equation: (i) the *dynamic energy* consumption of the NoC - Eqs. (3), (5), (7) and (8); and (ii) the *static and idle energy consumption* that is associated to the application execution time – Eq. (13). Two parts split *ObjFn*: (*i*) a MoC dependent algorithm and (*ii*) a target architecture dependent algorithm, called *NoCAlg*. This disjoint implementation aims to simplify the understanding of each algorithmic problem.

5.1. Objective functions of Cq and Cqe

Fig. 6 describes the pseudo-code of *Cq_ObjFn* that is the *objective function* for both, Cq and Cqe..

For each mapping, both objective functions start setting the variable **mapCost** to zero, which computes the total mapping cost of a given application description over an arbitrary core-mapping. In fact, the **mapCost** for Cq and Cqe is the dynamic energy consumption described in Eqs. (3) and (4), respectively.

Two nested loops implement both *Cq_ObjFn*. The outer loop (lines 2–9) searches for source vertices and the inner loop (lines 3–8) searches for all communications of this source vertex. Each communication is an edge of the graph representing Cq or Cqe. Source and target vertices positions (that may change in each new mapping) and the communication weight are input parameters for *Cq_NoCAlg*. This algorithm returns the estimations of energy consumed (**mc**) by all component involved in the source to target communication and the time spent (**time**) to transmit all flits.

The energy consumption computed on *Cq_NoCAlg* is the Eqs. (2) or (4), for Cq or Cqe, respectively. Additionally, Eq. (12) computes the **time** of both MoCs with n_{abq} equal to w_{ab} divided by the NoC flit size.

Cq_NoCAlg implies a very optimistic application execution time (*texec*) by considering it as the bigger value among all communications. Then, when the algorithm finishes the loops execution, *texec* is equal to the variable **appTime** and *Cq_EiNoC* is the Eq. (13).

Cq model allows good estimations of dynamic energy consumption since it enables computing the bits volume, which is responsible for the

```
Cq_ObjFn:
1  mapCost, appTime ← 0
2  for(s ← firstSourceVertex; s ≠ null; s ← s.next) {
3     for(t ← s.firstTargetVertex; t ≠ null; t ← t.next) {
4        (mc, time) ← Cq_NoCAlg(s.x, s.y, t.x, t.y, t.weight)
5        mapCost ← mapCost + mc
6        if(time > appTime)
7           appTime ← time
8     }
9  }
10 return mapCost + Cq_EiNoC(appTime)
```

Fig. 6. Pseudo-code of Cq and Cqe objective functions.

```
Cqo_ObjFn:
1   mapCost, appTime ← 0
2   for(n ← firstTimeStamp; n ≠ null; n ← n.next) {
3     for(m ← firstElement; m ≠ null; m ← m.next) {
4       (mc, time) ← Cqo_NoCAlg(n.timeTag, m)
5       mapCost ← mapCost + mc
6       if(time > appTime)
7         appTime ← time
8     }
9   }
10  return mapCost + Cqo_EiNoC(appTime)
```

Fig. 7. Pseudo-code of Cqo objective function.

calculus of the main parcel of energy consumption. Besides, Cqe enables better estimations due to the bit model detailing.

5.2. Objective function of Cqo

Fig. 7 depicts the *objective function* for Cqo mappings (*Cqo_ObjFn*), which starts each mapping setting to zero the total mapping cost (**mapCost**). In fact, **mapCost** is the dynamic energy consumption described in Eq. (6). Two nested loops implement *Cqo_ObjFn*. The outer one (lines 2–9) searches for the list of communications associated to a given time stamp. The inner loop (lines 3–8) searches for all communications of the searched time stamp (**n.timeTag**). Time stamps, as well as, **m** with source and target vertices places and the communication weight are input parameters for the *Cqo_NoCAlg*..

Fig. 8 shows *Cqo_NoCAlg* that is the target architecture dependent algorithm for Cqo. *Cqo_NoCAlg* returns **commCost** and **t**, which are the packet communication cost based on the energy costs of each NoC component and the time when the packet reaches the target core, respectively. The first loop (lines 3–6) computes the packet traffic from the X position of source core (**m.xSource**) to the X position of target core (**m.xTarget**). The second loop (lines 7–10) computes the remaining communication, i.e., the packet traffic from the Y position of source core (**m.ySource**) to the Y position of target core (**m.yTarget**)..

ResourceCost function computes the energy consumed by links and routers where the packet passes through, which receives as parameters: (i) the resource current position (**x**, **y**), (ii) the communication containing source and target cores (**m**), (iii) the related communication weight (inside of **m** variable), and (iv) the time tag (**t**). The algorithm adds each energy cost to **commCost**, returning the sum of all costs to *Cqo_ObjFn*, aiming to compute the **mapCost**, and **commCost** is the Ep_{abqn} value of Eq. (6). Further, since *Cqo_NoCAlg* returns the time **t** of a message, the greatest **t** among all messages of *Cqo_ObjFn* (computed in lines 6–7 of Fig. 7) is the application execution time (*texec*), and *Cqo_EiNoC* is the Eq. (13).

```
Cqo_NoCAlg:
1  commCost ← 0
2  t ← n.timeTag
3  for(x ← m.xSource; m.xTarget ≠ x; x ← x + 1) {
4     commCost ← commCost + ResourceCost(x, y, m, t)
5     t ← ResourceTime(x, y, t)
6  }
7  for(y ← m.ySource; m.yTarget ≠ y; y ← y + 1) {
8     commCost ← commCost + ResourceCost(x, y, m, t)
9     t ← ResourceTime(x, y, t)
10 }
11 return (commCost, t)
```

Fig. 8. Pseudo-code of Cqo NoC algorithm.

Finally, *Cqo_ObjFn* allows good estimations of dynamic energy consumption since it enables computing the bits traffic. Moreover, *Cqo_NoCAlg* can reasonably describe *texec*, enabling to estimate *EiNoC*.

5.3. Objective function of Cqd

The objective function of Cqd (*Cqd_ObjFn*) takes into account applications represented by the weight and by the dependence of the communication. The dependence knowledge allows estimating packet contentions of independent messages that concur for the same communication resource at the same time. This work employs an auxiliary structure called *Communication Dependence List* (CDL), which is a list of lists representing levels of dependences among messages to achieve an efficient manipulation of messages dependencies. Vertices on the same level in CDL are independent and may depend only on vertices of lower levels.

CDL is produced by applying to Cqd an algorithm similar to ASAP scheduling [46] with following rules: (i) all messages scheduling happens as soon as their dependences are solved; (ii) all independent messages that are scheduled at the same time perform a dependence level and are placed on the same list. Additionally, there is no messages order inside a given dependence level; (iii) to each message is associated a list of all messages that this one is dependent. For instance, the *dpList* of p_{AF4} is $\{p_{BF1}, p_{AB2}, p_{EA3}\}$; (iv) each new dependence level is associated to the previous dependence levels by an ordered list; (v) the algorithm finishes when all Cqd paths reach *End* vertex. CDL enables to minimize *Cqd_ObjFn* complexity and speed up the algorithm performance.

Fig. 9 displays the synthetic example of Cqd shown in Fig. 3(d) and its corresponding CDL..

Fig. 10 describes two nested loops that implement *Cqd_ObjFn*. The outer one (lines 2–10) searches for the list of CDL levels, starting from *Level_1* vertex until reach *null*. The inner loop (lines 3–9) searches for all messages of a given level, starting from the vertex pointed by *pList* until reach *null*. Each loop iteration calls *SearchForStartTime* and then *Cqd_NoCAlg* functions. *SearchForStartTime* computes the initial time ($\mathbf{t_i}$) of **m**, searching, within the list of dependent messages, the highest message delivery time that occurs when the last flit of all dependent messages reaches its target core. *Cqd_NoCAlg* returns the mapping cost (**mc**) of the message **m** and its final time ($\mathbf{t_f}$) that are used to compute *EdNoC* (**mapCost**) and *EiNoC* (*Cqd_EiNoC*)..

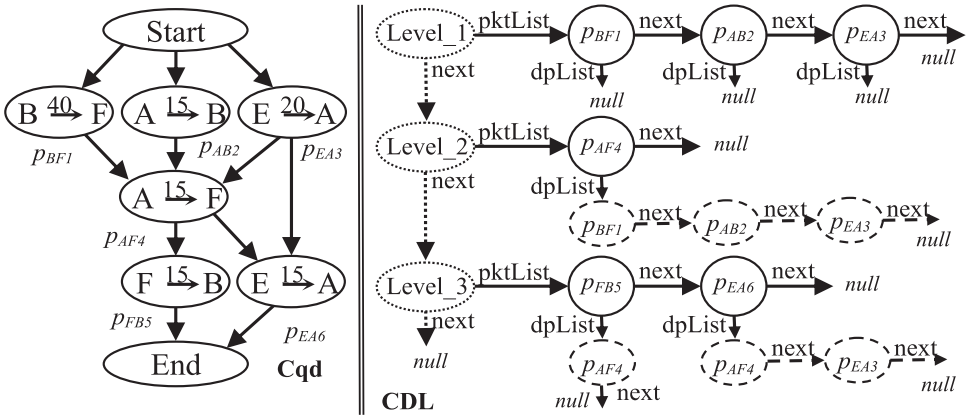

Fig. 9. The Cqd synthetic example of Fig. 3(d) and the matching CDL. In CDL, the dotted arrows and ellipses represent the list of dependence levels; Continuous arrows and circles represent lists of messages of the same level; Continuous circle messages depend on the lists of messages represented by dashed arrows and ellipses.

```
Cqd_ObjFn:
1  mapCost, appTime ← 0
2  for(n ← Level_1; n ≠ null; n ← n.next) {
3     for(m ← n.pList; m ≠ null; m ← m.next) {
4        t_i ← SearchForStartTime(m)
5        (mc, t_f) ← Cqd_NoCAlg(m, t_i)
6        mapCost ← mapCost + mc
7        if(t_f > appTime)
8           appTime ← t_f
9     }
10 }
11 return mapCost + Cqd_EiNoC(appTime)
```

Fig. 10. Pseudo-code of Cqd objective function.

```
Cqd_NoCAlg:
1  commCost ← 0
2  time ← t_i
3  for(x ← m.xSource; m.xTarget ≠ x; x ← x + 1) {
4     (ec, time) ← ResourceCost(x, y, m, time)
5     commCost ← commCost + ec
6  }
7  for(y ← m.ySource; m.yTarget ≠ y; y ← y + 1) {
8     (ec, time) ← ResourceCost(x, y, m, time)
9     commCost ← commCost + ec
10 }
11 m.endTime ← time
12 return (commCost, m.endTime)
```

Fig. 11. Pseudo-code of Cqd NoC algorithm.

Fig. 11 depicts *Cqd_NoCAlg* algorithm that uses the information of parameter **m** (i.e., the message communication weight, the source and target vertices and a list of messages that **m** is dependent) to estimate possible packet contentions. Dependent packets never concur for the same resource. On the other hand, independent packets with the same dependence level are candidates to concur for some resources at the same time. The algorithm tries to avoid mappings where independent packets concur for NoC resources increasing the **mapCost**..

Cqd_NoCAlg works pessimistically, i.e., if two or more packets can concur for the same resource at the same time, the algorithm considers that concurrences will happen. *ResourceCost* function implements this functionality according to the resource position (**x**, **y**), the start **time** of using the resource and the message (**m**), and returns the energy consumed (**ec**) and the **time** augmented by the resource delay. *Cqd_NoCAlg* starts clearing the dynamic energy consumption (**commCost**). The *ResourceCost* annotates the **time** of a message in all resources that it passes. Thus, the last resource is annotated with last instant of time that the message is inside the NoC; and the algorithm attributes **time** to the greatest message delivery time (**m.endTime**).

Fig. 12 shows the operation of *ResourceCost* function with the same example of Fig. 9, taking into account an arbitrary mapping {(A→τ_1, B→τ_2, E→τ_3, F→τ_4)}..

Each router and each link contains a list of incoming packets with the corresponding start and ending time (in clock cycles). The example considers t_r =t_l =1 clock cycle, and 1 bit-length of flit; thus Eq. (12) may be rewrote by (2×(|x_j - x_i| +|y_j - y_i|) +2+w_{abq}) to compute the entire packet delay without contentions. For instance, the total packet's delay for p_{BF1} is 44 clock cycles (2×(|1-1|+|1-0|)+2+40).

The transmission of each packet starts as soon as all its dependences are solved. For example, p_{AF4} depends on p_{BF1}, p_{AB2} and p_{EA3} that have as **endTime**, 44, 19 and 24, respectively. Thus, the transmission of p_{AF4} starts in cycle 45. In addition, when two or more packets concur for the same resource at the same time, the algorithm annotates the first scheduled packet without considering contentions,

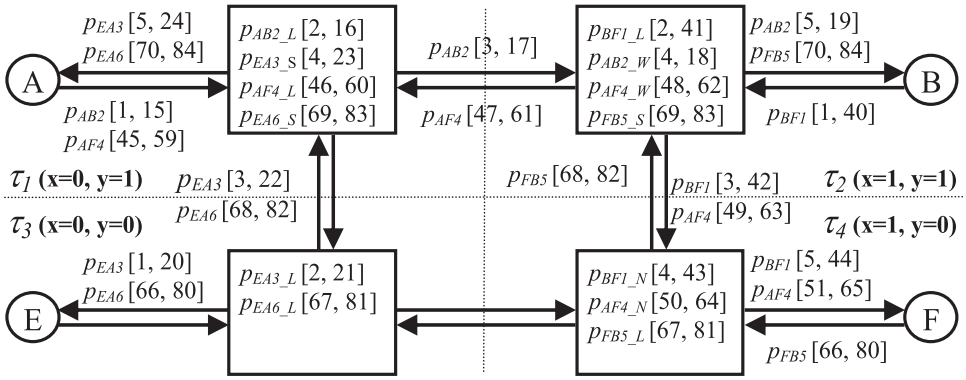

Fig. 12. An arbitrary mapping for the application described in Fig. 9 and the corresponding ResourceCost taking into account the Cqd model.

```
CqdPq_ObjFn:
1   mapCost, appTime ← 0
2   for(n ← Level_1; n ≠ null; n ← n.next) {
3     CDL_Reordering(n)
4     for(m ← n.pList; m ≠ null; m ← m.next) {
5       t_i ← SearchForStartTime(m)
6       (mc, t_f) ← CqdPq_NoCAlg(m, t_i)
7       mapCost ← mapCost + mc
8       if(t_f > appTime)
9         appTime ← t_f
10    }
11  }
12  return mapCost + CqdPq_EiNoC(appTime)
```

Fig. 13. Pseudo-code of CqdPq objective function.

and the time of the scheduled packet postpones subsequent packets.

Cqd_ObjFn is not appropriated to compute *texec* since Cqd does not take into account the processing time, but only communication time, and consequently *texec* is underestimated. On the other hand, Cqd enables to avoid contentions, reducing the dynamic energy consumption, mainly by the reduction of buffers occupation.

5.4. Objective function of CqdPq

The objective function of CqdPq (*CqdPq_ObjFn*) improves *Cqd_ObjFn* by exploring the processing time knowledge to estimate the intervals that packets are occupying the NoC. Similar to the *Cqd_ObjFn* approach, this algorithm starts with CDL, but inside each level, it orders all messages according to their processing time. Fig. 13 describes the two nested loops that implement *CqdPq_ObjFn*..

The outer loop in Fig. 13 (lines 2–11) searches for the list of messages of CDL, starting from *Level_1* vertex until reach *null*. The inner loop (lines 4–10) starts reordering messages inside the level using *CDL_Reordering* algorithm and then searches for all messages of a given level, starting from the vertex pointed by *pList* until reach *null*. Each loop iterations calls the algorithm that returns the mapping cost used to compute *EdNoC*, and the end time of the message, which is used to compute *EiNoC*.

CqdPq provides the processing time that precedes each message. This information allows the *CDL_Reordering* to reorder messages producing reliable time estimations. The message reordering is performed considering the time that their dependences are solved added with the processing time that precedes the transmission of the message. Although the parameter **m** of the model CqdPq carries more information than the equivalent one of Cqd, *CqdPq_NoCAlg* is practically the same of *Cqd_NoCAlg*. Thus, its implementation was omitted.

Fig. 14 shows the same example of Fig. 12 with the same input parameters, aiming to compare the implementations of *ResourceCost* of models Cqd and CqdPq..

CqdPq_ObjFn improves *Cqd_ObjFn*, because it considers computation time, eliminating the pessimistic evaluation. Hence, *CqdPq_ObjFn* enables to find better mappings, due to the accurate knowledge of packet contention that could imply larger *texec* and increases *EiNoC*, which is computed by *CqdPq_EiNoC*(**appTime**).

5.5. Analysis of timing complexity of MoCs

Two nested loops performing $|C|$ iterations implement the *Cq_ObjFn* function; thus, the time complexity of *Cq_ObjFn* is $O(|C|^2)$. However, practical examples show that few cores comprise the inner loop, reducing the algorithm execution time and justifying its good performance in experimental results. Additionally, since all messages exchanged by the same pairs of cores are computed into single communications, this algorithm is not susceptible to the variation of the quantity of messages, but only to the quantity of communicating cores.

Two nested loops implement the *Cqo_ObjFn* function, where the outer one is dependent on ($|T|$) and the inner one is dependent on the number of messages dispatched by time tag event. Nevertheless, the quantity of iterations caused by the execution of both loops is exactly the total quantity of the application's messages; thus, the time complexity for *Cqo_ObjFn* is $O(|M|)$. Hence, the algorithm is indirectly susceptible to the application execution time, since the more time the application is executed, the more messages are dispatched.

The *Cqd_ObjFn* function involves two nested loops that search messages within each level of CDL. This search implies exactly $|M|$ iterations since it corresponds to the total quantity of the application's messages. Additionally, each one of these iterations calls the *SearchForStartTime* function, which searches for each message up to $|M| - 1$ dependent messages. The inclusion of *SearchForStartTime* into the inner loop implies that the time complexity of *Cqd_ObjFn* is $O(|M|^2)$, showing that the objective functions above analyzed are much less complex. Even so, real applications show that any message is directly dependent on a few other messages, which reduces this complexity.

Regarding only timing complexity aspects, the *CqdPq_ObjFn* function differs from *Cqd_ObjFn* on the outer loop that includes the *CDL_Reordering* function. Although the complexity of *CDL_Reordering* is $O(|M|)$, the time complexity of *CqdPq_ObjFn* is $O(|M|^2)$, because the joint execution of the nested loops implies exactly $|M|$ iterations, and the *CqdPq_ObjFn* function never reaches the complexity $O(|M|^3)$.

6. Experimental results

The following four subsections compose this section: (i) the acquisition of input applications; (ii) the evaluation of the achieved mappings quality; (iii) the electrical calibration of the mapping tool; and (iv) the achieved results.

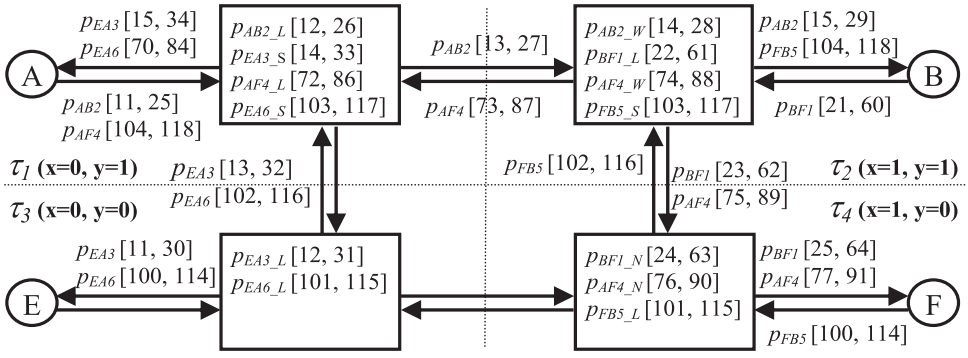

Fig. 14. Arbitrary mapping of application shown in Fig. 9 and a corresponding *ResourceCost* operation taking into account the CqdPq model.

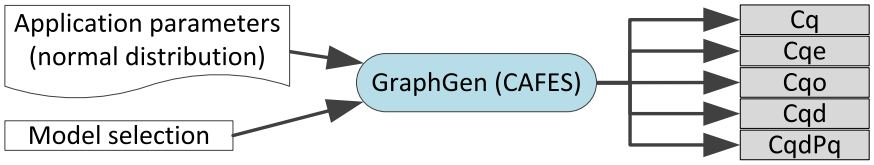

Fig. 15. GraphGen tool for synthetic application generation.

6.1. Generation of synthetic applications

Fig. 15 shows *GraphGen* (a customizable generator graph available at CAFES framework [47] with a set of input parameters, which allow modeling application classes that are used as input of simulation experiments. The *GraphGen* parameterization is probabilistically defined according to a normal distribution. For example, to set the traffic injection rate of each core, the designer provides the mean, the standard deviation, the maximum and minimum quantities, and then the *GraphGen* creates probabilistic traffic injections. The core traffic distribution is the same for all MoCs to characterize the same synthetic application..

6.2. Embedded applications modeling

The experiments also encompass four embedded applications: (PBX) a digital privative branch exchange; (MMS) a multimedia system; (RBG) a distributed algorithm for Romberg integral calculus; (IRS) a parallel system for object recognition through image segmentation. The following features characterize the embedded applications: the *application algorithm description* (Appl), the *target architecture* (Arch), and the *input traffic pattern* (InTr). Table 3 groups these features into four sets, according to the next dependences: (i) number of tasks (Appl dependence); (ii) the number of cores and tile area (Arch and Appl dependences); (iii) bit flips (InTr dependence); and (iv) number of communication channels, communication quantity average, and traffic injection rate (Appl, Arch and InTr dependences). For instance, the number of communication channels is dependent on: (i) Appl, since the application algorithm contains channels implemented by send/receive functions; (ii) Arch, since the tasks grouping into the same core do not perform communications through NoC channels; and (iii) InTr, since a given logical condition of the application algorithm enable (or not) to send or receive a message.

Having the embedded applications in C or Java, a set of tools provided by CAFES enables to model the application following a set of steps shown in Fig. 16..

The application algorithm provides the number of parallel tasks, and the designer chose the number of cores to fulfill the computation requirements through tasks partitioning. The number of communication channels is attained after grouping tasks (for future task-core mapping), since communicating tasks that are not grouped require a physical path to implement the communication channel inside the network.

Application modeling starts manually by inserting the API functions (step 1), defined in the *ModGen* (Model Generator) package of CAFES [47], into the application code, which enables to describe the application behavior according to the selected MoC. For example, Cq requires only computing the number of bits transmitted between cores. Thus, the designer has only to link the API functions to *send/receive* commands, informing the source and target cores, and the number of transmitted bits. The API links counters and timers to each send/receive, which enables to achieve the bit flip percentage, the average communication quantity, and the traffic injection rate during the application execution.

The second step is the application simulation, where the *ModGen* receives an embedded application and generates output files containing the application described according to the MoC. *ModGen* automatically generates output files containing the Cq, Cqe and Cqo application models. However, CqdPq and Cqd extraction requires an extra task to attain the partial dependence of each communication.

Fig. 16 shows a dashed arrow that represents manual dependence extraction. Additionally, the *ModConv* tool can automatically extract Cqd having CqdPq as input model (step 3).

Table 3
Description of four embedded applications.

Features			PBX	MMS	RBG	IRS
(i)	**Appl**	Number of tasks	24	34	180	48
(ii)	**Arch**	Number of cores	5	16	55	24
	Appl	Tile area (mm^2) – square format	100	25	16	9
(iii)	**InTr**	Bit flip (on average)	20.3%	47.8%	41.6%	3.8%
(iv)	**Appl** **Arch**	Number of communication channels	12	28	110	134
	InTr	Average communication quantity (bytes)	2334	22,135	35	30,827
		Traffic injection rate (on average)	2.7%	10.2%	31.2%	25.7%

Legend: Appl – representing a feature dependent on the application.
Arch – representing a feature dependent on the target architecture.
InTr – representing a feature dependent on the input traffic.

6.3. Electrical parameters and mapping calibration

The *Mapper* was calibrated to simulate the communication behavior, and energy consumption of Hermes mesh NoC [48], with XY deterministic routing, wormhole, 16-bit flits, 4-depth input buffer, credit-based flow control and FIFO arbitration policy for input packet selection. The following scenarios were applied: (i) *idle scenario* – composed by only periods where the NoC has been idle, without receiving any communication; and (ii) *traffic scenario* – consisting of time intervals where NoC received several sets of synthetic traffic, with controlled bit flip variations. The *idle scenario* extracts the parameter *PiNoC*, which was obtained as an average power dissipation of one router. The *traffic scenario* allows calculating the energy consumption increases against the traffic injection load and the traffic pattern. The influence of injection rate, packet size, and bits flip enables to estimate the *ERbit*, *ELbit*, $ERbit^F$, $ERbit^N$, $ELbit^F$ and $ELbit^N$.

Mapper tool calibration considers electrical and timing parameters, which are influenced by the credit-based flow control and the arbitration policy. The credit-based flow control takes t_l =1 clock cycle, whereas the policy of router arbitration takes t_r =3 clock cycles, on average.

Having as target architecture a 3×3 NoC mesh with 10 mm×10 mm square tiles, which was synthesized to 65 nm CMOS technology with 1.0 V and 1 GHz of operation frequency, Cadence RTL Compiler allowed to estimate the following electrical parameters: *ERbit* =1.35pJ, *ELbit* =0.43pJ, $ERbit^F$ =1.63pJ, $ERbit^N$ =0.57pJ, $ELbit^F$ =0.57pJ, $ELbit^N$ =0.02pJ and *PiNoC* =0.53 mW. Note that these parameters, although they have been calibrated for a 3×3 NoC, they are replicable for any NoC size since the electrical features of the router and tile characteristics are maintained.

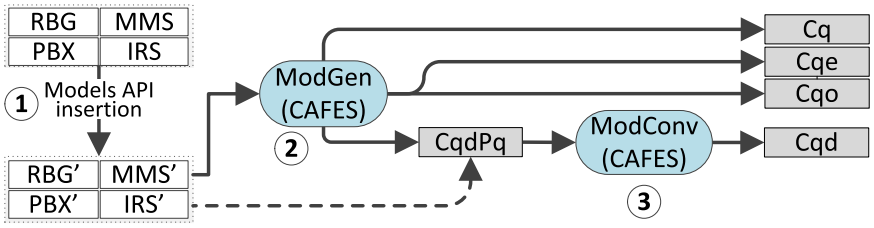

Fig. 16. Extraction flow for embedded application modeling.

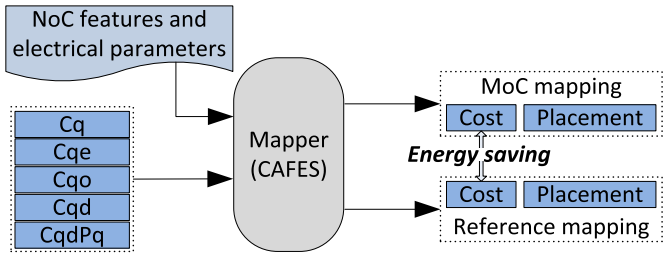

Fig. 17. Energy saving results evaluation.

6.4. Method applied on mapping results evaluation

The *Mapper* tool carries out the mapping quality assessment by entering the MoC, the NoC size and the electrical parameters specifying the NoC technology (Fig. 17). The tool performs each MoC mapping algorithm returning the mapping cost (i.e., energy consumption) and the cores placement that minimizes this cost..

The results are presented using a *mapping reference* to evaluate the MoCs potentialities and limitations. The algorithms that derive reference mappings are equal to those used for obtaining best mappings. They differ only in the cost function related to the external algorithm, which is inverted to obtain mappings that maximize the energy consumption. Thus, for each experiment, the mapping reference presents the highest mapping cost found.

6.5. Mapping results evaluation

All experiments show comparisons of Cq, Cqe, Cqo, Cqd and CqdPq models on static and dynamic energy consumption minimization for several synthetic and some embedded applications.

The experiments explore three parameters: (i) NoC size indicating the number of application cores and the number of communication paths; (ii) application connectivity degree implying the communication channels quantity among application cores. Given that 0% of connectivity means no communication and 100% of connectivity means that all cores send messages to all others; and (iii) traffic injection rate that is the average of traffic injection of all cores, which is an indicative if an application is I/O or CPU bounded. For all next graphics of all experiments, the value of each point represents an average of energy consumption (i.e., dynamic, static and idle) of all applications. Values are presented as a percentage of the difference between a reference value (refer to Section 6.4) and the calculated average.

Fig. 18 and Fig. 19 show the first set of experiments, exploring the influence of NoC size for the MoCs on energy consumption. The *GraphGen* produces synthetic applications having on average 15% of connectivity, 10% of injection rate, and traffic pattern modeled to perform 50% of bit flips (e.g., 1100110011001100). Experiments used as target architecture the following NoCs: 3×4 (12 tiles), 5×6 (30 tiles), 8×8 (64 tiles), 10×10 (100 tiles) and 12×12 (144 tiles)...

Fig. 18 shows that Cqe is the MoC that most save dynamic energy consumption for all NoC sizes, which is justified by the high degree of bit flips that is captured by this model only. The communication dependence is also important because it avoids the contention of packets enabling to minimize the use of buffers. This aspect is observed on the results of Cqd and CqdPq, when both are compared with Cqo and Cq.

Fig. 19 shows how the MoCs capture the influence of NoC size on minimization of static and idle energy consumption. Only Cqo and CqdPq can capture core computation time with accuracy, and hence, better estimation of static and idle energy consumption. Due to contentions avoidance, Cqd minimizes runtime allowing slightly static and idle energy consumption reduction. Finally, Cq and Cqe are inappropriate to fulfill this requirement.

For all mappings algorithms, the first set of experiments points out that the more NoC size increases, the less energy saving is achieved. However, it happens due to the external mapping algorithm that minimizes its efficiency with the increase of the NoC size. Therefore, the worst mapping obtained, which is used as a reference, is not necessarily the worst possible mapping, and the best mapping found in each MoC mapping exploration is not necessarily the best possible mapping. Consequently, using CPUs with more computational power

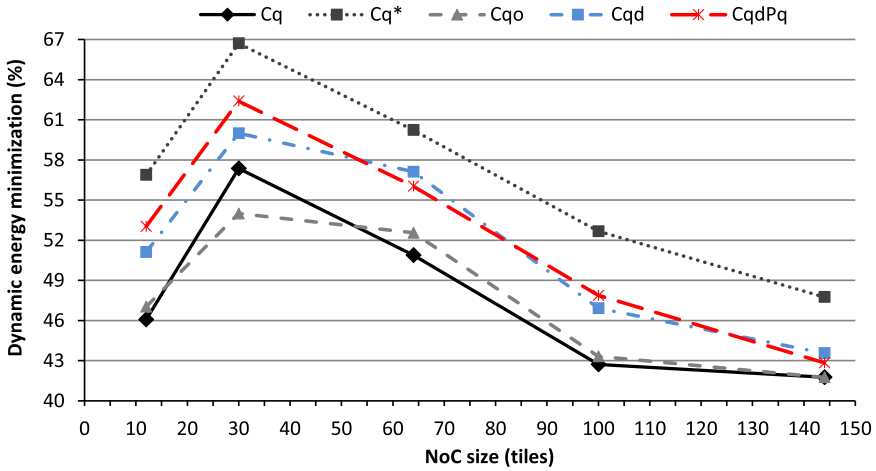

Fig. 18. Influence of NoC size on dynamic energy saving.

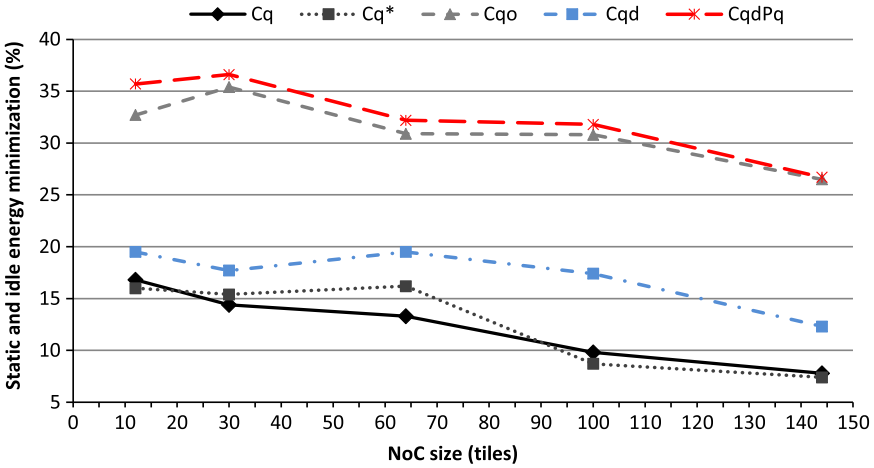

Fig. 19. Influence of NoC size on static and idle energy saving.

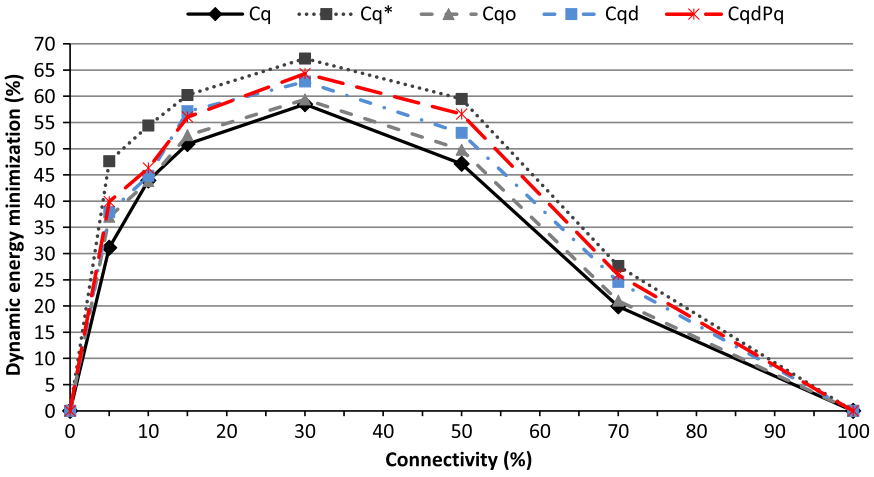

Fig. 20. Influence of application connectivity degree on dynamic energy saving.

probably will save more energy independent of the MoC mapping algorithm.

Fig. 20 and Fig. 21 encompass the second set of experiments that show the results of synthetic applications with a variation on connectivity degree. All experiments target 8×8 NoCs, an average injection rate of 10% and traffic modeled to perform 50% of bit flips, on average. This set of experiments evaluates the following connectivity degrees: 0%, 5%, 10%, 15%, 30%, 50%, 70% and 100%...

Fig. 20 demonstrates that, when connectivity degree increases, the cores mapping has no longer high influence on minimization of dynamic energy consumption, reducing the importance of a good MoC choice. Additionally, 0% of connectivity implies no communication and consequently no energy consumed on the communication architecture. Furthermore, if the application has 100% of connectivity and for all communication the same volume of communication and time of packet injection then, regardless of the mapping, the NoC will

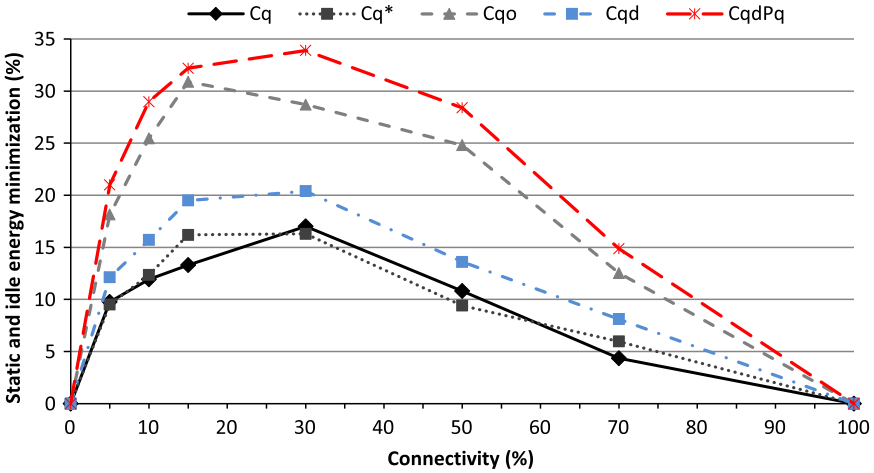

Fig. 21. Influence of connectivity degree on static and idle energy saving.

consume the same energy (i.e., the cost of energy will be equal for all mapping algorithms). On the other hand, when the application has around 30% of connectivity degree, the mappings reach the maximum of energy consumption reduction for all mapping algorithms, and for applications with less than 70% of connectivity degree, the choice of MoC implies significant minimization of dynamic energy consumption.

The bit flipping influences more the dynamic energy consumption on the links than on the routers. This fact is most evident on the chosen target architecture, which considers relatively long links between routers (10 mm). Therefore, similarly to the previous experiment, Cqe model, which capture bit flip information, provides better mappings. However, smaller tiles and/or smaller rates of bit flip may significantly reduce this advantage.

Fig. 21 shows that the selection of a suitable MoC implies significant minimization on static and idle energy consumption. Furthermore, the mapping algorithms, whose underlying MoC carries time information, may estimate these energies accurately and in consequence provide a better mapping. Furthermore, for more than 30% of the application connectivity degree, the importance of an appropriate MoC selection is reduced linearly.

Fig. 22 illustrates the third set of experiments, which explores the effect of traffic injection rate on dynamic energy consumption reduction..

This set of experiments employs an 8×8 NoC mesh to execute synthetic applications with 15% of connectivity degree and traffic pattern with 50% of bit flips, on average. This set of experiments evaluates the following traffic injection rates: 5%, 10%, 20%, 40%, 80% and 100%. For instance, 100% of traffic injection rate is acquired when all cores are inserting flits in all clock cycles, which represents a high I/O bounded application. Note that for this third set of experiments are not displayed static power consumption results or the energy consumption in idle since the differences between the MoCs are insignificant.

Independent of the traffic injection rate, CqdPq is the model that more minimizes dynamic energy, since it captures accurate traffic injection time, which enables to reduce packet contentions. In scenarios of low traffic injection rate, the number of pessimistic mappings exploited by Cqd is relatively small if compared with the number of mappings that requires low energy consumption. This behavior allows to the mapping algorithm to choose low energy consumption mappings (those that approximate high communicating cores/tasks) and yet reducing contentions. As the traffic injection rate increases, the number of pessimistic mappings also grows concerning the possible mappings of low energy consumption. Thus, the mapping choice tends to be less efficient.

The time tags captured by Cqo are the instants of time of packets transmission during the application execution, but without considering the tasks/cores placement. Consequently, the time of packets transmission obtained after mapping can differ from the ones provided by Cqo. For low injection rates, this effect is not significant because the communications are sparse, implying low quantity of packets into the NoC producing contentions. Thus, Cqo allows better mappings of CPU bounded applications than I/O bounded applications.

The models Cq and Cqe cannot capture changes in injection rate, and their mapping algorithms tend to produce mappings that disregard this variation. However, Cqe enables to capture the bits flip that is one of the most important information for dynamic energy computation, enabling to produce better mappings than the ones produced with Cq algorithms.

Fig. 23 illustrates the fourth set of experiments comprising four embedded applications detailed in Table 3.

This last figure shows that applications modeled with CqdPq enable to produce mappings that most save energy, followed by Cqo mappings, which obtain energy results around 8% less efficient than the ones achieved with CqdPq mappings. Cqd and Cqe enable mappings that consume about 13% more energy than the mappings achieved with CqdPq, on average. Finally, Cq provides mappings with the smaller amount of energy saving (22% less efficient, on average).

Using Cqe to model MMS and RBG applications, the achieved mappings are better than the ones achieved with Cqo. This fact occurs due to the high bit flip rate of the input traffic and the relatively high tile size, which increases the influence of *ELbit* on the calculation of total energy consumption. However, with low modeling cost, the number of bits flip can be added to another MoC, like CqdPq, generating a new, and potentially more efficient, MoC. On the other hand, due to the complexity of modeling applications with dependence information (e.g., Cqd and CqdPq) and in counterpart the facility to model applications with both order and quantity information (e.g., Cqo), we conclude that a promising MoC is achieved with the combination of the Cqo characteristics (order) with bits flip information (quantity).

7. Conclusions

This paper addresses discussions about mapping applications onto NoC-based SoCs. It primarily focuses on application characteristics aiming to provide a novel classification scheme, analyzing a comprehensive set of Models of Computation (MoCs) to capture applications characteristics and to enable the mappings algorithms for selecting efficient mappings. The experimental results indicate advantages and limitations of each MoC, although they are dependable on factors such as the communication architecture and some characteristics of the

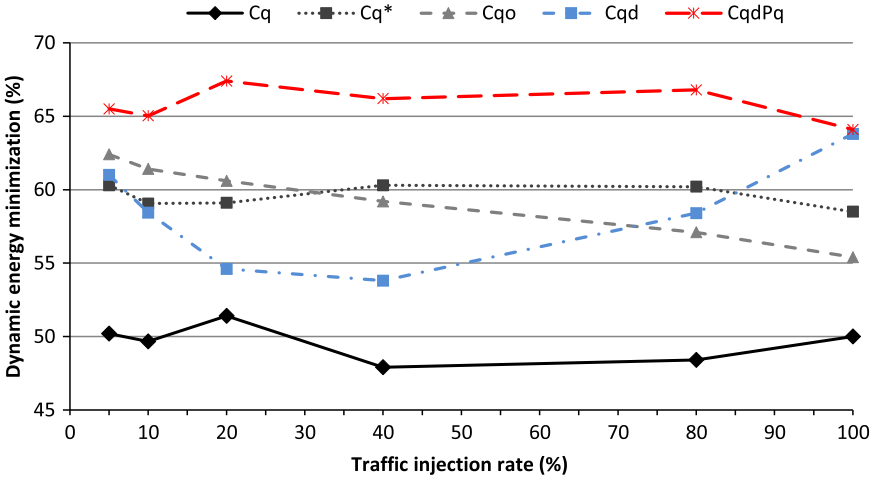

Fig. 22. Influence of traffic injection rate on dynamic energy saving.

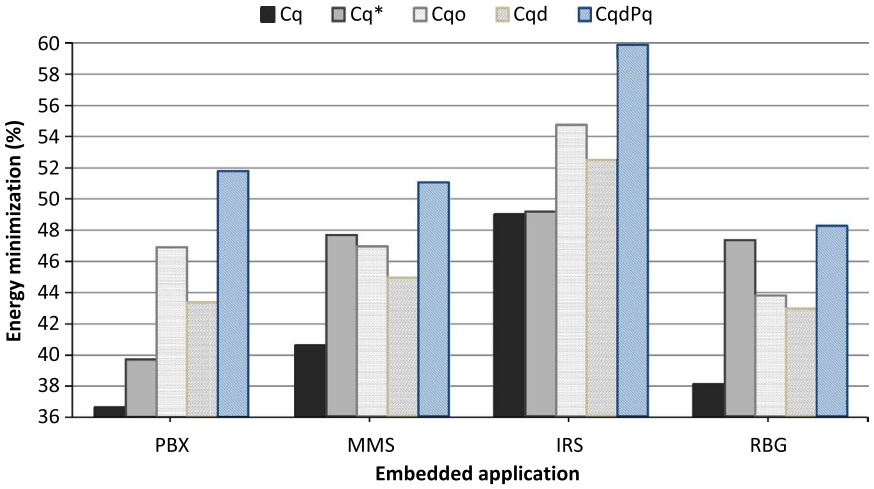

Fig. 23. Minimization of static and dynamic energy consumption for a set of embedded applications.

chosen applications.

Traffic features influence on the dynamic energy consumption, mainly due to the bit flip that implies loading and unloading of capacitances of buffers and links. Specifically, the analysis of the communication order enables to capture packets collision, affecting issues mostly related to the instantaneous power dissipation, and dynamic energy consumption on the buffers of routers.

This paper shows that the communication volume essentially influences the dynamic energy consumption due to the time necessary to transmit all packets through NoC resources. It also influences on the application execution time, especially for I/O bounded applications. Nevertheless, the processing time of each core mainly influences on the (i) estimation of application execution time, on the (ii) static energy consumption, and on the (iii) energy consumed by the router circuits that operate in idle periods (i.e., periods without communication), being even more relevant for CPU bounded applications.

This paper provided a thorough comparison of MoCs leading us to draw some major conclusions. Firstly, models that are suitable for reasonable energy consumption estimation are presented, since all models capture the traffic bits volume. Related to Cq and Cqe, both are easily extracted from the application description (e.g., extracted by straightforward simulation techniques), and both models present low computational complexity. The Cqe enables designers to capture, with more precision, the dynamic energy consumption by separating bits transition and bits volume. Related to Cqo results, it might also be extracted from the application description aggregating time information on each communication event, i.e., the model captures an appropriate application execution time for CPU-bounded application. Cqd and CqdPq models capture the communication dependence information that enables to prevent packet contention on the communication architecture. Nonetheless, Cqd is a pessimistic model being adequate to model I/O-bounded applications.

A model that tends to generate better mapping results may be the one that includes the bit transitions information on CqdPq, which is a characteristic explored in Cqe. Nevertheless, CqdPq implies communication dependence information, which is hard to model by automatic tools and error prone task when performed manually. Finally, a worthwhile model that may be easily captured is the one that combines the timing knowledge of Cqo with the bit pattern knowledge of Cqe.

MoCs often exploit the axes of computing and communication only, minimizing the importance of storage, which is implicitly modeled within computing. However, the speed of computation and storage changes dramatically with the emerging technologies, and the mapping activity in NoCs needs new ways of modeling applications taking into account memory aspects, for example, temporal locality and predictability. In this context, as a future work, we plan to explore MoCs that consider the synergy of storage, communication, and computing.

Acknowledgement

This work is supported by FAPERGS under grant PqG 12/1777-4, and Grant Docfix SPI n.2843-25.51/12-3.

References

[1] A. Aguiar, S. Filho, F. Magalhães, F. Hessel, On the design space exploration through theu Hellfire framework, J. Syst. Archit. 60 (1) (2014) 94–107. http://dx.doi.org/10.1016/j.sysarc.2013.10.011.

[2] M. Arjomand, A. Boroumand, H. Sarbazi-Azad, A generic FPGA prototype for on-chip systems with network-on-chip communication infrastructure, Comput. Electr. Eng. 40 (1) (2014) 158–167. http://dx.doi.org/10.1016/j.compeleceng.2013.11.031.

[3] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, G. De Micheli, NoC synthesis flow for customized domain specific multiprocessor systems-on-chip, IEEE Trans. Parallel Distrib. Syst. 16 (2) (2005) 113–129. http://dx.doi.org/10.1109/TPDS.2005.22.

[4] T. Bjerregaard, M. Stensgaard, and J. Sparsø, A scalable, timing-safe, network-on-chip architecture with an integrated clock distribution method, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'07), 2007, pp. 48–653. ⟨http://dx.doi.org/10.1109/DATE.2007.364667⟩.

[5] P. Bogdan, R. Marculescu, Non-stationary traffic analysis and its implications on multicore platform design, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (4) (2011) 508–519. http://dx.doi.org/10.1109/TCAD.2011.2111270.

[6] E. Carvalho, C. Marcon, N. Calazans, and F. Moraes, Evaluation of static and dynamic task mapping algorithms in NoC-based MPSoCs, in: Proceedings of the International Symposium on System-on-Chip (SOC), 2010, pp. 87–90. ⟨http://dx.doi.org/10.1109/SOCC.2009.5335672⟩.

[7] J. Castrillon, R. Leupers, G. Ascheid, MAPS: mapping concurrent dataflow applications to heterogeneous MPSoCs, IEEE Trans. Ind. Inform. 9 (1) (2013) 527–545. http://dx.doi.org/10.1109/TII.2011.2173941.

[8] K. Cheshmi, S. Mohammad, D. Versick, D. Tavangarian, and J. Trajkovic, A clustered GALS NoC architecture with communication-aware mapping, in: Proceedings of the Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), 2015, pp. 425–429 ⟨http://dx.doi.org/10.1109/PDP.2015.113⟩.

[9] G. Chen, F. Li, and M. Kandemir, Compiler-directed application mapping for NoC based chip multiprocessors, in: Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), 42, 7, 2007, pp. 155–157. ⟨http://dx.doi.org/10.1145/1273444.1254796⟩.

[10] C.-L. Chou, U. Ogras, R. Marculescu, Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (10) (2008) 1866–1879. http://dx.doi.org/10.1109/TCAD.2008.2003301.

[11] Y. Cui, W. Zhang, and H. Yu, Decentralized agent based re-clustering for task mapping of tera-scale network-on-chip system, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2012, pp. 2437–2440. ⟨http://dx.doi.org/10.1109/ISCAS.2012.6271791⟩.

[12] W. Dally and B. Towles, Route packets, not wires: on-chip interconnection networks. in: Proceedings of the Design Automation Conference (DAC) 2001, pp. 684–689. ⟨http://dx.doi.org/10.1109/DAC.2001.156225⟩.

[13] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli, Design of embedded systems: formal models, validation and synthesis, in: Proceedings of the IEEE 85, 3 1997, pp. 336–390. ⟨http://dx.doi.org/10.1109/5.558710⟩.

[14] E. Fusella, and A. Cilardo, PhoNoCMap: an application mapping tool for photonic networks-on-chip, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, pp. 289–292.
[15] A. Habibi, M. Arjomand, and H. Sarbazi-Azad, Multicast-Aware Mapping Algorithm for On-chip Networks, in: Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2011. pp. 455–462. ⟨http://dx.doi.org/10.1109/PDP.2011.76⟩.
[16] A. Hansson, M. Coenen, and K. Goossens, Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip, in: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, (DATE '07), 2007, pp. 954–959. ⟨http://dx.doi.org/10.1109/DATE.2007.364416⟩.
[17] O. He, S.-Q. Dong, W. Jang, J. Bian, D.Z. Pan, UNISM: unified scheduling and mapping for general networks on chip, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20 (8) (2012) 1496–1509. http://dx.doi.org/10.1109/TVLSI.2011.2159280.
[18] J. Hu and R. Marculescu, Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints, in: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), 2004, pp. 234–239. ⟨http://dx.doi.org/10.1109/DATE.2004.1268854⟩.
[19] J. Hu, R. Marculescu, Energy- and performance-aware mapping for regular NoC architectures, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 24 (4) (2005) 551–562. http://dx.doi.org/10.1109/TCAD.2005.844106.
[20] J. Huang, C. Buckl, A. Raabe, and A. Knoll, Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems, in: Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2011, pp. 447–454. ⟨http://dx.doi.org/10.1109/PDP.2011.10⟩.
[21] M. Hosseinabady, J. Nunez-Yanez, Run-time stochastic task mapping on a large scale network-on-chip with dynamically reconfigurable tiles, IET Comput. Digit. Technol. 6 (1) (2012) 1–11. http://dx.doi.org/10.1049/iet-cdt.2010.0097.
[22] C.-T. Hwang, J.-H. Lee, Y.-C. Hsu, A formal approach to the scheduling problem in high level synthesis, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 10 (4) (1991) 464–475 [10.1109/43.75629].
[23] A. Iyer and D. Marculescu, Power and performance evaluation of globally asynchronous locally synchronous processors. in: Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA), 2002, pp. 158–168. ⟨http://dx.doi.org/10.1109/ISCA.2002.1003573⟩.
[24] N. Kapadia, S. Pasricha, A system-level cosynthesis framework for power delivery and on-chip data networks in application-specific 3-D ICs, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 24 (1) (2016) 3–16. http://dx.doi.org/10.1109/TVLSI.2015.2399279.
[25] S. Kaushik, A. Singh, and T. Srikanthan, Preprocessing-based run-time mapping of applications on NoC-based MPSoCs, in: Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2011. pp. 337–338. ⟨http://dx.doi.org/10.1109/ISVLSI.2011.43⟩.
[26] K. Keutzer, S. Malik, A. Newton, J. Rabaey, A. Sangiovanni-Vincentelli, System-level design: orthogonalization of concerns and platform-based design, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 19 (12) (2000) 1523–1543. http://dx.doi.org/10.1109/43.898830.
[27] M. Kreutz, C. Marcon, N. Calazans, L. Carro, and A. Susin, Energy and latency evaluation of NoC Topologies, in: Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), 2005, pp. 5866–5869. ⟨http://dx.doi.org/10.1109/ISCAS.2005.1465973⟩.
[28] S. Li, F. Jafari, A. Hemani, and S. Kumar, Layered spiral algorithm for memory-aware mapping and scheduling on network-on-chip, in: Proceedings of the NORCHIP, 2010. pp. 1–6 ⟨http://dx.doi.org/10.1109/NORCHIP.2010.5669442⟩.
[29] S.-S. Lu, C.-H., Lu, and P.-A. Hsiung, Congestion- and energy-aware run-time mapping for tile-based network-on-chip architecture. in: Proceedings of the IET International Conference on Frontier Computing. Theory, Technologies and Applications, 2010, pp. 300–305. ⟨http://dx.doi.org/10.1049/cp.2010.0578⟩.
[30] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner, Time and energy efficient mapping of embedded applications onto NoCs, in: Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC), 2005a, pp. 33–38. ⟨http://dx.doi.org/10.1109/ASPDAC.2005.1466125⟩.
[31] C. Marcon, N. Calazans, F. Moraes, F. Hessel, I. Reis, A. Susin, Exploring NoC mapping strategies: an energy and timing aware technique, Proc. Des. Autom. Test. Eur. (2005) 502–507. http://dx.doi.org/10.1109/DATE.2005.149.
[32] C. Marcon, J. Palma, N. Calazans, F. Moraes, A. Susin, R. Reis, Modeling the traffic effect for the application cores mapping problem onto NoCs, VLSI-SOC: From Systems To Silicon 240, Springer, 2005, pp. 179–194. http://dx.doi.org/10.1007/978-0-387-73661-7_12.
[33] C. Marcon, T. Webber, L. Poehls, and I. Pinotti. 2014. Pre-mapping algorithm for heterogeneous MPSoCs, in: Proceedings of the International Conference on VLSI Design, pp. 252–257. ⟨http://dx.doi.org/10.1109/VLSID.2014.50⟩.
[34] C. Marcon, N. Calazans, E. Moreno, F. Moraes, F. Hessel, A. Susin, CAFES: a framework for intrachip application modeling and communication architecture design, J. Parallel Distrib. Comput. 71 (5) (2011) 714–728. http://dx.doi.org/10.1016/j.jpdc.2010.10.002.
[35] F. Moraes, N. Calazans, A. Mello, L. Möller, L. Ost, HERMES: an infrastructure for low area overhead packet-switching networks on chip, Integr., VLSI J. 38 (1) (2004) 69–93. http://dx.doi.org/10.1016/j.vlsi.2004.03.003.
[36] S. Murali and G. De Micheli, Bandwidth-constrained mapping of cores onto NoC architectures, in: Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE) 2004, pp. 896–901. ⟨http://dx.doi.org/10.1109/DATE.2004.1269002⟩.
[37] S. Murali and G. De Micheli, SUNMAP: a tool for automatic topology selection and generation for NoCs, in: Proceedings of the Design Automation Conference (DAC), 2004, pp. 914–919. ⟨http://dx.doi.org/10.1145/996566.996809⟩.
[38] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, Mapping and configuration methods for multi-use-case networks on chips, in: Proceedings of the Conference on Asia and South Pacific Design Automation (ASP-DAC), 2006, pp. 146–151 ⟨http://dx.doi.org/10.1109/ASPDAC.2006.1594673⟩.
[39] S. Murali, M. Coenen, A. Radulescu, K. Goossens, G. De Micheli, A methodology for mapping multiple use-cases onto networks on chips, Proc. Des., Autom. Test. Eur. (DATE '06) (2006) 118–123. http://dx.doi.org/10.1109/DATE.2006.244007.
[40] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, Designing application-specific networks on chips with floorplan information, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '06), 2006, pp. 355–362. ⟨http://dx.doi.org/10.1109/ICCAD.2006.320058⟩.
[41] M. Sgroi, L. Lavagno, A. Sangiovanni-Vincentelli, Formal models for embedded system design, IEEE Des. Test. Comput. 17 (2) (2000) 14–27. http://dx.doi.org/10.1109/54.844330.
[42] P. Sahu, S. Chattopadhyay, A survey on application mapping strategies for Network-on-Chip design, J. Syst. Archit. 59 (1) (2013) 60–76. http://dx.doi.org/10.1016/j.sysarc.2012.10.004.
[43] M. Stefani, T. Webber, R. Fernandes, R. Cataldo, L. Poehls, and C. Marcon, Task partitioning optimization algorithm for energy saving and load balance on NoC-based MPSoCs. in: Proceedings of the Sixteenth International Symposium on Quality Electronic Design (ISQED) 2015, pp. 130–134. ⟨http://dx.doi.org/10.1109/ISQED.2015.7085412⟩.
[44] G. Sun, Y. Li, Y. Zhang, L. Su, D. Jin, and L. Zeng, Energy-aware run-time mapping for homogeneous NoC, in: Proceedings of the International Symposium on System on Chip (SoC), 2010, pp. 8–11. ⟨http://dx.doi.org/10.1109/ISSOC.2010.5625542⟩.
[45] S. Tosun, O. Ozturk, and M. Ozen, An ILP formulation for application mapping onto Network-on-Chips, in: Proceedings of the International Conference on Application of Information and Communication Technologies (AICT), 2009, pp. 1–5. ⟨http://dx.doi.org/10.1109/ICAICT.2009.5372524⟩.
[46] C. Wu, C. Deng, L. Liu, J. Han, J. Chen, S. Yin, S. Wei, An efficient application mapping approach for the co-optimization of reliability, energy, and performance in reconfigurable NoC architectures, IEEE Trans. Comput.-Aided Des. Integr. Circuits Systems 34 (8) (2015) 1264–1277. http://dx.doi.org/10.1109/TCAD.2015.2422843.
[47] B. Yanga, L. Guanga, T. Sänttia, J. Plosilaa, Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip, Microprocess. Microsyst. 37 (4) (2013) 460–471. http://dx.doi.org/10.1016/j.micpro.2012.08.005.
[48] T. Ye, L. Benini, and G. De Micheli, Analysis of power consumption on switch fabrics in network routers, in: Proceedings of the Design Automation Conference (DAC), 2002, pp. 524–529. ⟨http://dx.doi.org/10.1109/DAC.2002.1012681⟩.

Further reading

[49] C. Zimmer and F. Mueller, Low contention mapping of real-time tasks onto a TilePro 64 core processor, in: Proceedings of the IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), 2012, pp. 131–140. ⟨http://dx.doi.org/10.1109/RTAS.2012.36⟩.
[50] W. Zhou, Y. Zhang, and Z. Mao, Pareto based multi-objective mapping IP cores onto NoC architectures, in: Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), 2006, pp. 331–334. ⟨http://dx.doi.org/10.1109/APCCAS.2006.342418⟩.