# Models for Embedded Application Mapping onto NoCs: Timing Analysis

César Marcon, Márcio Kreutz, Altamiro Susin
GME – Instituto de Informática – UFRGS
Av. B. Gonçalves, 9500 - Porto Alegre, RS, Brazil
{marcon, kreutz, *susin*}@inf.ufrgs.br

Ney Calazans
PPGCC – Faculdade de Informática – PUCRS
Av. Ipiranga, 6681 - Porto Alegre, RS, Brazil
calazans@inf.pucrs.br

## Abstract

*Networks-on-chip (NoCs) are an emergent communication infrastructure, which can be designed to deal with growing system complexity and technology evolution. The efficient use of NoCs needs techniques for application cores mapping, allowing reducing the message latency and consequently the overall execution time. To obtain mappings that fulfill the requirements during high-level design, appropriate models for NoCs and application cores become mandatory. High abstraction levels modeling may lead to unreliable estimates. On the other hand, detailed models may imply complex algorithms and high computational effort, with unacceptable computation time to get satisfactory results. NoC modeling for latency estimation requires capturing some infrastructure characteristics like topology and routing policies. Application cores models have to capture the application behavior, in terms of computation and/or communication. For instance, communication weighted models (CWM) and communication dependence model (CDM) consider only application communication aspects. However, the communication dependence and computation model (CDCM) consider both aspects of an application. This work compares these three models, according to their algorithm complexity and accuracy to model the application performance. We show that depending on the application characteristics, one of the models can be more suitable than the others.*

## 1 Introduction

New technologies allow that many millions of transistors be integrated onto a single chip and thus enable the implementation of complex systems-on-chip (SoCs). These systems need special communication resources to handle very tight design requirements. Many designers propose to change from the mainstream synchronous design paradigm to a globally asynchronous and locally synchronous (GALS) design paradigm [1]. GALS design method divide the application into synchronous domains placed inside a limited region, which is usually called *tile*. An asynchronous communication resource provides the communication between these tiles. A network-on-chip (NoC) is an infrastructure essentially composed by a set of routers interconnected by point-to-point communication channels. NoCs are easily adapted to implement systems based on the GALS paradigm. NoC channels can be designed to provide an asynchronous communication protocol between synchronous domains. In addition, NoCs provide high scalability, reusability, reliability, and efficient energy consumption [2].

Consider a SoC implemented with GALS paradigm, composed by ***n*** cores and employing a NoC as communication infrastructure. The application-mapping problem consists in finding an association of each core to a tile (a *mapping*) such that some cost function is minimized. In general, this mapping problem allows ***n!*** possible solutions. Given future SoCs with hundreds of tiles [3], exhaustive search into all solutions' space will rapidly become unfeasible. Thus, the optimal implementation of such SoCs requires efficient mapping strategies and sound application models. Some mapping strategies have been proposed. For example, [4] and [5] propose two different strategies with the same application model. We call this model *communication weighted model* (CWM), since it takes into account the overall volume of communication between each pair of cores. [6] compares CWM with *communication dependence model* (CDM), since CDM considers also the communication timing. [7] compares CWM with *communication dependence and computation model* (CDCM), which considers the communication timing and also the computation quantity. CDM and CDCM can lead to a better mapping than the ones achieved with CWM. However, it still needs to be verified the exact strength and weaknesses of each model in capturing applications behaviors, as well as their algorithm complexity. To clarify these issues, this paper evaluates the modeling effect in the application-mapping task, aiming to reduce the overall execution time of an application running on a mesh NoC architecture.

The remaining of the paper is organized as follows. Section 2 discusses related work. Section 3 defines application models for the mapping problem. Section 4 describes and compares algorithms and models. Section 5 presents experimental results and Section 6 presents some conclusions.

## 2 Related Work

Hu and Marculescu [4] propose a model called *application characterization graph* (APCG), which is a way of capturing the communication weight of a given

application. Murali and De Micheli [5], propose a model similar to that in [4], which is represented by a structure called *core graph*. Both works, whose models are classified here as CWM, are used to achieve mappings that minimize some design constraints, like energy consumption and average communication delay.

In all approaches that use the CWM strategy, essential information regarding the instant of time that messages are exchanged is lost, compromising some mapping results.

The authors of [6] propose a model that captures the communication volume and dependence. This model allows describing applications more accurately than the previous model, generating mappings with less overall execution time. The same authors introduce in [7] a new model that captures not only the communication volume and dependence, but also the computation quantity. The joint effect of computation and communication leads to better mappings, because contentions can be better estimated and then avoided.

This work evaluates application models aiming to compare their associated complexities and algorithms.

## 3 Problem Formulation

The problem of mapping application cores onto NoCs is a complex one. The designer splits the application tasks into cores. Each core behavior can be modeled by its computation and communication characteristics. This Section gives a formulation of the mapping problem in terms of graph structures and proposed models.

### 3.1 Graph Definitions

**Definition 1**: A *communication weighted graph* (CWG) is a directed graph $<C, W>$. The set of vertices $C = \{c_1, c_2 \ldots, c_n\}$ represents the set of cores in one application. Let $w_{ab}$ be the number of bits of all packets sent from a core $c_a$ to a core $c_b$. Then, the set of edges $W$ is $\{(c_a, c_b) \mid c_a, c_b \in C \text{ and } w_{ab} \neq 0\}$, and each edge is labeled with the value $w_{ab}$.

$W$ represents all communications between application cores, and CWG informs the relative communication volume of the application. This is similar to the definitions of *APCG* [4] and *core graph* [5].

**Definition 2**: A *communication dependence graph* (CDG) is a directed graph $<V, D>$ [6]. Let $C$ be a set of cores of a given application and let $v_q = (c_a, c_b, w_{ab})$ be the *q-th* message sent from core $c_a$ to core $c_b$ with bit volume $w_{ab}$. $V = \{v_1, v_2, \ldots, v_k\}$ denotes the set of all messages between all cores and corresponds to the set of CDG vertices. There are also two special vertices named START and END. START does not depend on any vertex, and no vertex depends on END. $D = \{(v_i, v_j) \mid v_i, v_j \in V\}$ represents the set of message dependences, corresponding to the set edges.

CDG represents all core communication of a given application. Edges are non-valued, and the edge direction denotes that the message contained into the target vertex depends on the origin vertex. In other words, the target vertex is communication *dependent* on the origin vertex.

**Definition 3**: The *communication dependence and computation graph* (CDCG) is a directed graph $<P, D>$ [7]. CDCG has definition similar to CDG, the difference consisting on the CDCG vertices ($P$), which contain the computation quantity besides the messages exchanged between each pair of application cores. Elements of $P$ are 4-tuples $p_{abq} = (c_a, c_b, t_{aq}, w_{abq})$, where $c_a, c_b \in C$, and $p_{abq}$ is the q-th message sent from $c_a$ to $c_b$. Messages contain $w_{abq}$ bits that are transmitted after the computation time ($t_{aq}$) of the originating core ($c_a$) has elapsed.

The use of these models for solving the mapping problem is evaluated onto a NoC with mesh topology using wormhole, deterministic XY routing algorithm [8]. $\boldsymbol{n}$ tiles compose an instance of this NoC. Figure 1 (a) depicts the structure of this NoC, where each tile ($\tau$) contains a router ($r$) and a local core ($c$). Figure 1 (b) shows that each router connects up to 5 external I/O channels and each channel has buffered inputs and unbuffered outputs. The local channel is devoted to provide communication with the local core. The remaining channels provide inter-router communication.

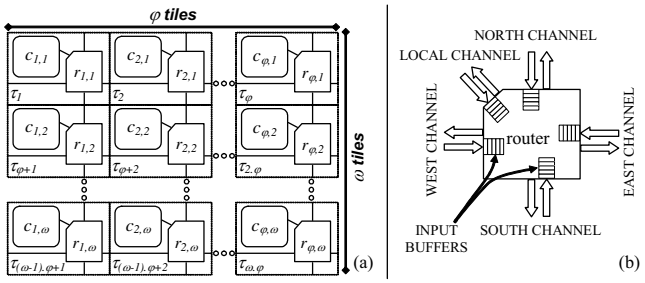


**Figure 1 – NoC mesh topology (a) and its router (b).**

**Definition 4**: A *communication resource graph* (CRG) is a directed graph $<\Gamma, L>$, where the vertex set is the set of tiles $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and the edge set $L = \{(\tau_i, \tau_j), \forall \tau_i, \tau_j \in \Gamma\}$ designates the set of routing paths from $\tau_i$ to $\tau_j$.

The value $\boldsymbol{n}$ is again the total number of tiles and is equal to the product of the two NoC dimensions, $\varphi$ and $\omega$. CRG edges and vertices represent physical links and tiles of the target architecture, respectively.

Figure 2 illustrates the above definitions using a hypothetical application with four IP cores, exchanging a total of six messages, in a 2×2 NoC.

Figure 2(a) depicts one possible CDCG where

$P$ = {$p_{EA1}$ = (E, A, 10, 20), $p_{EA2}$ = (E, A, 20, 15), $p_{AF1}$ = (A, F, 6, 15), …} and $D$ = {(START, $p_{EA1}$), ($p_{EA1}$, $p_{EA2}$), ($p_{AB1}$, $p_{AF1}$), …}. Figure 2(b) depicts the correspondent CDG for the chosen CDCG, where $P$ = {$p_{EA1}$ = (E, A, 20), $p_{EA2}$ = (E, A, 15), $p_{AF1}$ = (A, F, 15), …} and the same $D$ of CDCG. Figure 2(c) shows a CWG, where $C$ = {A, B, E, F}, the edge labels are $w_{AB}$ = 15, $w_{AF}$ = 15, $w_{BF}$ = 40, $w_{EA}$ = 35, $w_{FB}$ = 15 and the set $W$ can be extracted easily from the figure. Figure 2(d) depicts an arbitrary mapping of $C$ onto de NoC, corresponding to a CRG as follows: CRG = <{$\tau_1$, $\tau_2$, $\tau_3$, $\tau_4$}, {($\tau_1$, $\tau_2$), ($\tau_2$, $\tau_1$), ($\tau_1$, $\tau_3$), ($\tau_3$, $\tau_1$), ($\tau_2$, $\tau_4$), ($\tau_4$, $\tau_2$), ($\tau_3$, $\tau_4$), ($\tau_4$, $\tau_3$)}>.

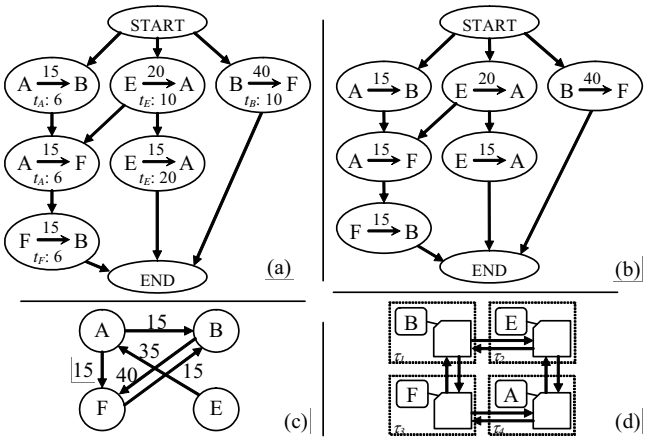


**Figure 2 – (a) CDCG, (b) CDG, (c) CWG, (d) CRG.**

### 3.2 Timing Models

To estimate the execution time of the application, this section presents a timing model for XY deterministic algorithm with wormhole routing. The *total message delay* is composed by the routing delay and by the payload delay. The *routing delay* is the time spent to create the communication path, which is determined during the traffic of the first flit. The *payload delay* depends only on the number of the remaining flits. Let $n_{ab}$ be the number of flits of a message sent from $c_a$ to $c_b$. Let $\lambda$ be the period of a clock cycle, and let $t_r$ be the number of cycles needed for routing decision. In addition, let $t_l$ be the number of cycles needed to transmit a flit through a link (between tiles or between core and router). The routing delay ($d_{Rij}$) and the payload delay ($d_{Pij}$) from $\tau_i$ to $\tau_j$, are given by equations (1) and (2), respectively, considering that a packet goes through $\eta$ routers without contention. Contentions are not expressed here, since they can only be determined by the knowledge of the overall messages timing.

$$(1) \quad d_{Rij} = (\eta \times (t_r + t_l) + t_l) \times \lambda$$

$$(2) \quad d_{Pij} = (t_l \times (n_{ab} - 1)) \times \lambda$$

The total message delay ($d_{ij}$), obtained from the sum of ($d_{Rij}$) and ($d_{Pij}$), is expressed by Equation (3). All algorithms use $d_{ij}$ to compute the latency of a message.

$$(3) \quad d_{ij} = (\eta \times (t_r + t_l) + t_l \times n_{ab}) \times \lambda$$

## 4 Mapping Algorithms

We implemented an algorithm that mix simulated annealing [9] and simulated evolution [10]. This algorithm is called *external algorithm* since it works as a shell that calls three mapping cost algorithms (ModelMapping function), one for each model. Figure 3 depicts the external algorithm.

```
1.  map ← InitialRandomMapping
2.  cost ← ModelMapping
3.  if cost < savedCost
        savedMap ← map
        savedCost ← cost
    elsif cost < savedCost + temperature
        SaveMapping (savedMap, savedCost)
        savedMap ← map
        savedCost ← cost
4.  map ← RandomMapping(temperature)
5.  temperature ← temperature – 1
6.  if temperature > 0
        go to step 2
7.  savedMap, savedCost ← BestMapping
8.  map ← RandomMapping(1)
9.  cost ← ModelMapping
10. if cost < savedCost
        savedMap ← map
        savedCost ← cost
11. iteration ← iteration – 1
12. if iteration > 0
        go to step 8
13. end
```

**Figure 3 – External mapping algorithm.**

The algorithm starts with an initial random mapping associating cores to tiles; it then evaluates the mapping cost, executing the specific algorithm for each model; and searches for a new mapping that reduces the computed cost, until reaching a stop condition. For all algorithms, the mapping cost is computed by the ModelMapping function, which depends on the model and the infrastructure. The external algorithm is divided in two sets of steps: (*i*) from step 1 to 6 - the algorithm accepts mappings worse than previous ones, but always, saving the best mappings in SaveMappings list. The acceptance level and the degree of variations between mappings are reduced with a temperature parameter. This procedure searches for the best mappings trying to avoid local minima. (*ii*) from step 7 to 13 - the second part of the algorithm starts searching for the best mapping of the saved mappings list and tries to find a better one by a smaller grain search method, which produces mappings that change only one association between tiles and cores in each execution loop.

## 4.1 CWM Mapping Algorithm

The CWM algorithm is a ModelMapping function of the external algorithm. This algorithm starts selecting a source vertex. For each source vertex, all output edges are visited to find the target vertex and the number of bits transmitted. It proceeds until no unvisited vertices are left. When an edge is visited, the number of bits transmitted, and the source and target vertices are informed to the function that maps them onto the NoC graph (CRGMapping function). Figure 4 (a) depicts the ModelMapping function of CWM.

```
1. sourceVertex ← SearchSource
2. if sourceVertex = NULL
       go to step 7
3. targetVertex, bits ← SearchTarget
4. if targetVertex = NULL
       go to step 1
5. time ← CRGMapping(sourceVertex,
                     targetVertex, bits)
6. go to step 3
7. return time
```

(a) ModelMapping of CWM

```
for x of source to x of target
    for y of source to y of target
        time ←TimeOfResource
return time
```

(b) CRGMapping

**Figure 4 – (a) ModelMapping function for CWM mapping, and (b) CRG mapping algorithm.**

The effective CWG mapping onto CRG is performed according to the XY routing algorithm, and is called CRGMapping. It receives source and target vertices, since it allows associating the CWG vertices with the XY physical places, allowing computing the communication path. Figure 4 (b) describes this algorithm. TimeOfResource is the delay of the message according to the timing model.

CWM model enables to compute only the time spent in the communication infrastructure. TimeOfResource cannot compute contentions, since CWM model does not model any kind of timing information.

## 4.2 CDM Mapping Algorithm

CDM improves CWM, as it considers message dependence. Dependent messages cannot be concurrent. On the other hand, independent messages can be sent at the same time, and consequently may lead to contentions, if they share a given communication resource. Message contention implies larger execution time. The goal of CDM algorithms is to search for mappings that minimize the sharing of communication resources for concurrent messages.

The ModelMapping function of CDM is implemented by firstly setting dependences in all vertices, i.e. all dependence paths are searched, and for all vertices found in the paths there is a *dependence list* to annotate the predecessor vertices. Each vertex has also an integer to store its dependence level. CRG can execute vertices with the same dependence level at the same time, because there is no dependence relation between them. Figure 5 (a) shows these steps, which are performed only once and are used during all mappings.

```
from Start to End Vertex
    search all paths
        SetDependenceLevel
        NoteDependeces
```

(a) Dependence settings

```
1. dependence ← 0
2. if dependence = DependenceLevel(End)
       return time
3. vertex ← SearchVertices(dependence)
4. if vertex = NULL
       go to step 6
5. time ← CRGMapping(vertex)
6. go to step 3
7. dependence ← dependence + 1
8. go to step 2
```

(b) ModelMapping of CDM

**Figure 5 – ModelMapping function for CDM mapping.**

The ModelMapping function uses the dependence level to execute CDG over CRG. The algorithm stops when it reaches the dependence level of END vertex, returning the computed execution time. For each dependence level, all vertices are mapped onto CRG, i.e. the message contained into the vertex is computed into CRG according to the CRGMapping described in Figure 4 (b). However, the CRGMapping only receives a vertex as parameter, since the CDG vertex contains the source and target vertices, and the number of bits transmitted. The TimeOfResource function computes all independent messages as concurrent. Thus, the algorithm considers that these messages will be contained in the routing buffers, increasing the execution time. Because of that, mappings with many contentions can be avoided. Figure 5 (b) depicts the CDM ModelMapping function. When all vertices at the same dependence level are executed, a dependence control variable is incremented and the process is repeated.

## 4.3 CDCM Mapping Algorithm

CDCM improves CDM by adding the tasks computation time. It changes the algorithm focus. While CDM tries to avoid contentions by a pessimistic approach - “if messages are not dependent, either the mapping avoids resource sharing or messages will cause contentions”, CDCM can more effectively compute the exact time slice of each message. Thus, even independent messages can concur for the same resource without causing contentions, because resources are shared at different moments of time.

The CDCM ModelMapping function is implemented by firstly setting the dependences level of each vertex, similarly to CDM. However, there is no list of dependences, since the dependence knowledge is only necessary to compute the dependence levels. Dependence levels with computation time define a total ordering of messages.

The CDCM ModelMapping function is similar to that of CDM, depicted in Figure 5 (b). The only difference is in the accounting of task computation time. The main difference

between CDCM and CDM algorithms relies on the TimeOfResource function. Because of the exact knowledge of message time slice, message contentions are only computed if the messages concur for a same resource at the same time. This approach reduces the number of "false" possible contentions, enabling the algorithm to explore mappings reducing contentions that really will occur.

### 4.4 Algorithms Comparison

The essential difference between the models is that CWM and CDM is not able to estimate application execution time. CWM cannot evaluate message contention either. The main advantages of CWM are (*i*) easy extraction of application core graph (CWG), since it can be done by simulation techniques; and (*ii*) low computational complexity. The automatic extraction of CDM and CDCM core graphs are harder tasks, since simulations only allow extracting possible message orderings, but not the desired message dependence information. Therefore, CDGs and CDCGs are normally described by hand, increasing error susceptibility.

The main drawback of CDM and CDCM algorithms is that in real embedded applications, the number of messages exchanged between cores is much larger than the number of cores. Since vertices of CDG and CDCG represent message exchanges between cores and each vertex of CWG represents just a core, CDGs and CDCGs are naturally larger than CWGs.

CWM algorithms are less complex, because this only informs direction and quantity of bits transmitted. CDM and CDCM algorithms are more complex, since they have to deal with message dependences. However, the CDM algorithm is the most complex one, because its pessimistic approach needs to compute lists of dependences for all vertices, and these lists have to be recomputed each time a vertex is evaluated.

When a message has more bits than the lower level protocols allows, it is broke into several packets. Messages with more than one packet can be interleaved, in the NoC resources making difficult high-level contention estimation, even for models like CDCM. On the other hand, the pessimistic approach of CDM tends to avoid this kind of contentions, since the associated algorithm searches for mappings that avoid the concurrency of all independent messages, and dependent messages cannot have interleaved packets.

## 5 Experimental Results

In order to compare the models, hypothetical applications with special characteristics were used as benchmarks. The chosen application characteristics are computation versus communication and message dependence versus message concurrency. The applications comprise create two set of benchmarks whose purposes are to evaluate the models accuracy in finding mappings that reduce the overall execution time. One set compares computation versus communication and another one compares dependence versus concurrency. The benchmarks contain applications that vary all characteristics from 0% to 100%. Both benchmarks include 9 applications, with an average of 18 vertices, and were built over the CDCM model, since CDM and CWM can be automatically extracted from CDCM with an auxiliary tool. The hypothetical applications aim to evaluate a large range of characteristics combinations, which can be matched with real applications. An assumption is that if the characteristics of a real application matches the characteristics of a hypothetical one, the best model found during the mapping process also matches. Therefore, the designer has just to evaluate the characteristics of its application to choose the best model for its purpose.

### 5.1 Concurrence and Dependence Evaluation

It is not trivial to determine how much an application is message concurrent or message dependent. Since there are many variables involved to quantify these characteristics, this work proposes a heuristics that proportionally relate *graph dependence* (*gd*) with *graph concurrency* (*gc*).

Let $n$ be the number of vertices of an application graph, and let $vd(v)$ be the list of vertices that vertex $v$ depends on. Then, $gd$ is computed with by the sum of the $vd$ of all $n$ vertices as depicted by Equation (4).

$$(4) \quad gd = \sum_{i=1}^{n} vd(v_i)$$

Let $cc(v)$ be the set of all *concurrency combinations* of vertex $v$, i.e. all combinations of vertices that may execute concurrently with vertex $v$. Equation (5) depicts $CC$, which is the set union of the $ccs$ of all $n$ vertices. The value $gc$ is the cardinality of $CC$. We assume this because $CC$ represents all possible concurrence combinations. Equation (6) shows the computation of $gc$.

$$(5) \quad CC = cc(v_1) \cup cc(v_2) \cup \ldots \cup cc(v_n)$$

$$(6) \quad gc = / \text{ CC } /$$

The adopted heuristics estimates concurrency and dependency percentages through the application of Equation (7). Note that in (7) the increase of concurrency implies the decrease of dependence and vice-versa.

$$(7) \quad gd\_gc = \frac{gd}{gc + gd} * 100\%$$

Equations (4), (6) and (7) are exemplified in Figure 6

and **Table 1**, which present a simple example of hypothetical application graph. Figure 6 shows a hypothetical application composed by six vertices V = {$V_1$, $V_2$, $V_3$, $V_4$, $V_5$, $V_6$} representing messages, besides START and END special vertices. In the right side of each vertex, there is a list of dependences. For instance, vertex $V_5$ depends on vertices START, $V_1$ and $V_3$, and vertex END depends on the execution of all vertices described in the graph. When its dependences are solved the application execution is finished.

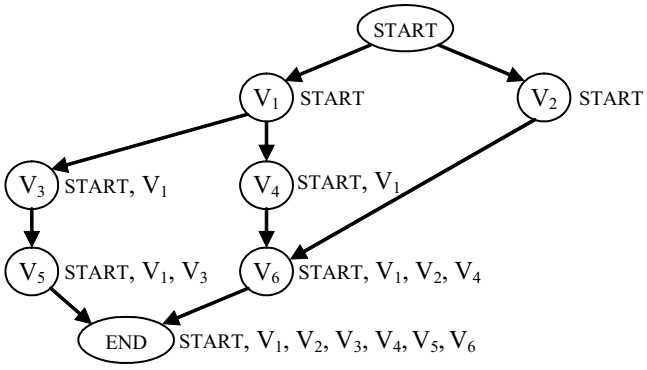


**Figure 6 – Comparison of concurrency versus dependence of a hypothetical application graph.**

**Table 1** illustrates the concurrency combinations of all vertices of Figure 6.

**Table 1 – Concurrence combinations of all vertex of the hypothetical application graph showed in Figure 6.**

| | |
|---|---|
| $cc(V_1)$ | { ($V_1$,$V_2$) } |
| $cc(V_2)$ | { ($V_2$,$V_3$,$V_4$), ($V_2$,$V_4$,$V_5$), ($V_1$,$V_2$), ($V_2$,$V_3$), ($V_2$,$V_4$), ($V_2$,$V_5$) } |
| $cc(V_3)$ | { ($V_2$,$V_3$), ($V_3$,$V_4$), ($V_3$,$V_6$), ($V_2$,$V_3$,$V_4$) } |
| $cc(V_4)$ | { ($V_2$, $V_4$), ($V_3$,$V_4$), ($V_4$,$V_5$), ($V_2$,$V_3$,$V_4$), ($V_2$,$V_4$,$V_5$), } |
| $cc(V_5)$ | { ($V_5$,$V_6$), ($V_4$,$V_5$), ($V_2$,$V_4$,$V_5$) } |
| $cc(V_6)$ | { ($V_3$,$V_6$), ($V_5$,$V_6$) } |

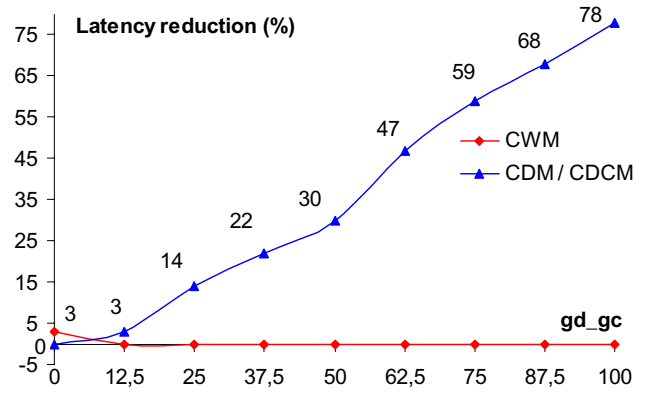


**Figure 7 – Average latency reduction achieved with mappings of CWM, CDM and CDCM models, when concurrency and dependence variations are concerned.**

Applying Equation (5) gives *CC* = {($V_1$,$V_2$), ($V_2$,$V_3$), ($V_2$,$V_4$), ($V_2$,$V_5$), ($V_3$,$V_4$), ($V_3$,$V_6$), ($V_4$,$V_5$), ($V_5$,$V_6$), ($V_2$,$V_3$,$V_4$), ($V_2$,$V_4$,$V_5$) }. Next, applying Equation (6) $gc = / CC / = 10$. In addition, applying Equation (4) gd = 20, is obtained. Finally, applying Equation (7) gives $gd\_gc = 66{,}67\%$. The interpretation is that this application is more dependent than concurrent. Using a benchmark with applications that do not consider computation and varies the dependence from 0% to 100%, and the opposite situation for concurrency, the results of Figure 7 are obtained.

Without considering computation, CDM and CDCM lead to identical results. However, it is possible to observe that CDM and CDCM, when compared to CWM, present a linear increase in latency reduction with the increase of dependence. All models have similar results only with applications with *gd_gc* lower than 12,5%.

## 5.2 Computation and Communication Evaluation

In order to comparatively evaluate computation and communication, all vertices of the hypothetical applications were assigned with an amount of message bits and an amount of computation time. We use a heuristic based on a Normal distribution, avoiding that the relation between these application characteristics be masked by discrepant values.

Let $s_i$ be the i-th sample, $\mu$ be the medium value of the samples computed by Equation (8), and $\sigma$ be the deviation assumed (we consider 30%), which is computed by Equation (9). Then, Equation (10) computes the average of the considered samples ($\omega$).

$$(8)\quad \mu = \frac{\sum_{i=1}^{n} s_i}{n}$$

$$(9)\quad \sigma_- = \mu * 0.7 \text{ and } \sigma_+ = \mu * 1.3$$

$$(10)\quad \omega = \frac{\sum_{i=1}^{n} \mu - \sigma_- < s_i < \mu - \sigma_+}{/\ \mu - \sigma_- < s_i < \mu - \sigma_+ \ / \ \forall\ 1 \le i \le n}$$

The *graph computation time* (*gct*) is obtained by applying Equation (10), considering that each sample is equal to the *computation time* of each vertex.

Since it is not possible to precisely estimate the time spent by the communication before mapping, we use a heuristics based on probabilistic traffic evaluation and NoC size. To define the probabilistic parameters, a practical observation was used. Quality mappings aimed at latency reduction obtain, in average, communication paths, which take less 20% of the average communication time (*mtc*). The *mtc* value is computed by the average size of paths and

the time spent in each path according to Section 3.2. The reduction to less than 20% happens because cores with high communication rate are placed in neighbor tiles.

Let *fs* be a function that calculates the number of flits according to the number of bits, and let $w_{ij}$ be the number of bits transmitted from core *i* to core *j*. Then, to estimate the time caused by communication between these cores ($t_{ij}$), we apply Equation (11).

$$t_{ij} = fs(w_{ij}) * mtc * 0.2 \quad (11)$$

The *time* spent by *graph communication* (*tgc*) is obtained by applying Equation (10), considering that each sample is equal to the time spent by the message communication, described in each vertex.

Similarly, to display dependence versus concurrency, we use Equation (12) to estimate the relative percentage between communication and computation.

$$tgc\_gct = \frac{tgc}{gct + tgc} * 100\% \quad (12)$$

In computation versus communication comparison, all hypothetical applications of the benchmark are adjusted to have around 50% dependency and 50% concurrency. Figure 8 shows the results for computation versus communication variations.

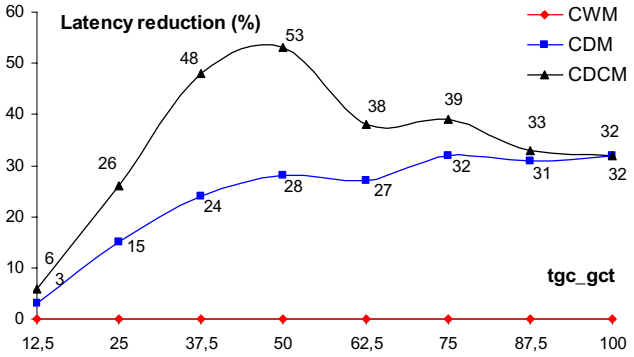


**Figure 8 – Average latency reduction achieved with mappings of CWM, CDM and CDCM models, when computation and communication are concerned.**

As it can be observed in Figure 8, the maximum latency reduction is achieved for balanced applications – 50% communication and 50% computation - using CDCM. This justifies the model efficiency to capture the expressiveness of computations in the application. However, when the application is dominated either by communication or by computation, the results for CDM and CDCM are similar. This happens because in both situations, communication volume drives the mapping algorithm, and both models capture the communication behavior with similar expressiveness.

# 6 Conclusions and Future Work

This paper addresses the problem of mapping applications onto NoCs, based on different models to express application behavior. Three models were compared – CWM, CDCM and CDM – each having different capacities to express the relationship among several applications characteristics, such as communication versus computation and dependence versus concurrence. A model to estimate the relative dominance of an application characteristic over another is proposed and justified.

A set of experiments were conducted, based on hypothetical benchmarks. The proposed relative dominance models was used to show how the merits of application models (CDM, CDCM, CWM) can be evaluated. The experiments exemplify the optimization of latency, but the proposed models can be extended to deal with other communication parameters.

As future improvements, the authors expect to evaluate typical embedded applications, such as multimedia. The authors believe that this has the potential to obtain insights on important characteristics of application behavior.

# References


[1] A. Iyer and D. Marculescu. *Power and performance evaluation of globally asynchronous locally synchronous processors*. **ISCA**, pp.158-168, May 2002.

[2] W. Dally and B. Towles. *Route packets, not wires: on-chip interconnection networks*. **DAC**, pp.684-689, June 2001.

[3] S. Kumar et al. *A network on chip architecture and design methodology*. **ISVLSI**, pp.105-112, April 2002.

[4] J. Hu and R. Marculescu. *Energy-aware mapping for tile-based NoC architectures under performance constraints*. **ASP-DAC**, pp.233-239, January 2003.

[5] S. Murali and G. De Micheli. *Bandwidth-constrained mapping of cores onto NoC architectures*. **DATE**, pp.896-901, February 2004.

[6] C. Marcon et al. *Time and Energy Efficient Mapping of Embedded Applications onto NoCs*. **ASP-DAC**, January 2005.

[7] C. Marcon et al. *Exploring NoC Mapping Strategies: An Energy and Timing Aware Technique*. **DATE**, pp. 502-507, March 2005.

[8] F. Moraes et al. *HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip*. Integration, the VLSI Journal, v. 38, n. 1, p. 69-93, October 2004.

[9] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, *Science* **220** pp.671-680, 1983

[10] L. Fogel. *Intelligence through Simulated Evolution: Forty Years of Evolutionary Programming*, Wiley-Interscience, July 1999.