High level analysis of multiprocessor system on chip

MAALEJ Issam
LESTER
CNRS FRE 2734, UBS
Rue st maude
56100, Lorient, France
Maalej@iuplo.univ-ubs.fr

GOGNIAT Guy
LESTER
CNRS FRE 2734, UBS
Rue st maude
56100, Lorient, France
guy.gogniat@univ.ubs.fr

ABID Mohamed
GMS,ENIS
B.P : W3038
SFAX,TUNISIA
mohamed.abid@enis.rnu.tn

PHILIPPE Jean Luc
LESTER
CNRS FRE 2734, UBS
Rue st maude
56100, Lorient, France
Jean-Luc.Philippe@univ.ubs.fr

ABSTRACT

Traditionally codesign flow aims to help the designers to take decisions about the system architecture, however due to the complexity of both system architectures and applications it is now necessary to perform before any codesign flow a first application analysis step in order to highlight the application's specificity. An efficient way to analyze the application is to define accurate metrics that exhibit the application's specificity impacting the system performance. In this paper, we propose several metrics that allow an efficient analysis of a system right after the specification of the application (tasks graph) before any design flow step. These metrics enable to measure some characteristics of the application as communication data size and data sharing between tasks... As they are defined before any design flow step they do not measure directly the system performance or the costs but they provide information that leads to optimize these costs. Several experimental results performed on an UMTS application demonstrate the efficiency of these metrics to exhibit the main characteristics of an application. The results also show the ability of the proposed metrics to correctly measure the applications specificity. Finally, the results show the interest of the metrics to be used in a clustering approach or other codesign flow.

Keywords

System on Chip (SoC), Multiprocessor, Metrics, Communication, System architecture.

1. INTRODUCTION

A thorough analysis of the application is necessary to define efficient and flexible multiprocessor system on chip (SoC). During the design steps of such systems, designers have to make important decisions as system architecture definition, communication architecture selection, resources allocation, scheduling... Of course, these decisions depend on the application and have a very important impact on the final system performance and costs (time, area, power...). Unfortunately the complexity of the systems leads to make these decisions more and more difficult. To be more efficient these decisions must take into account some application's specificity. For example, emphasizing some intensive communicating tasks will help the designers to optimize their system costs by allocating both tasks within the same processing resource. Traditionally codesign flow aims to help the designers to take these decisions, however due to the complexity of both system architectures and applications it is now necessary to perform before any codesign flow a first application analysis in order to highlight the application's specificity. In this paper as will be shown in the following we propose some metrics to realize this first analysis step.

To justify our proposition the complexity of both system architectures and applications is presented in the following. Generally, during the design flow, designers use their own experience to analyze the systems and perform manually some design steps. For example, communication architectures are generally designed based only on the experience of the designers. Thus sometimes it may lead to either an over-designed system or a failure to meet the requirements because of the complexity of the systems. This point is due to several reasons: On one hand the complexity of the applications is raising continuously (systems are composed of an increasing number of tasks) and at the same time, time to market constraint is more and more stringent. On the other hand, systems usually include many hardware and software components, numerous communication resources and a large number of memories. The complexity of both system architectures and applications (a large number of communicating tasks) leads to consider the communications as one of the more critical issue in the design flow. In fact, during the last years, numerous works have focused on the development of new communication resources such as for example communication supports (buses [1,2,3], micro-networks [4], network on chip [5]) and protocols in order to cope with the communication bottleneck.

Hence, to improve system design it is necessary to propose new approaches that first perform an analysis step. An efficient way to analyze the application is to define accurate metrics that exhibit the application's specificity impacting the system performance. These metrics enable to measure some characteristics of the application as communication data size and data sharing between tasks... These metrics do not correspond to system performance or costs estimates, but measure application's specificity which can guide the designers to optimize system performance and costs. For example, links between tasks have an important impact on the final system performance (this impact is more detailed later in this paper).

In this paper, we propose several metrics that allow an efficient analysis of a system right after the specification of the application (tasks graph) before any design flow step. These metrics have to highlight 1) the characteristics of the tasks and 2) the links between the tasks as will be shown in the following.

The paper is organized as follows. Section 2 presents an overview of related work dealing with metrics before describing our contribution. The specification graph that describes the system is presented in section 3. Section 4 details our metrics and a discussion about their relevance is proposed. The metrics are then validated in section 5 by experimental results related to an UMTS application. Section 6 draws some conclusions and discusses future works.

2. REVIEW OF RELATED WORK AND CONTRIBUTION

In this section, we review related previous work and describe our contribution.

2.1. Related work

A thorough analysis of the application is necessary for multiprocessor system on chip design. In SoC design flow, estimates of system costs as power, time and area are computed to build the system design. There are many codesign methodologies to design such system. Always, these methods try to take into account all task interactions as dependencies, communication and scheduling. But, due to the complexity of systems, it is very difficult to take into account all these parameters at the same time. Then, always, some assumptions and/or modifications on the application are made before beginning the codesign flow. For example in [6], Knudsen and Madsen manually define the communication architecture and resources of the system. Then, they propose several models to estimate the communication cost to design the hardware/software architecture. The main drawback of these approaches is due to the fact that an intuitive definition of the communication architecture may lead to a final architecture that doesn't meet requirements. Application analysis is done manually and the designers define the communication architecture and resources based on their own experience to make suitable the codesign flow (figure.1).

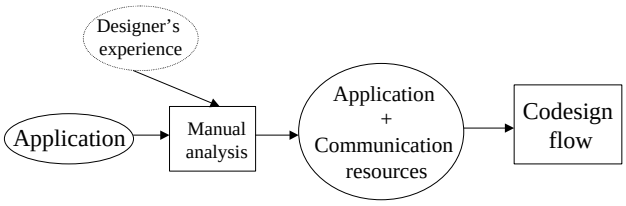

Figure 1. Earlier manual analysis

But due to the increasing complexity of applications and systems, we claim that a manual analysis, such the one proposed by Knudsen and Madsen [6], isn't anymore enough efficient to prepare the application to the design flow.

Hence, several researchers have proposed the definition of an earlier analysis step (automatic and fast) based on metrics computation. Some metrics are proposed before the codesign flow [7,8,9]. In [7], Le Moullec et al. present a tool to compute some metrics for data and control dependency analysis of a task. This tool is named design trotter. They also presume that it is possible to extend existing metrics to estimate the local memory resources size necessary for one task. The values proposed by [7] are based on a graph named HCDFG (Hierarchical Control Data Flow Graph) describing one task. However an application is composed of several tasks, a limitation of their approach is due to the fact that they only consider a single task at a time and they do not take into account the interactions between the tasks. For example, the local memory resources size value does not take into account the memory size shared between the tasks of the system. In [8], Carro et al. present three interesting metrics representing the characteristic of a task in terms of control, data transformation and data accesses. However, these metrics do not give information about the exclusion or the preference of the execution of a task with existing processing element as digital signal processor (DSP), video signal processor (VSP), general purpose processor or others. In [9], Scuito et al. define more specific metrics where exclusion and preference are not binary. In their case, these metrics correspond to a real value between 0 and 1 called Affinity. They have defined affinity to GPP, DSP and ASIC. This notion of affinity is more significant than totally exclusion or preference since it can represent more accurately the matching between a task and a processing element. Hence, in their approach they use the affinity metrics in order to propose a first allocation of processing element to each task of the application as shown in figure 2. Note that the application is analyzed before any codesign flow steps. Once the analysis step ends the application (noted application FCF (for codesign flow)) is ready for the codesign flow.

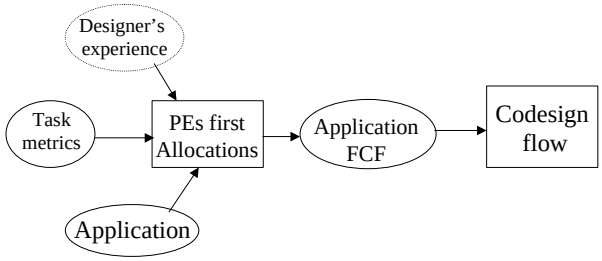

Figure 2. analysis based on tasks

Of course, task's characteristic has an important impact on the system performances, but considering only this feature is not always enough to take a decision. Even though a task is more suitable to a processing element (in term of affinity) it can have a large cost on power and area when implemented on it. If the task has a large amount of data exchange with another task (even if this task has an important affinity to another processing element) it may be more interesting to gather them in the same processing element to limit the communication cost and complexity. Hence, the affinity metrics are still limited since the analysis does not take into account the interaction between tasks that compose the system, and in some cases this can be a strong limitation.

2.2. Contribution

In order to consider the interactions between the tasks a first analysis step must be done to exhibit these interactions. The aim is to define different clusters of tasks that take into account the interactions between the tasks, the application is thus transformed

in order to reduce the complexity of the design flow and to define more rapidly an efficient and flexible system architecture.

Hence, in this paper, we propose several metrics able to make an efficient analysis of the systems at a tasks graph level. These metrics belong to the methodology described in figure 3.

In this methodology, clustering is used to define the application FCF. Clustering is defined as grouping tasks into groups to be later allocated onto processing elements. Dave and al. in [10] and [11] have demonstrated the interest of using clustering. Their clustering method is based in four values. Preference and exclusion vector indicates the most suitable mapping or the exclusivity between the tasks and all the processing elements of the library. These vectors are defined from the experience of the designer. The value of a preference or exclusion can be 0 or 1. However, in some cases due to interaction constraints a task can be more efficiently executed on one processing element even if its attribute is preferential or exclusion. The proposed tool by Dave et al. can not detect these cases. Execution time vector indicates the worst-case execution time of a task on all the processing elements of the library. Memory vector indicates the different types of storage required for a task. The two last values require a tool to be computed (it can be done by emulation, by simulation or preferentially by estimation) since as known the number of processing elements in the library increases continuously.

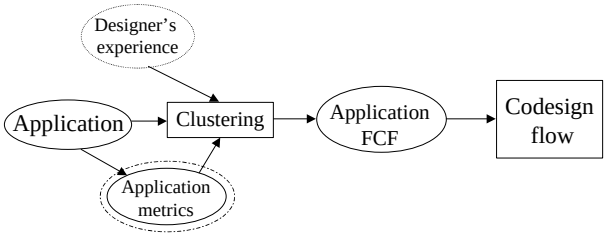

Figure 3. Our approach used for the earlier analysis

In our approach, the clustering is based on the application analysis using several metrics that are computed from the application's specificity and tasks characteristics. Compare to previous work we present metrics which are able to make an analysis (takes into account task interaction) of the systems from a tasks graph. These metrics are computed to evaluate the impact of the tasks and application's specificity onto the performance and the costs of the system. They do not estimate the system costs as power, time or area. But, they measure the impact of application proprieties as data transfer and data sharing onto the system costs. They also measure the impact of the task proprieties as task affinity and the interaction between the tasks as tasks cluster affinity onto the system costs. Thanks to such a thorough analysis of the application and tasks characteristics, the designers can perform an optimization of the application to improve and reduce the complexity of a SoC codesign flow.

In [12], Vahid and Gajski present similar metrics to analyze the application at a high description level. Their metrics are used to analyze the interaction between behaviors within tasks. Their analysis is used to change behaviors from a task to another in order to make more suitable the application for the codesign flow. This methodology is similar to our approach, but it can't be used in our context. First, transfer behaviors from a task to another leads to create for each system some new tasks. This point is not compatible with the notion of reuse, which consists on reusing the same task description for all systems. Also, interactions between behaviors within tasks are different from interactions between tasks within groups. So, the metrics used on the first case can't be used in the second.

In this paper we focus on the description of the application at a tasks graph level and on the computation of metrics able to analyze efficiently the application. The clustering step is not described in this paper, but we have implemented a genetic algorithm to perform this step. Thus, in this work we define the metrics to analyze the impact of grouping tasks onto the system performance. We propose communication metric, memory metric, throughput constraint metric, channel metric and affinity metric. Experimental results show the efficiency of our approach to evaluate the impact of each metric onto the performance.

3. SPECIFICATION GRAPH

To apply these metrics we first define a tasks graph, which describes the application at the tasks level. The graph describes the characteristics of each task of the application and the interaction between them like dependencies or data transfers.

The graph consists of nodes and edges. A node represents a task and an edge corresponds to a data dependency. The node is weighted with different values, which characterize the task. Each node is characterized by the following attributes:

- Throughput constraint: Usually, application has some computing constraints especially input and output constraints. In video computing, for example, system has to send 25 frames (images) per second. This throughput constraint characterizes the output tasks of the system. **Tc** names throughput constraint, it is an integer value and its definition is in bit per second.
- Affinity vector: It is a vector, which gathers the affinity values of a task **ti** to Processing Elements (PEs) possibilities. Affinity **Affinity (ti) (PEj)** of the task **ti** to a PE **PEj** is a real value between 0 and 1. If the value is 0, the task can not be executed on **PEj**. If the value is equal to 1, the task is fully suitable to be executed on **PEj**. Even if the value is 1 for a particular PE, it does not imply that the value is equal to 0 for another PE. These values can be determined based on prior experience of the designer or based on some work as proposed by Sciuto et al. [13]. If there is no preference or exclusion for any PEs, the affinity is considered to be the same for all PEs and is equal to 1/(number of PE possibilities).
- Local memory size: Memory resources used in a system can be the shared memory used by the components, or resources used locally by the tasks as registers. Local memory size determines the size of the memory resources required by the task. This value is evaluated using DesignTrotter [11]. It is named **LM_size.**
- Execution frequency: This value indicates the number of the execution of the task during one execution of the system. It is named **Exec_freq**.

The edge represents the interaction between tasks as dependency and data transfer. It is labeled with information about data size transfer and the type of transfer. Each data transferred from a task to another may be direct or memorized. If the **Memorization** value is equal to 1, the transfer needs memorization else the

transfer is direct. The **Word size** is a value representing the granularity of the data transfer. The **Word number** is a value representing the number of words transferred from a task to another. For example, the transfer of a vector of 12 integers, corresponds to **Word number** = 12 and **Word size** = 32 bits (size of integer). Finally, E**xecutions occurrence** is a value representing the number of transfers for a single execution of the producer task.

4. METRICS

In this section, we define several metrics, which allows analyzing the system characteristics at the tasks level. These metrics evaluate the impact of communication, memory, throughput and affinity on the final system performances. We have defined five metrics that are communication metric, memory metric, throughput constraint metric, channel metric and affinity metric. Metrics are represented by values between 0 and 1. The value is "1" when considering this metric will lead to an efficient optimization of the application, and "0" when no optimization is obvious considering this metric.

4.1. Communication metrics

These metrics evaluate the degree of optimization of communications and their impact on the system performances. Due to the complexity of systems, several communication resources need to be used. Hence, the tasks graph needs to be divided into clusters where each one shares a communication resource. To optimize the influence of the communication in such architecture, we define two communication metrics: communication inside the cluster, and communication between clusters (connections and data size) as shown in figure 4.

4.1.1. Equilibrate the exchanged data sizes between the different clusters

We make the assumption that *"Small exchanged data sizes lead to better system cost in time, area, power and design time"*. In fact, when the number of data to be transferred is important, the communication resources that must be used are costly to respect constraints and to perform data transfers at the required rate. In consequence, there is more area, power consumption and complexity of connections. Furthermore to cope with this large amount of exchanged data, various data transfer protocols must be used as burst, split… So, extra area is needed for the implementation of these protocols. Finally, to manage important data transfer load, multi-masters protocols are used. It leads that in addition to extra area used to implement these protocols, delay due to the master changes and access permissions are added in the system cost. Moreover, more exchanged data quantity is important more transfer synchronization is costly in time, memory resources and design time for the complex interface module.

Hence, to optimize the communication cost, exchanged data quantity must be the lowest in each cluster. Thus, it must be distributed equally within the different clusters.
We define **Equilibrate Data exchange** metric **E_De**, which enables to inform about the balance between the exchanged data quantity in each cluster. This metric is defined as follow:

`E_De = Min(Com_Degree(p))/Max(Com_Degree(p))`

Where `p` is a cluster of the architecture and:

`ComDegree(p)=Bit_exchange(p)/Bit_exchange(GS)`

$Bit\_exchange(p)=\sum_{(ti,tj)\in(p)} Task\_bits\_exchanges(ti,tj)$

Where `GS` is the Global System and:

`Task_bits_exchanges(ti,tj)=word_size×word_number×execution_occurrence×exec_freq`

Where `ti` is the task number `i`.

In other words, we first calculate the size of data exchanged in a cluster *p* **Bit_exchange(p)** (data represented by circles in fig.4). This data size corresponds to the sum of the size of all data exchanged by the tasks in the cluster. Data transfer is calculated based on the values associated to the edges. **Bit_exchange(p)** corresponds to the data transferred by the edge multiplied by the execution frequency of the task that produces this edge. Second, we compare the data quantity in each cluster by dividing the smallest quantity by the greatest one. If data quantity is balanced, then the minimum is equal to the max, so E_De = 1.

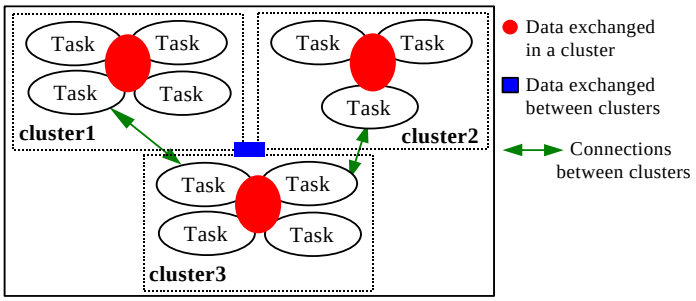

Figure 4. Target clustering to optimize communications

4.1.2. Minimize data transfers between clusters

We make the assumption that "*Transfers between clusters have a great cost in time, power and area"*. In fact, communications between two tasks in different clusters use communication resources within each cluster and the communication supports between the two clusters, or a dedicated communication resource (not shared) between the two tasks. In the first case, communications between tasks, which are in the same cluster, and communication between two tasks in different clusters are blocked until the current communication ends to release the communication resources. In the second case, there is simply an additional cost in design time, power and area due to the dedicated link.

To minimize the communication cost, clustering have to minimize data transfers between clusters. To evaluate this point, we propose two metrics. The first calculates the degree of exchanged data quantity between clusters. This metric is named **Data Exchange Inter Cluster** metric **DEIC**. The second calculates the degree of connections between clusters. It's named **Connection Inter Cluster** metric **CIC**.

DEIC = 1-(Bit_exchange_IC/Bit_exchange(GS))

$Bit\_exchange\_IC=\sum_{(ti,tj)\in(p)}[Task\_bit\_exchange(ti,tj);\ if\ cluster(ti)\bullet cluster(tj)]$

Where `Cluster(ti)` corresponds to the cluster whose the task `ti` belongs to.

This metric corresponds to the percentage of data, which are exchanged between clusters (Bit_exchange_IC) in relation to the total number of data in the system. If all data is exchanged between cluster, the clustering is very poor, and the metric is 0.

CIC = Connection_IC/Connection_GS

Where Connection_IC and Connection_GS are respectively the number of connections between clusters and within the global system.

4.2. Memory metric

As known, larger memory size implies slower access time and higher power consumption. Furthermore new technology as reconfigurable architecture supports spatial distribution of the memory resources (registers, memory bank…). So it's more interesting to distribute uniformly the memory used by the system to get the less possible cost in time, area and power consumption.

We propose a memory metric **Mem**, which informs about the better distribution of memory when it is equal to 1. This metric evaluates the memory resources size needed within a cluster: shared and local resources. For the local resources, the local memory size **LM_size** is computed using DesingTrotter [11]. Shared memory resources is determined with the graph.

Mem = Min(Mem_Degree(p))/Max(Mem_Degree(p))

Where `MemDegree(p)` corresponds to the percentage of memory used by the cluster `p` and:

$MemDegree(p)=Mem\_resources(p)/Mem\_resources(GS)$

Where `Mem_resources(p)` corresponds to the memory resources required by the cluster `p` and:

$Mem\_resources(p)=LM\_size(p)+shared\_mem(p)$

$LM\_size(p)=\sum_{ti\in(p)}[LM\_size(ti)];\ if\ ti\in p$

$shared\_mem(p)=\sum_{(ti,tj)\in(p)}edge.Memorization(ti,tj)\times Task\_bits\_exchanges(ti,tj)$

This metric evaluates from the graph the percentage of memory used by each cluster and verifies the balance of memory distribution (the value is defined between 0 and 1). This value is 1 when the memory is the same in each cluster.

4.3. Affinity metric

When a task has the highest affinity to a PE, its performance is optimal when implemented on this PE. The optimal clustering in each case, is the one that leads to build clusters of tasks having the higher affinity to the same PE.

AFF = min(met_aff(p))

Where

$Met\_aff(p)=Ag(MaxA(p))(p)/Amax(MaxA(p),nbt(p))$

Where `Ag(PEj)(p)` corresponds to the affinity of the cluster `p` to `PEj`, and:

$Ag(PEj)(p)=\sum_{ti\in(p)}A(ti)\ (PEj)\ if\ ti\in p$

Where `A(ti) (PEj)` corresponds to the affinity of the task `ti` to `PEj`, this is the value defined in the specification graph. `MaxA(p)` is the PE who has the maximal affinity to the cluster `p`.

$MaxA(p) = PEj,\ if\ Ag(PEj)(p) \geq Ag(PEk)(p)\ \forall\ k$

Where `Nbt(p)` is equal to the number of tasks in the cluster `p`. `Amax(PEj,n)` is equal to the sum of the `Nbt(p)` higher affinity to `PEj` of task in GS.

In other word, we calculate the affinity of the cluster, which corresponds to the sum of the tasks affinity within the cluster. We compare the maximal affinity value to a PE of the cluster with the best affinity value that can have this PE in the whole application.

4.4. Throughput constraint metric

As shown on the specification graph, some tasks have a throughput constraint. This throughput consists in the minimum throughput that communication resources must have to support efficiently the communication. This throughput must be imposed to all tasks that share the same communication resources. So, if another task in the same cluster has a different (higher or lower) throughput constraint, the communication resources have at least the higher throughput of the two tasks. As consequence, the communication resources have to be over-designed and cost more in time, power (frequency) and complexity. That is why the best clustering has to distribute efficiently the throughput constraints. We define the throughput constraint metric **Tc** as follows:

Tc = min (throughput_degree (p))

Where $Throughput\_degree(p)=Diff\_throughput(p)/Diff\_throughput(GS)$

Where `Diff_throughput(p)` corresponds to the difference between the higher and the lower throughput of tasks in the cluster `p`.

This metric calculates the degree between the different throughput constraints in one cluster, which is computed by the difference between the max and the min throughput in the cluster Diff_throughput(p). To evaluate the efficiency of the distribution of the throughput constraints we compare it with the difference of the min and the max of all the system. This degree informs about the good or not distribution of throughput constraints in one cluster. As, the best is to maximize a global distribution on clusters, we check the minimum value of all throughput

distribution. This value corresponds to the metric and enables to analyze the throughput constraint distribution.

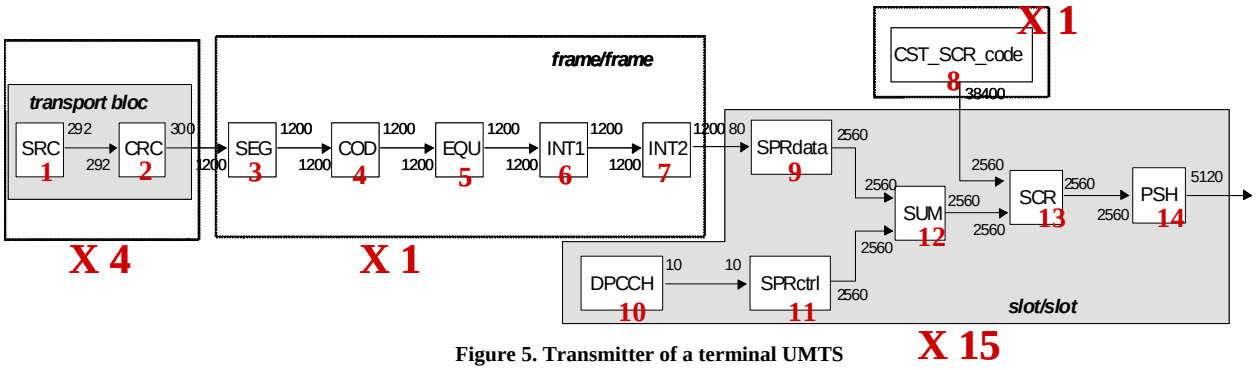

Figure 5. Transmitter of a terminal UMTS

5. METRICS APPLICATION: UMTS

In this section, we validate the proposed metrics and we demonstrate that they enable to make an efficient analysis of the system performances at the tasks level. This validation is performed using an UMTS application (uplink transmitter). To apply these metrics we first specify the UMTS application using the proposed tasks graph. Then, we perform several experimental results on the UMTS application to demonstrate the efficiency of these metrics to exhibit the different characteristics that we have considered (communication, memory, throughput).

5.1. UMTS specification graph

In this section, we do not describe the algorithms of the tasks that compose the UMTS application since we do not need this information to perform our analysis. However, we suppose that the affinity metrics have already computed by for example the method proposed by Sciuto et al. [13]. To define the specification graph, we only need to describe the data dependencies and tasks proprieties as shown in figure 5.

The transmitter of the uplink terminal UMTS is composed of 14 tasks and 13 edges. Each task is labeled with a figure in order to simplify the discussion about the results. As noted by ×4, task 1 and 2 are executed 4 times during one execution time of the application. Tasks 3 to 8 are executed only one time and the other tasks 15 times (tasks 9 to 14). Edges are labeled with the data quantities that are transferred. Two values are associated with each edge, one corresponds to the data emitted and the second one to the data received.

The UMTS application is under a real time constraint since the whole application has to be executed every 10 ms. It leads to apply some throughput constraints to different tasks: task1 - 114 kbits/s, task8 - 37 Mbits/s, task10 - 15 kbits/s and task14 - 8 Mbits/s.

Concerning the specification of the application no other information is required. The analysis can be performed.

5.2. Experimental results

In this section we have applied the metrics to different clustering examples to show their ability to efficiently evaluate the characteristics of the application.

As a first example we consider a 2 clusters solution that is described in table 1. The values presented in table 1 correspond to intermediate results required to compute the metrics presented in table 2. These intermediate values are presented in order to enable the discussion that follows about the metrics. Note that in the case of a real utilization of the metrics these intermediate values are not provided to the designer.

	Cluster 1	Cluster 2
Tasks	1-2-3-4-8-13-14	4-5-6-7-9-10-11-12
Data (bits)	80368	80550
ComDegree(p)	0.4008	0.4017
Shared_mem(p)	39600	1200
Mem_dedree(p)	0.9705	0.0294
Min(tc)	116800	15000
Max (tc)	38400000	15000
Throughput degree(p)	0.9969	0
Met_aff(p)	0.910714	1

Table 1. Characteristics of the application for a 2 clusters solution

In this example, the exchanged data quantity in each cluster is nearly the same as shown in the Data line in table 1. Hence, exchanged data is well balanced for this clustering example. This result is highlighted by the E_De metric since its value is equal to 0.9977. It demonstrates the efficiency of E_De to evaluate the good balance of data exchange in clusters. The inter clusters data exchange is 39600 bits (or 19.74 percent of the total exchanged data in the system). Hence, it is normal that the DEIC value is not equal to 1 but to 0.8. However this value is close to 1 since even if the solution does not correspond to the best one, it is still a good compromise. We can say that the DEIC metric takes correctly into account this trade-off. The same remarks can be done for the CIC metrics.

MEM and TC, as shown in table 2 are nearly equal to 0. It means that the considered clustering does not optimize the memory and the throughput distribution. In fact, as shown in table 1, nearly the totality of the memory requirements is in the cluster 1, and nearly nothing in the second. So, the memory distribution is not balanced. Furthermore, in cluster 1 there are 3 different throughputs : 37 Mbits/s, 8 Mbits/s and 114Kbits/s that is close to the minimum throughput of the application which is on cluster 2

(15 kbits/s). Hence, the throughput distribution is also not balanced for this clustering as shown by the metric TC.

E_De	DEIC	CIC	MEM	AFF	TC
0.9977	0.8025	0.8461	0.0303	0.9107	0.0031

Table 2. Metrics for example 1

This first part has demonstrated the ability of the proposed metrics to analyze the clustering efficiency. Below, we further discuss this point since we show that when the number of clusters is not adapted to the application some metrics are closed to 0. A manual analysis of the UMTS specification graph shows that exchanged data sizes are so different that it is not possible to balance the exchanged data sizes when the system in partitioned in 3 clusters. To demonstrate this analysis and to show the ability of the E_De metric to detect such feature different examples of 3 clusters are considered in the following. In the table 3, each clustering example is represented as a vector of 14 values, where each value represents the cluster whose the task belongs to. For example, in the first vector the task 1 belongs to the cluster1, task 2 to the cluster 1 … Table 3 presents the E_De value for these different examples. As expected, the E_De value is always close to 0.

Example	E_De
11333322222211	0.0460
11111122223333	0.0104
11223311223332	0.0149

Table 3 . Some examples of 3 groups clustering

We can notice that even if the solutions are not efficient, we can still compare them and for the considered examples, the first solution is the best one.

The last part of the experimental results section focuses on the ability of the metric to compare different clustering solutions. The proposed metrics are very interesting to analyze and to compare the clustering possibilities. In table 4 another example of a 2 clusters solution is proposed for which we have calculated the metrics (and the intermediate results in order to discuss these metrics) to compare with the first example discussed above.

	Cluster 1	Cluster 2
Tasks	1-2-3-4-5-6-7	8-9-10-11-12-13-14
Data (bits)	7168	192150
ComDegree(p)	0.0357	0.9582
Shared_mem(p)	1200	39600
Mem_dedree(p)	0.0294	0.9705
Min(tc)	116800	15000
Max (tc)	116800	38400000
Throughput degree(p)	0	0.9996
Met_aff(p)	1	0.0004

Table 4. Characteristics of the application for a 2 clusters solution

In table 5 the different metrics are presented for that second example. As we can see even if the DEIC, CIC and AFF metrics are very good, this solution is not as efficient as the first one since the E_De and the MEM metrics are lower than the values of the first solution. Hence, thanks to these metrics it is straightforward to compare different solutions.

E_De	DEIC	CIC	MEM	AFF	TC
0.0373	0.9944	0.9285	0	0.9107	0.0031

Table 5. Metric for example 2

6. CONCLUSION

Due the complexity of both the applications and the systems, classical codesign flows are not enough efficient to design SoC system without first modifying the application or adding some assumptions on the architecture. That's why an earlier, fast and automatic analysis step is essential to identify the application's specificity in order to help the codesign flow.

To perform such an approach, we propose in this paper, several metrics (communications, memory, affinity and throughput) which allows analyzing the system characteristics at a tasks level. These metrics are computed to evaluate the impact of the tasks and application proprieties onto the performance and the costs of the system. They don't estimate the system costs as power, time or area. But, they measure the impact that some application proprieties as communication and memory sharing have onto the system costs. They also measure the impact onto the system costs of task proprieties (as task affinity) and of interaction between task affinity within a cluster. Analyzing the application and the task characteristics the designer can optimize the application for a SoC codesign flow.

The metrics have been validated on an UMTS application. Results have demonstrated that the metrics are able to evaluate efficiency the characteristics analyzed and that they enable to compare different tasks clustering solutions.

This work is part of a design space exploration approach that has been developed within the DesignTrotter environment. DesignTrotter starts from a task graph and leads to the definition of the architecture (computing, memory and communication resources) and the hardware-software partitioning of the application. As the actual researches, our research is focused on the new challenges of embedded real-time system consequence of the new technologies. This work is a first step of approaches which is based on application analysis to predict the impact of the application on the system performances and costs independently of the technology and execution support.

7. REFERENCES

[1] "AMBA 2.0 Specification", www.arm.com/armtech/AMBA

[2] "Peripheral Interconnect Bus architecture", www.sussex.ac.uk/engg/ research_groups/vlsi/projects/pibus/

[3] “The AVALON bus specification”, www.altera.com/products/ devices/excalibur/features/exc-nios_avalon_bus.html

[4] “Sonics µNetworks, Technical Overview”, www.sonicsinc.com

[5] John Dielissen, Andrei Radulescu, Edwin Rijpkema, Kees Goossens, " Concepts and Implementation of the Philips Network-on-Chip", International workshop on IP based system on chip design. Grenoble, France, November 2003.

[6] Peter Voigt Knudsen and Jan Madsen. Integrating communication protocol selection with hardware/software codesign. In IEEE Transactions on Computer- Aided Design of Integrated Circuits and Systems, pages 1077–1095, August 1999.

[7] Yannick Le Moullec, Nader Ben Amor, Jean-Philippe Diguet, Mohamed Abid et Jean-Luc Philippe « Multi-Granularity Metrics for the Era of Strongly Personalized SOCs» Design, Automation and Test in Europe Conference (DATE 03), Munich, Allemagne, 3-7 Mars 2003.

[8] L.Carro, M.Kreutz, F.Wagner and M.Oyamada, "System Synthesis for Multiprocessor Embedded Applications", DATE'00, Paris, France, Mar. 2000

[9] D.Sciuto, F.Salice, L.Pomante and W.Fornaciari, "Metrics for Design Space Exploration of Heterogeneous Multiprocessor Embedded Systems", CODES’02, Estes Park, USA, May 2002

[10] B.P.Dave, G.Lakshminarayana and N.K. Jha, “COSYN : Hardware software Co-synthesis of heterogeneous distributed embedded systems”, In IEEE Transaction on software Engineering, volume 7, mars 1999.

[11] B.P.Dave and N.K. Jha, “Casper : concurrent Hardware software Co-synthesis of embedded system architectures”, In book Design, Automation \ & Test in europe conf. Feb 1998

[12] F.Vahid and D.D.Gajski, "Closeness Metrics for System-Level Functional Partitioning", EDAC'95, U.K, Sep. 1995