Multi-Granularity Metrics for the Era of Strongly Personalized SOCs

Y. Le Moullec[1], N. Ben Amor[1,2], J-Ph. Diguet[1], M. Abid[2] and J-L. Philippe[1]
[1]Université de Bretagne Sud, Lorient, France, [2]ENIS engineering school, Sfax, Tunisia
Email: moullec@iuplo.univ-ubs.fr

Abstract

This paper details the first step of the Design Trotter framework for design space exploration applied to dedicated SOCs. The aim of this step is to provide metrics in order to guide the designer and the synthesis tool towards an efficient application architecture matching. This work presents a computation of metrics at all levels of the application graph-based hierarchy. These metrics are computed through data and control dependency analysis. They quantify the memory, control and processing orientations as well as the average of parallelism for different granularities.

1. Introduction

The domain of embedded systems is driven by the requirement of fast design methodologies guarantying optimized chips in terms of energy, area and real-time constraints. This matter of fact leads to an increasing use of CAD tools which enable to set rapidly parameters such as the selection of instructions, the capabilities of local or I/O memories, the bandwidth of communication channels, the parallelism of processing units and the choice of dedicated hardware. This type of option-based methodology can be applied to various targets including IPs, ASIPs, DSPs, processor cores and associated compilers. Moreover, we observe an evolution of programmable chips which i) enables a massive use of parallelism versus relatively low clock frequencies and ii) permits a tolerable ratio of deficient gates while insuring reliability. So, regarding this trend towards personalization of chips, it arises that the frontier between software and hardware modules is gradually disappearing. Consequently, the design issue is no more software or hardware selection but turns into the selection of the amount, the nature (spatial, pipeline, …) and the granularity of parallelism for processing and bandwidth resources that fit with the application constraints. Moreover, by improving the application-architecture matching, the system designer also alleviates high clock frequency requirements and consequently reduces the design effort at the gate level. So, in a design space exploration strategy, a first step consisting in a metric-based analysis can be performed rapidly without any architectural directive. In a second step the results can be used to sketch the target architecture in order to perform a first set of estimations. Then the aim of the metrics is to stress the proper architecture style for a given function or task. These features include the wider/deeper trade-off, namely the ratio of explicit parallelism versus the pipeline depth, the necessity of complex control instructions, the requirements in terms of local memories and specific bandwidth or the need of processing resources for address generation.

Previous work have been completed around metrics in the areas of high-level synthesis [1,2] and hardware software codesign [3,4,5]. In [1] the metrics provide algorithm properties regarding a hardware implementation, the quantified metrics address the concurrency of arithmetic operations based on uniformed scheduling probabilities and the regularity that measures the repetition rate of a given pattern. In [2], some probability based metrics are proposed to quantify the communication link between arithmetic operators (through memory or registers). These metrics focus on a fine grain analysis and are mainly used to guide the design of datapaths especially to optimize local connections and resource reuse. The metrics from [3] are computed at the functional level to highlight resource, data and communication channel sharing capabilities in order to perform a pre-partitioning resulting in functions clustering to guide the next design step, which is hardware/software partitioning. The main issue is the placement of *close* functions on the same component in order to optimize communications and resource sharing. An interesting method for processor selection is presented in [4]. Three metrics representing the orientation of functions in terms of control, data transformation and data accesses orientation of functions are computed by counting specific instructions from a processor independent code. Then a distance is calculated, with specific characteristics of processors regarding their control, bandwidth and processing capabilities. In this framework a coarse and fixed granularity level is considered and the target is limited to predefined processors. Moreover the technique does not take into account instruction dependencies and there is no detail about the different types of memory accesses regarding the abstract processor model used. However we can reuse the concept of distance during the design steps located at lower levels. Finally, in [5] finer metrics are defined to characterize the *affinity* between functions and three kinds of targets : GPP, DSP and ASIC. The metrics are the result of the analysis and counting of C code instructions in order to highlight instruction sequences which can be DSP-oriented (buffer circularity, MAC operations inside loops, etc.), ASIC-oriented (bit level instructions) or GPP-oriented (conditional or I/O instructions ratio). Then a HW/SW partitioning tool is driven by the *affinity* metrics. Like in [4] these metrics are dedicated to HW/SW partitioning, they do not exploit instruction dependencies and address a fixed (C procedures) granularity. Moreover, the locality of data bandwidth is not clearly taken into account.

This paper deals with the first step of the *Design Trotter Framework,* namely the *characterization* step. This framework is devoted to the design of strongly personalized SOCs, therefore the introduction of architectural directives is avoided as much as possible during the first analysis step. Thus, we propose guidance metrics based on a hierarchical graph specification of the application in order to highlight parallelism opportunities at different levels of granularity. We define metrics independent from a target architecture in order to drive the platform specification. Theses metrics focus on local/global data-transfers, data-processing and control operations while considering data and control dependencies.

2. Efficient graph-based specification

2.1 Definitions

Each C function of the specification is a node at the top level of the Hierarchical Control and Data Flow Graph (HCDFG). A function is a HCDFG. A HCDFG is a graph that contains only HCDFGs and CDFGs. A CDFG contains only elementary conditional nodes and DFGs. A DFG contains only elementary memory and processing nodes. Namely, it represents a sequence of non-conditional operations. There are three kinds of elementary (i.e., non-hierarchical) nodes of which the granularity depends on the architectural model: a processing node represents an arithmetic or logic operation, its granularity depends on the architectural model: (ALU, MAC, +, -, etc.). A memory node represents a data transfer (memory operation). Its parameters are the transfer mode (read/write), the data format and the hierarchy level that can be fixed by the designer. A conditional node represents a test operation (if, case, loops, etc.) There are also three types of dependencies represented by edges: a control dependency indicates an order dependency between operations without memory transfers (e.g., index computation before array access). Control dependency edges can also be used to impose an order between independent operations or graphs in order to favor resource usage optimization. A scalar data dependency between two vertex A and B indicates that node B uses a scalar issued from B vertex. A multi-dimensional data dependency is a data dependency where data produced is not a scalar but an array. Such an edge is created between a loop CDFG that reads an array produced by another loop CDFG.

2.2 Graph creation rules

The graph is traveled with a depth-first search algorithm. A HCDFG/CDFG is created when a conditional node is found at the next hierarchy level. When no more conditional nodes are found, a DFG is built. In order to facilitate the estimation process, CDFG patterns have been defined to identify rapidly *loop, if etc.* structures. Another important point is that the model covers the complete application complexity. Thus, index computation (address computation), conditional tests and loop index evolution are represented with DFGs. We distinguish several types of memory nodes:

1. input/ouput nodes (N1)
2. temporary data (produced by computations) (N2)
3. re-usable data (re-used input nodes) (N3)
4. accumulator data (N4)

N1 data are always global, N4 data are always local, N2 and N3 data can initially be local (stored in the register file) but they can be moved to the global memory if ever the local memory size becomes too small as compared to the application's requirements. The first step of the metric calculation is located at the highest level of abstraction, without any architectural assumption. So, the data accesses considered are the global ones, corresponding to N1 data nodes. The metrics that we will define are independent from the type of transfers (synchronous, asynchronous, buffered) and from the memory hierarchy. These parameters will be chosen later on, after the application characterization which is the aim of our system-level metrics. A HCDFG example is depicted in Fig. 1.

2.2 Hierarchical Characterization

The HCDFG representation enables multi-level granularity specification and characterization. Therefore the notion of function can correspond to several levels of granularity. At the lowest level, a function can represent for example a FIR filter. At an intermediate level, a function can represent a DWT. At the highest level, a function can represent a JPEG2K encoder. The scheme used for characterizing the application specification is based on a hierarchical bottom-up approach. The characterization results obtained for a certain level in the specification are combined together in order to characterize its upper level. The lower level characterization is performed with a fine grain granularity. At that level, the type of operations can be either processing (shifting, multiplications etc.) or data transfer. Once the lower levels have been estimated, the higher levels are estimated through combinations. This step can be performed rapidly as the information relevant to each low level function has been saved within its graph.

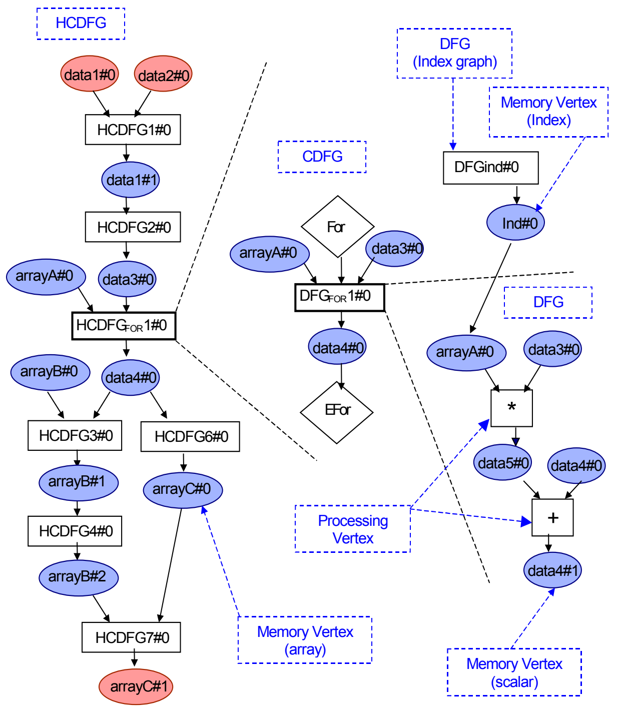

Fig. 1. a HCDFG specification example

3. Metrics Computation

In this section we define 3 metrics: γ (Average Parallelism Metric), MOM (Memory Orientation Metric) and COM (Control Orientation Metric). We explain how they are computed in the leaf graphs and how they are combined to characterize CDFGs and HCDFGs.

3.1 γ metric

For a DFG graph γ is defined as :

$$\gamma = \frac{Nb\ \ of\ \ global\ \ memory\ \ accesses\ \ and\ \ processing\ \ operations}{Critical\ \ Path} \qquad (1)$$

The critical path, noted CP, in a DFG graph, is the number of cycles of the longest sequential chain of operations (processing, control, memory). CP is computed for each hierarchical level with a data and control dependencies analysis. Our analysis method is not exclusively statistical contrary to [4, 5] metrics. As defined, γ indicates the average spatial parallelism available at a given hierarchy level. For instance, if a HCDFG contains five parallel DFGs where each DFG is fully sequential, then γ equals one for each DFG and five at the HCDFG level. γ enables the classification of application functions according to their criticality, namely their capability to exploit the available parallelism. In the following design steps functions with highest γ can be first considered since they have the most important optimization potential regarding the acceleration and consequently energy savings. Note that it is also used to distribute cycle budgets to functions during the estimation and synthesis design steps of DesignTrotter framework.

Functions with high γ values are considered as appropriate to architectures with large explicit parallelisms. Functions that have a low γ value (circa 1) are rather sequential, so the acceleration can only be reached by exploiting temporal parallelism (i.e. deep pipeline). The parallelism is fully exploited during the next estimation step of our framework based on adaptive scheduling [7].

3.2 combination rules

The metrics are computed using a bottom-top approach, they are firstly calculated for leaf DFGs, then for higher level CDFGs and HCDFGs with combination rules according to sequential, parallel, exclusive and loop structures. Hereafter are introduced combination rules for sequential, parallel, IF and FOR patterns for the computation of γ. The same approach is used for the other metrics.

A "IF" CDFG is composed of three subgraphs. The first one specifies the condition, the two others correspond to the true and false branches.

For this graph, γ is calculated with the following formula:

$$\gamma_{if} = \frac{P_{true}*Nop_{true}+P_{false}*Nop_{false}+Nop_{c}}{P_{true}*CP_{true}+P_{false}*CP_{false}+CP_{c}} \qquad (2)$$

where "P_{true}" and "P_{false}" are the probabilities to execute the true and false branches respectively. The probability values are considered equiprobable by default but can be modified after profiling the application. "Nop_c" and

"$Nop_{true/false}$" are the number of operations (global memory accesses, processing nodes and test nodes) in the condition graph and the conditional branches respectively (a branch can be a HCDFG).

A "FOR" CDFG is composed of three subgraphs: 1) an evolution graph for the loop index computation, 2) a body graph for the loop body and 3) an evaluation graph for the loop test. γ for "FOR" CDFGs is calculated according to formula (3) which is independent from the iteration number since loop unfolding is not considered at this level of the design flow.

$$\gamma_{for} = \frac{(Nop_{evolution} + Nop_{for} + Nop_{evaluation})}{(CP_{evolution} + CP_{for} + CP_{evaluation})} \qquad (3)$$

The computations of γ for "DO-WHILE" and "SWITCH" graphs are generalizations of "FOR" and "IF" formulas respectively. To determine the γ value of a HCDFG graph, we have to analyze its hierarchical structure. In Fig.2 an example of a HCDFG tree representation is given. It is composed of two nested "IF" CDFGs. The algorithm calculates Nop and CP for the following graphs: "IF_DHeq0_CONDITION" and "IF_DHeq0_TRUE" since they are simple DFGs and do not contain any subgraph. The "IF_Dheq0_FALSE" graph contains a subgraph ("IF_TMPsup"), therefore our algorithm goes down into the hierarchy, determines Nop_{IF_TMPsup} and CP_{IF_TMPsup} values. Then $Nop_{IF\_Dheq0\_FALSE}$ and $CP_{IF\_Dheq0\_FALSE}$ are computed. Finally, Nop_{IF_DHeq0} and CP_{IF_DHeq0} (and therefore γ_{IF_DHeq0}) can be determined. This approach is recursively applied to the whole graph in order to compute its metric values. A HCDFG can be made of sequential and parallel graphs. For sequential graphs we use formula (4) to calculate γ. If there are parallel graphs (or combination of parallel and sequential graphs) we use formula (5):

$$\gamma_{seq} = \frac{\sum_{subgraphs} Nop_i}{\sum_{subgraphs} CP_i} \quad (4) \qquad \gamma_{par} = \frac{\sum_{subgraphs} Nop_i}{\mathrm{Max}_i\{CP_i\}} \quad (5)$$

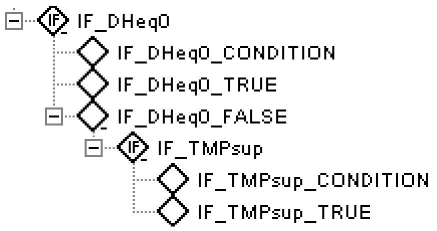

Fig. 2. Nested IF HCDFG coding

3.3 MOM (Memory Orientation Metric)

This metric is defined for a DFG by the following formula (6) :

$$MOM = \frac{Nb\ of\ global\ memory\ accesses}{Nb\ of\ processing\ operations + Nb\ of\ global\ memory\ accesses} \qquad (6)$$

MOM indicates the frequency of memory accesses in a graph. MOM values are normalized in the [0;1] interval. The closer to 1 MOM is, the more the function is considered as data-access dominated. Therefore in the case of hard time constraints, high performance memories are required (large bandwidth, dual-port memory, etc.) as well as an efficient use of memory hierarchy and data locality [6].

To calculate MOM for {H}CDFGs, we follow the same approach as for γ computation. For a DFG graph, the global memory and processing nodes are enumerated and saved as graph attributes. Then the MOM value is computed for the DFG. These attributes are used to compute MOM metric values for graphs located at higher hierarchical levels. We detail MOM computations for two CDFG cases: "IF" CDFGs, and "FOR" CDFGs.

For "IF" CDFGs, the formula used to calculate MOM is given in (7)

$$Mom_{if} = \frac{Nm_{true} * p_{true} + Nm_{false} * p_{false} + Nm_c}{\sum_{x=m,p,t} Nx_{true} * p_{true} + Nx_{false} * p_{false} + Nx_c} \qquad (7)$$

where $Nm_{true/false/c}$, $Np_{true/false/c}$ and $Nt_{true/false/c}$ are the numbers of global memory accesses, processing operations and test operations in the branch(true), branch(false) and conditional graph respectively. For the "FOR" CDFG, we use formula (8):

$$Mom_{for} = \frac{Nm_{evaluation} + Nm_{for} + Nm_{evolution}}{\sum_{x=m,p,t} Nx_{evaluation} + Nx_{for} + Nx_{evolution}} \qquad (8)$$

To determine HCDFGs MOM values, the algorithm goes through the hierarchy, extracts the Nm and Nt attributes from the different graphs and then calculates the equivalent MOM with formula (9):

$$Mom = \frac{\sum_{subgraphs} Nm_i}{\sum_{subgraphs} Nm_i + Np_i + Nt_i} \qquad (9)$$

3.4 COM (Control Metric Orientation)

To calculate this metric, test operations, namely the following operators: <=, <, >, >=, !=, ==, must be identified. COM is defined by the general formula (10):

$$COM = \frac{Nb\ of\ test.op.}{Nb\ of\ global\ mem.\ accesses + Nb\ of\ proc.\ op. + Nb\ of\ test\ op.} \qquad (10)$$

It indicates the appearance frequency of control operations (i.e., tests that cannot be eliminated during compilation) in a CDFG or HCDFG since there is no test within a DFG. COM values are normalized in the [0;1] interval. The closer to 1 COM is, the more the function is control dominated, so needs complex control structures. It also indicates that the use of the pipeline technique is not efficient for such functions. We detail COM computations for two CDFG cases: "IF" CDFGs, and "FOR" CDFGs. For "IF" CDFGs, COM is calculated with formula (11):

$$COM_{if} = \frac{Nt_{true} * p_{true} + Nt_{false} * p_{false} + Nt_c}{\sum_{x=m,p,t} (Nx_{true}) * p_{true} + (Nx_{false}) * p_{false} + Nx_c} \quad (11)$$

For "FOR" CDFGs, COM is given by formula (12):

$$COM_{for} = \frac{Nt_{for}}{\sum_{x=m,p,t} Nx_{evaluation} + Nx_{for} + Nx_{evolution}} \quad (12)$$

Note that Nt is calculated (in the case of "FOR" CDFGs) only for loops of which the number of iterations can not solved at compilation time. Actually, all control (test) operations within deterministic loops can be removed with unfolding. The COM computation for a complex graph uses the same approach as for MOM computation (eq.13).

$$COM = \frac{\sum_{subgraphs} \mathrm{Nt_i}}{\sum_{subgraphs} Nm_i + Np_i + Nt_i} \quad (13)$$

3.5 Additional metrics

Other useful metrics have been implemented in our framework but cannot be detailed due to the paper length restriction:

- The DRM metric includes a local memory estimation based on data life times and provides a data-locality ratio [7].
- The AOM is the ratio of index computation that stresses address generator requirements.
- Data format histograms.
- Simple operation counting.

4. Experimental results

We have applied the previously defined metrics to functions widely used in embedded systems. Hereafter we present results for Wavelet (DWT) and 2D-DCT transforms as well as for a G722 audio decoder and the TCP protocol.

The DWT algorithm is implemented with the lifting scheme and table 1 shows the results for the different functional blocs and for the whole HCDFG. The first observation is that the COM metric is null for all graphs, since this application is composed of deterministic loops and does not contain any test. Secondly we observe that MOM values for the wavelet functional blocs are higher than 0,7; this means that more than 7/10 of operations are data accesses, so the application is clearly, at all levels, memory oriented. Finally, γ values are around 1,5 for all the functional blocs, this indicates a weak fine grain spatial parallelism. However γ increases at the second level of granularity (greater than 2,7 for the DWT graph), this indicates that a coarse grain parallelism is available. We can now conclude about a way to specify the abstract architectural model for the second step of the general framework, namely the system-level estimation [7]. The fact that there is no need for complex control structures, the high data-accesses requirements and the coarse grain parallelism mean that optimizations can be obtained with a pipelined architecture with possible coarse grain dedicated hardware modules providing a large bandwidth. So if high performances are required, an ASIP or a programmable dedicated hardware can be introduced within the SOC.

The results for the G722 decoder (H320 standard) are given in table 2. This application is composed of many functional blocs, hereafter we present the results obtained for the adaptive predictor bloc. We can notice that the results are quite similar to the previous example. Indeed we observe also high MOM values and that parallelism is weak at fine grain level and increases at the highest level of the specification.

The third application is a 2D-DCT for 8x8 image blocs (table 3). From a HCDFG point of view, it is composed of two identical and sequential 1D-DCT subgraphs, so all the graphs have the same metric values. This third experience is interesting since, like previously, the results are identical for the COM metrics which are null (all tests can be eliminated), the MOM values are also quite high and denote large bandwidth requirements but the γ values are different. In fact, γ is greater than 5 at the lowest levels and does not increase at the second level of granularity. It means that a fine grain spatial parallelism is available if large data accesses are provided. So, if performances are a key point, the designer can efficiently implement this algorithm with an architecture presenting five fine grain processing units. The COM metric values indicate that for communication a pipeline structure or local small memories can be introduced.

Finally we have computed the TCP protocol metrics in order to test another kind of classical application. Each function represents a TCP state. Table 4 shows the analysis result for some representative functions. We can notice that the functions have relatively high COM values denoting heavily conditioned data-flows. The MOM metric values (greater than 1/3) also indicate an important data accesses frequency. It indicates that these functions are control-oriented and require high memory bandwidth. So, a suitable target architecture is a GPP (General Purpose Processor) powered by efficient I/O devices. There is no need for a DSP and for a complex data path structure, since the parallelism cannot be exploited at any level. Note that another very efficient architecture could be implemented using a dedicated FSM associated with fast FIFO.

5. Conclusion

In this paper, we have presented the first step of the *Design Trotter* framework for design space exploration of SOCs. This step provides an application characterization which is based on the analysis of a hierarchical control data-flow graph (HCDFG). For all granularity levels, the characterization provides firstly measures of control, processing and data-access orientations and secondly the parallelism intrinsic capabilities. Experiences with classical algorithms show firstly how functions with a high potential of optimization (whatever the hierarchy level is) can be detected and secondly how the characterization can finely highlight architectural opportunities and directions to improve application-architecture matching. Note also, that the characterization step is also a fast and simple way to compare different algorithmic specifications for a given functional bloc.

6. References

[1] L.Guerra, M.Potkonjak and J.Rabaey, "System-Level Design Guidance Using Algorithm Properties", IEEE Work. on VLSI Signal Processing, San Diego, USA, Oct. 1994

[2] J-Ph.Diguet, O.Sentieys, J-L.Philippe and E.Martin, "Probabilistic Resource Estimation for pipeline architecture", IEEE Work. on VLSI Signal Processing, Sakai, Japan, Oct. 1995

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

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

[5] 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

[6] S.Wuytack, J.Ph.Diguet, F.Catthoor and H.De Man, "Formalized Methodology for Data Reuse Exploration for Low-Power Hierarchical Memory Mappings", IEEE Trans. on VLSI Systems, Vol . 6, No.4, pp 529-537,Dec. 1998

[7] Y. Le Moullec, J-Ph. Diguet, D. Heller and J-L. Philippe, "Fast and Adaptive Data-flow and Data-transfer Scheduling for Large Design Space Exploration", GLSVLSI'02, April 18-19, New-York, http://lester.univ-ubs.fr:8080/~moullec/glsvlsi02.pdf

Functional bloc (graph)	MOM	COM	γ
HFirstLiftingStep_FOR11	0,721	0	1,576
HFirstDualLiftingStep_FOR21	0,721	0	1,576
HSecondLiftingStep_FOR31	0,721	0	1,576
HSecondDualLiftingStep_FOR41	0,722	0	1,579
HScaling_FOR51	0,802	0	2,136
HRearrange_FOR61	0,904	0	1,843
VFirstLiftingStep_FOR71	0,721	0	1,576
VFirstDualLiftingStep_FOR81	0,721	0	1,576
VSecondLiftingStep_FOR91	0,721	0	1,576
VSecondDualLiftingStep_FOR10_1	0,722	0	1,579
VScaling_FOR11_1	0,802	0	2,136
VRearrange_FOR12_1	0,904	0	1,843
dwt	0,765	0	2,704

Table 1. DWT metrics

Functional blocs	MOM	COM	γ
parrec_recons	0,714	0	2,333
upzero	0,758	0,039	1,686
uppol2	0,674	0,087	2,045
uppol1	0,743	0,086	2,188
filtez	0,688	0	1,375
filtep	0,5	0	2,000
predic	0,75	0	1,333
predic_sup	0,738	0,037	3,602
predicteur_sup	0,739	0,037	3,621

Table 2. Adaptive predictor (G722) metrics

Functional bloc	MOM	COM	γ
DCT8L	0,575	0	5,714
DCT8C	0,575	0	5,714
DCT8x8	0,575	0	5,714

Table 3. 2D 8x8 DCT metrics

function name	MOM	COM
TCPTIMEWAIT	0,482	0,06
TCPFIN_WAIT2	0,534	0,055
TCPABORT	0,457	0,343
TCPwakeup	0,333	0,556
Tfinsert	0,5	0,01
TCPdodat	0,375	0,06
TCPSENT	0,508	0,320
TCPRESET	0,667	0,148

Table 4. TCP states metrics