UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

Energy-Efficient NoC-Based Systems for Real-Time Multimedia Applications using Approximate Computing

Wagner Ishizaka Penny

Pelotas, 2021

Wagner Ishizaka Penny

Energy-Efficient NoC-Based Systems for Real-Time Multimedia Applications using Approximate Computing

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Advisor: Prof. Dr. Bruno Zatt
Coadvisors: Prof. Dr. Daniel Munari Palomino
Prof. Dr. Marcelo Schiavon Porto

Pelotas, 2021

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

P416e Penny, Wagner Ishizaka

Energy-efficient NoC-based systems for real-time multimedia applications using approximate computing / Wagner Ishizaka Penny ; Bruno Zatt, orientador ; Daniel Munari Palomino, Marcelo Schiavon Porto, coorientadores. — Pelotas, 2021.

142 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2021.

1. NoC. 2. Approximate computing. 3. Machine learning. 4. Video coding. 5. Hardware acceleration. I. Zatt, Bruno, orient. II. Palomino, Daniel Munari, coorient. III. Porto, Marcelo Schiavon, coorient. IV. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Wagner Ishizaka Penny

Energy-Efficient NoC-Based Systems for Real-Time Multimedia Applications using Approximate Computing

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 14 de dezembro de 2020

Banca Examinadora:

Prof. Dr. Bruno Zatt (orientador)
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Daniel Munari Palomino (coorientador)
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Marcelo Schiavon Porto (coorientador)
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. César Augusto Missio Marcon
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Júlio Carlos Balzano de Mattos
Doutor em Ciência da Computação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Sergio Bampi
Doutor em Engenharia Elétrica Microeletrônica pela *Stanford University (USA)*.

Dedico este trabalho aos meus filhos Thiago e Antonella, o sorriso de vocês é o meu sorriso.

AGRADECIMENTOS

Ao ler os agradecimentos de inúmeras outras teses que passaram pelos meus olhos ao longo desses anos, percebo que este pequeno momento, breve, anterior à grande quantidade de conteúdo técnico que uma tese requer (e que está por vir nas páginas seguintes), é, sobretudo, um momento de reflexão, ao qual o doutorando, contendo a ansiedade natural diante do momento de ter avaliado o trabalho de quatro anos de sua vida, o fechamento de um ciclo, se dá ao luxo por um momento: de refletir, de descontrair, de desabafar. Por que todas essas ações e inúmeras outras se traduzem no ato de agradecer. Nada mais humano do que ter gratidão. Quando paramos para analisar a nossa vida, e refletimos sobre todos os conjuntos de ações que nos trouxeram até aqui, um conjunto quase infinito de *ifs* e *elses*, que, quando combinados, resultam exatamente no momento que estamos vivendo, vemos que talvez as coisas não aconteçam ao acaso, e que alguma força maior, invisível, talvez nos conduza. Agradeço primeiramente a Deus, e não é um agradecimento vazio, por mera formalidade. Não acredito num Deus que magicamente faça as coisas acontecerem, que tenha feito cair no meu colo anos de estudo e uma tese magicamente escrita. Acredito, sim, numa força superior que me deu forças, que me proporcionou exatamente os momentos que eu teria (e terei) que viver, porque são exatamente essas as experiências que preciso viver para me tornar um ser humano melhor, afinal, parafraseando uma série que eu adoro (*Dark – Netflix*), "*everything is connected*".

Como não agradecer aos meus orientadores e coorientadores? Ao professor Bruno Zatt, meu orientador, incansável, paciente, dedicado. Com certeza este trabalho não seria possível sem sua magnífica orientação. Foi meu orientador no mestrado, foi meu orientador neste doutorado, e desejo seguir sendo parceiro de pesquisa ao longo dos próximos anos. Ao professor Daniel Palomino, que passou a ser meu coorientador no doutorado, que contribuiu de forma ímpar na execução deste trabalho. Ao professor Marcelo Porto, meu coorientador de mestrado e também de doutorado, sempre incentivando o grupo de pesquisa e cada um de seus membros, igualmente fundamental para a execução deste trabalho. Por falar em grupo de pesquisa, que palavra forte essa: GRUPO. Faço parte de dois grupos de pesquisa na UFPEL, o GACI (Grupo de Arquiteturas e Circuitos Integrados) e o ViTech (*Video Technology Research Group*). Segundo Aristóteles, "o homem é um animal social", ao meu ver não existe frase mais correta, e isso se traduz na pesquisa. Tenho muito orgulho de fazer parte desses grupos, eles são a prova de que o trabalho em equipe é o caminho para o sucesso. Agradeço aos professores Luciano Agostini e Guilherme Corrêa, que, apesar de não me orientarem formalmente neste doutorado, foram fundamentais na minha jornada até aqui, por seus ensinamentos e parceria. Agradeço ao professor Leandro Indrusiak, que me recebeu por seis meses na Universidade de York, Reino Unido, junto ao *Real-*

Time Systems Group, durante meu estágio sanduíche. Foi um curto período no qual, além da experiência de vida que me foi proporcionada, pude incrementar em muito meus conhecimentos. Agradeço por todos seus ensinamentos e parceria. Agradeço aos demais professores que cruzaram meu caminho ao longo de todos esses anos, do ensino básico à graduação, e aos professores do PPGC da UFPEL, por todos os seus ensinamentos.

Segundo Isaac Newton: "se enxerguei mais longe foi porque me apoiei nos ombros de gigantes". Parafraseando Newton, eu afirmo que "se cheguei até aqui foi porque também estive apoiado nos ombros de gigantes", mas neste caso, meus gigantes são minha família. Agradeço a minha amada esposa Janice, que esteve ao meu lado ao longo dos últimos dez anos, suportando todas as etapas (algumas felizes e outras nem tanto) que a vida (tanto pessoal quanto acadêmica) proporciona. Agradeço por seu companheirismo, amor, amizade, compreensão, parceria, e uma série de outras tantas qualidades que eu poderia elencar. Com certeza este trabalho é um pouquinho dela também. Agradeço aos meus filhos Thiago e Antonella por alegrarem cada dia da minha existência. Sempre ouvi os antigos dizerem que o maior legado que os pais podem deixar para os filhos é a educação, e hoje, muito mais experiente do que outrora, enxergo claramente como são verdadeiras estas palavras. Agradeço profundamente aos meus pais, Solmar e Rosa, por todo amor, carinho, suporte e incentivo que me deram até hoje. A conclusão deste trabalho também passa por eles. Agradeço ao meu sogro Jorge e minha tia Sílvia, por todo carinho e admiração ao longo destes anos. Agradeço a minha irmã Amanda e cunhado Éder pelo carinho e amizade.

Voltando às situações da vida em que tomamos decisões que impactam nosso futuro de maneira drástica, volto ao momento no qual decidi cursar o curso técnico de Eletromecânica, lá nos idos de 2005. No curso fui aluno do professor e amigo Vladimir Afonso, quem muitos anos depois, quando eu concluía o curso de Engenharia Elétrica, convidou-me para cursar o mestrado em Computação, da Universidade Federal de Pelotas. Era um enorme desafio, uma mudança de área de pesquisa bastante drástica, que me exigiu enorme dedicação e esforço. Num primeiro momento antes de terminar a graduação, talvez por comodismo mesmo, eu não cogitasse a computação e permanecesse na engenharia elétrica. No entanto, a oportunidade oferecida pelo Vladimir foi desafiadora, foi quando conheci os professores que me orientam e mudei completamente os rumos da minha vida acadêmica. Agradeço à parceria de meus colegas de laboratório, bolsistas de iniciação científica, mestrandos e doutorandos, Ruhan Conceição, Robson Domanski, Luciano Braatz, Anderson Martins, Jones Goebel, Roberta Palau, Narúsci Bastos, Mário Saldanha, Ítalo Machado, Renato Souza, Maicon Cardoso, Murilo Perleberg, Roger Porto, Ísis Bender, Cristiano Santos, Rafael Ferreira, Gustavo Feijó, Marcel Moscarelli, Alex Borges, Carlos Betemps, Mateus Melo, Iago Storch, Douglas Corrêa, Paulo Gonçalves, Thiago Bubolz, e mais algum outro que eu possa estar injustamente esquecendo. A troca de experiências, regada a muito café e momentos de descontração, torna muito mais fácil e produtiva a arte de pesquisar. Uma pena que uma pandemia mundial tenha alterado nossa rotina na universidade. Agradeço também ao grande amigo Gustavo Kunzel, o qual conheci durante o estágio sanduíche, e foi uma parceira brasileira num laboratório com múltiplas nacionalidades coexistindo na terra da rainha. Com certeza o mate que ele levava todas as manhãs fez diferença na produtividade.

Agradeço a todas as instituições públicas de ensino por onde passei, desde a in-

fância até a pós-graduação: Escola Municipal Luciana de Araújo, Colégio Municipal Pelotense, Centro Federal de Educação Tecnológica de Pelotas (CEFET-RS), Instituto Federal Sul-rio-grandense (IFSUL – antigo CEFET-RS), e Universidade Federal de Pelotas (UFPEL). Cada uma destas instituições tem sua parcela de contribuição e importância na minha trajetória. Agradeço ao IFSul-Campus Pelotas e ao curso Técnico em Eletrotécnica, onde sou professor, a oportunidade de afastamento para estudos, fundamental para conclusão deste trabalho. Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de doutorado sanduíche, igualmente fundamental para a execução do presente trabalho.

Por fim, agradeço a todos aqueles que, de uma forma ou de outra, contribuíram para o desenvolvimento deste trabalho.

Se caíste, ergue-te e anda. Caminha para frente. Regressa aos teus deveres e esforça-te a cumpri-los. Ora, pedindo a Deus mais força para a marcha. Muitas vezes a queda é uma lição de vida. Quem cai sente o valor do perdão aos caídos. O futuro te espera... Segue e confia em Deus.
— CHICO XAVIER

ABSTRACT

PENNY, Wagner Ishizaka. **Energy-Efficient NoC-Based Systems for Real-Time Multimedia Applications using Approximate Computing**. Advisor: Bruno Zatt. 2021. 142 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2021.

This thesis presents an energy-efficient NoC-based system for real-time multimedia applications employing approximate computing. The proposed video processing system, called SApp-NoC, is efficient in both energy and quality (QoS), employing a scalable NoC architecture composed of processing elements designed to accelerate the HEVC Fractional Motion Estimation (FME). SApp-NoC architecture is organized using neighbor Tiles, sized to enable scalability across distinct throughput demands - depending on video resolution and frame rate - whereas reaching real-time processing for 4K UHD videos at 120 fps. Approximate computing is deployed using four types of processing elements implemented as dedicated hardware accelerators with distinct levels of approximation, designed based on the application error resiliency analysis. Therefore, two solutions are proposed: HSApp-NoC (Heuristc-based SApp-NoC), and MLSApp-NoC (Machine Learning-based SApp-NoC). At design time, video encoder statistical behavior is used to propose algorithms aiming the tiling definition, to properly size the NoC and to instantiate and place the approximate processing elements within SApp-NoC. At run-time, our application-aware dynamic task-mapping algorithm guarantees real-time processing while reducing energy consumption with low QoS degradation. When compared to a precise solution processing 4K videos at 120 fps, HSApp-NoC and MLSApp-NoC reduce about 48.19% and 31.81% the energy consumption, at small quality reduction of 2.74% and 1.09%, respectively. A set of schedulability analysis is proposed in order to guarantee the meeting of timing constraints at typical workload scenarios. Moreover, our system design methodology is suitable to be applied to other error-resilient processing kernels targeting energy saving with high throughput requirements.

Keywords: NoC. Approximate Computing. Machine Learning. Video Coding. Hardware Acceleration.

RESUMO

PENNY, Wagner Ishizaka. **Energy-Efficient NoC-Based Systems for Real-Time Multimedia Applications using Approximate Computing**. Orientador: Bruno Zatt. 2021. 142 f. Tese (Doutorado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2021.

Esta tese apresenta um sistema de tempo real energeticamente eficiente, baseado em NoC, para aplicações multimídia utilizando computação aproximada. O sistema de processamento de vídeo proposto, denominado SApp-NoC, é eficiente tanto em energia quanto qualidade (QoS), empregando uma arquitetura NoC escalável composta por elementos de processamento projetados para acelerar a Estimação de Movimento Fracionária (FME) do HEVC. A arquitetura SApp-NoC é organizada usando blocos vizinhos, dimensionada para permitir escalabilidade em diversos cenários de demanda - dependendo da resolução do vídeo e da taxa de quadros - atingindo desempenho para o processamento em tempo real de vídeos UHD 4K a 120 fps. A computação aproximada é aplicada utilizando quatro tipos de elementos de processamento, implementados como aceleradores de hardware dedicados com níveis distintos de aproximação, projetados com base na resiliência a erros da aplicação. Dessa forma, duas soluções são propostas: HSApp-NoC (Heuristc-based SApp-NoC), baseada em heurísticas, e MLSApp-NoC (Machine Learning-based SApp-NoC), baseada em aprendizado de máquina. Em tempo de projeto, o comportamento estatístico do codificador de vídeo é utilizado para dividir e dimensionar a NoC adequadamente, e, também, para instanciar e posicionar os elementos de processamento aproximados na SApp-NoC. Em tempo de execução, um algoritmo de mapeamento de tarefas dinâmico baseado na aplicação garante o processamento em tempo real enquanto reduz o consumo de energia com baixa degradação de QoS. Quando comparado a uma solução precisa de processamento de vídeos 4K a 120 fps, HSApp-NoC e MLSApp-NoC são capazes de reduzir em cerca de 48,19% e 31,81% o consumo de energia, com uma pequena redução de qualidade de 2,74% e 1,09%, respectivamente. Um conjunto de análises de escalonabilidade é proposto a fim de garantir o atendimento das restrições de tempo em cenários típicos de carga de trabalho. Além disso, nossa metodologia de projeto de sistema é adequada para ser aplicada a outros *kernels* de processamento resilientes a erros, visando economia de energia em aplicações com alta demanda em desempenho.

Palavras-chave: NoC. Computação Aproximada. Aprendizado de Máquina. Codificação de Vídeo. Aceleração em Hardware.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

2D	2 Dimensions
AI	All-Intra
ANN	Artificial Neural Network
ARFF	Attribute-Relation File Format
ASIC	Application Specific Integrated Circuit
AV1	AOMedia Video 1
AVC	Advanced Video Coding
AVS2	Audio and Video coding Standard 2.0
AUFF	Approximate Unified FME Filters
BD	Bjontegaard Difference
BD-BR	Bjontegaard Difference bit-rate
BD-PSNR	Bjontegaard Difference Peak Signal-to-Noise Ratio
BMA	Block Matching Algorithm
BPU	Basic Processing Unit
Cb	Blue Chrominance
CGRA	Coarse-Grained Reconfigurable Array
CNN	Convolutional Neural Network
COVID	Corona Virus Disease
CPU	Central Processing Unit
Cr	Red Chrominance
CRGA	Coarse-Grained Reconfigurable Array
CTC	Common Test Conditions
CTU	Coding Tree Unit
CU	Coding Unit
DAG	Directed Acyclic Graph
dB	Decibel

DBF	Deblocking Filter
DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
DVFS	Dynamic Voltage and Frequency Scaling
EiB	Exa Binary Byte
FAPP	Approximate FME Filter
FHD	Full-High Definition
FIFO	First In First Out
FIR	Finite Impulse Response
FME	Fractional Motion Estimation
FPGA	Field Programmable Gate Array
fps	Frames per Second
FS	Full Search
GACI	Grupo de Arquiteturas e Circuitos Integrados
GoP	Group of Pictures
GPB	Generalized P and B picture
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
H	Height
HAAC	Heuristic-based Application-aware Approximation Control
HARP	Heterogeneous Accelerator-Rich Platform
HEVC	High Efficiency Video Coding
HCS	High Complexity Sequence
HD	High Definition
HM	HEVC Test Model
HSApp-NoC	Heuristic-based Scalable Approximate Network-on-Chip
HVS	Human Visual System
HW	Hardware
IDCT	Inverse Discrete Cosine Transform
IDR	Instantaneous Decoding Refresh
IDST	Inverse Discrete Sine Transform
IEC	International Electrotechnical Commission
IFSul	Instituto Federal Sul-rio-grandense
IG	Information Gain

IGAE	Information Gain Attribute Evaluation
IME	Integer Motion Estimation
Int	Interpolator
IoT	Internet-of-Things
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
JCT-VC	Joint Collaborative Team on Video Coding
KDD	Knowledge Discovery from Data
KLD	Kullback-Leibler Divergence
LCS	Low Complexity Sequence
LD	Low Delay
LM	Logarithmic Multipliers
ME	Motion Estimation
MC	Motion Compensation
MI	Memory Input
ML	Machine Learning
MLAAC	Machine Learning-based Application-aware Approximation Control
MLSApp-NoC	Machine Learning-based Scalable Approximate Network-on-Chip
MO	Memory Output
MPB	Multi-point Progressive Blocking
MPEG	Moving Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
MSE	Mean-Squared Error
MV	Motion Vector
MVC	Multiview Video Coding
NoC	Network-on-Chip
PDF	Probability Density Function
PE	Processing Element
PSNR	Peak Signal-to-Noise Ratio
PU	Prediction Unit
QoS	Quality-of-Service
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
RA	Random Access

RCL	Residual Coding Loop
RD	Rate Distortion
RDO	Rate Distortion Optimization
RGB	Red Green Blue
RH	Research Hypothesis
RQT	Residual Quadtree
RTA	Response Time Analysis
RTL	Register Transfer Level
S&C	Search and Comparison
SAD	Sum of Absolute Differences
SAF	Store-and-Forward
SAO	Sample Adaptive Offset
SApp-NoC	Scalable Approximate Network-on-Chip
SI	Spatial Index
SNFT	Schedulability breakdown NoC Frequency Tracking algorithm
SoC	System-on-Chip
SQ	Sub-Question
SW	Search Window
TI	Temporal Index
TU	Transform Unit
TZS	Test Zonal Search
UFPEL	Universidade Federal de Pelotas
UHD	Ultra-High Definition
VC	Virtual Channel
VVC	Versatile Video Coding
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
ViTech	Video Technology Research Group
W	Width
WBSN	Wireless Body Sensor Node
WCRT	Worst-Case Response Time
WEKA	Waikato Environment for Knowledge Analysis
WHT	Walsh Hadamard Transform
Y	Luminance

CONTENTS

1 INTRODUCTION

Over the past few years, semiconductor technologies have advanced strongly, with an ever-increasing number of processing cores integrated into a single die. The market for semiconductor devices is one of the fastest-growing in the world, with an expected turnover of up to $403 billion in sales in 2020 (STATISTA, 2020a). Such huge advance affects the electronic consumer market, which presents a fast-growing demand for more powerful electronic devices. Therefore, current devices are now able to handle applications with increasing processing demands, posed by several complex and data/processing-intensive algorithms where performance for real-time throughput is necessary, such as deep learning, video processing, Internet-of-things (IoT), computer vision, networking, and so on.

This fast advance of the semiconductor industry was allowed by the continuous technology shrinking of the transistor size. Indeed, during many years, microprocessors have improved their processing capacity by increasing clock frequency while diminishing gate size. However, this phenomenon did not last forever. Since the 2000s, single-core solutions were no longer enough to provide performance enhancement, facing challenges regarding the so-called *power wall* (ESMAEILZADEH et al., 2011; VILLA et al., 2014). The threshold and supply voltage of transistors did not follow the shrinking in the gate size, breaking the rules prescribed by *Dennard's Scaling Law* (DENNARD et al., 1974)(BOHR, 2007), which originally predicted that the density of power would be kept constant with the increase in the number of transistors. Thus, the increase in power density resulted in an unwanted increase in temperatures reached by digital devices, which, in turn, resulted in increased cooling costs, being even more challenging in embedded systems, given their posed area and power constraints.

The development of energy-efficient devices is of utmost importance nowadays, mainly when considering the fast-paced expansion and popularization of mobile battery-powered devices (like tablets and smartphones) in the market. For instance, these devices were responsible for an Internet traffic that exceeded 19.01 EiB/month in 2018, with an expectation of reaching up to 77.49 EiB/month in 2022 (where 1 EiB is equal to 2^{60} bytes) (STATISTA, 2020b). Moreover, such an increase in Inter-

net traffic has indirectly posed strong energy demand for online (cloud) data centers, e.g., Google, Facebook, Apple, and so on. Since they need to move, process, and store more data, the data centers demand more energy. For instance, data centers around the world were responsible for an annual energy consumption of 200 TWh in 2018, which represented the annual energy consumption of countries like Iran (JONES, 2018). Thus, one can notice that there is a *need for the development of energy-efficient devices*.

The fast advances and popularization of digital devices leveraged the development/improvement of several application domains, with a spotlight for multimedia applications. Multimedia content is widely present in our daily lives, pushed up by the fast popularization of social media, like Facebook, Instagram, and TikTok, as well as video streaming services, like Youtube, Netflix, Amazon Prime, and many others, which have severely changed our current entertainment patterns. In fact, when looking at our current life context, seriously affected by the worldwide COVID-19 pandemic, such expansion has gained remarkable prominence. For instance, global social media content broadcasting has grown 32.6% (in April 2020) since the pandemic has started (STATISTA, 2020c). Moreover, Internet demand regarding online gaming has increased by 71% in the same period (VERIZON, 2020). Besides, according to CISCO (2020), 90% of all global Internet traffic will be related to video broadcasting in 2023. This growing demand for multimedia content, especially video processing, allied to the increasing demand for higher resolutions and frame rates, has led to ever-increasing demands in computational power. In addition, multimedia applications requiring real-time throughput, like live videos, online conferencing, remote surgeries, and satellite coupling, have also arisen, posing harder constraints for the video systems. Note that all these trends show *the need for the development of energy-efficient and processing-efficient solutions targeting multimedia applications*.

The aforementioned new features such as higher resolutions and frame rates required by current multimedia applications have implied an enormous amount of data to be processed and stored. The video coding process arises as necessary to handle such amount of data. For example, an Ultra-High Definition (UHD) 4K uncompressed video, presenting a resolution of 3840x2160 pixels and a frame rate of 60 fps, requires a bandwidth of 7.46 Gbps to be transmitted in real-time; which means that one hour of video would require 3.36 TB of storage (GRELLERT, 2018). Such prohibitive values led to the development of many video coding standards along the years, pursuing higher compression ratios, like the H.264/AVC (Advanced Video Coding) (RICHARDSON, 2003), the High Efficiency Video Coding (HEVC) (SULLIVAN et al., 2012; ISO/IEC-JCT1/SC29/WG11, 2013; SZE; BUDAGAVI; SULLIVAN, 2014), the Google VP9 (MUKHERJEE et al., 2013), the Audio and Video coding Standard 2.0 (AVS2) (HE et al., 2013), the AOMedia Video 1 (AV1) (LAYEK et al., 2017), and

so on. Current multimedia applications, such as video coding, implement complex tools and algorithms, which, in addition to strict QoS (Quality-of-Service) and real-time requirements, make them very complex applications, demanding high computational effort, which leads to high energy consumption/power dissipation regardless the system where the application is being implemented. Note that in this thesis the QoS was measured in terms of coding efficiency, using the BD bit-rate (BD-BR) metric (BJONTEGAARD, 2001), which is widely used by the academic and standardization community to fairly measure the quality losses. This metric informs, in percentage (%), the bit-rate increase or decrease for the same objective quality, when comparing a given encoding scenario with a baseline (see Section 2.7 for a detailed explanation). Such aspect raises new challenges for systems design, especially when dealing with battery-powered devices, regarding the *development of real-time multimedia systems enabling high QoS and low-energy consumption*.

In general, the GPP (General-Purpose Processor) usage as a processing element (PE) may not give real-time performance neither attend energy consumption/power dissipation constraints or high QoS levels, demanded by real-time multimedia systems. Indeed, considering their severe constraints, especially for embedded battery-powered systems, *dedicated hardware acceleration* becomes mandatory. Furthermore, in order to seek for greater energy efficiency ratios, researchers have been allying the hardware acceleration design to the usage of *approximate computing*, which has been seen as an alternative to improve performance/energy efficiency by compromising (in acceptable ranges) the quality of the applications (VENKATARAMANI et al., 2015). Approximate computing exploits the intrinsic error resilience of applications to realize improvements in performance or energy efficiency. Generally, multimedia processing is a suitable application to apply approximate computing since the resilience of the human visual system (HVS) to errors can be explored, by compromising the quality of the application in tolerable ranges, aiming for the computational effort/power dissipation reduction. Therefore, we can infer that approximate hardware acceleration can be seen as a *promising alternative to achieve energy efficiency while keeping high QoS in multimedia applications*.

When handling embedded and high-performance computing applications, e.g., the aforementioned video processing, performance requirements of applications, especially the ones posed by real-time constraints, cannot be satisfied by simply increasing the frequency of the PE (SINGH et al., 2017), which would lead to high power and heat dissipation. In order to overcome these bottlenecks, the exploration of parallel and distributed solutions, deploying multi-/many core systems operating at lower frequencies, has been proposed by chip manufactures. These solutions are interconnected by Networks-on-Chip (NoC), where the cores can communicate with each other (JERRAYA; WOLF, 2004)(BORKAR, 2007). NoCs can interconnect tens to hundreds

of processing cores by an on-chip packet-switching network that allows data to be transferred between the local memory of each core and from/to external memory (INDRUSIAK, 2014); compounding a complex Multiprocessor System-on-Chip (MPSoC), capable of reducing the computational time and meeting the timing constraints of complex applications, suitable for parallel processing (SMEI; JEMAI; SMIRI, 2017). Besides, NoCs are scalable interconnections to support the growing communication demands of System-on-Chip (SoC) components (BOKHARI et al., 2015) while allowing heterogeneous processing elements, tiling, and partial power/clock gating that enable performance/energy scalability. Scalability is desirable for real systems supporting distinct throughput demands such as video processing, which can require, for instance, multiple video resolutions and frame rates. Based on this discussion, *one may conclude that NoC infrastructures are promising solutions for deploying video processing inherently parallel tasks*.

Furthermore, scalability is also desirable in the context of problems posed by the *Dark Silicon* era, i.e., a significant amount of on-chip resources cannot be operated at full performance level at the same time (HENKEL et al., 2015). Besides providing guarantees for scalability, NoC-based infrastructure can also help with the management of dark silicon by facilitating the application of techniques such as power gating in unused circuit islands (BOKHARI et al., 2014).

1.1 Research Question and Research Hypotheses

There are several works in the literature employing NoCs, exploring the inherent parallelism of applications and trying to provide scalabilty and performace improvements. Some of them aim energy/power efficiency using NoCs for general applications (ZHAN et al., 2013; CLARK et al., 2018; ZHENG; LOURI, 2019), other works are more specific, focusing on the usage of NoCs for multimedia applications (YEMLIHA et al., 2008; ALIKHAH-ASL; RESHADI, 2016; PENNY et al., 2019a; MENDIS; AUDSLEY; INDRUSIAK, 2017; BARGE; ABABEI, 2017). Furthermore, NoC-based systems with hardware accelerators as processing elements for video coding are explored by NOURI; GHAZNAVI-YOUVALARI; NURMI (2018); POURABED; NOURI; NURMI (2018); and PENNY et al. (2019b). Approximate computing is being used to provide significant energy/power reduction in many research areas. Approximate dedicated hardware accelerators for multimedia have being proposed by (EL-HAROUNI et al., 2017; PRABAKARAN et al., 2019; SILVA; SIQUEIRA; GRELLERT, 2019; PASTUSZAK; ABRAMOWSKI, 2015; CHATTERJEE; SARAWADEKAR, 2019; PRAVEEN; ADIREDDY, 2013).

However, multiple aspects remain unsolved: ***How to create an energy-efficient system providing high performance considering multiple usage scenarios while***

delivering real-time support with acceptable QoS? In fact, it is a complex question, and, perhaps, a trivial solution would not fulfill all features desired in the answer. To simplify the discussion, we split such a research question into sub-questions (SQ), listed below:

- **SQ1** - How to provide scalability for multiple demands whereas sustaining high performance for extreme scenarios?
- **SQ2** - How to provide energy efficiency with real-time support?
- **SQ3** - How to keep the QoS at acceptable range while reducing the energy consumption?

Based on these questions, we draw a set of research hypotheses (RH) with potential to address each raised challenge.

- **RH1** - Scalability and high performance can be achieved by exploiting the parallelism of the applications. We hypothesize that employing NoC-based solutions on application-specific systems may (i) facilitate parallel processing and communication leading to performance increase whereas (ii) enabling scalability across multiple scenarios by providing ideal infrastructure for sector-based power gating and dynamic voltage and frequency scaling (DVFS). Indeed, the usage of NoC-based solutions can provide great performance increase by splitting and processing application tasks onto several processing elements, taking advantage of the communication efficiency posed by NoCs. Furthermore, scalability can be exploited by developing NoCs supporting different sizes, enabling only the necessary PEs according to application requirements, employing power gating and/or DVFS over the *idle* ones.
- **RH2** - Energy efficiency with real-time support can be obtained by exploring dedicated hardware acceleration applying approximate computing techniques over the applications. The hypothesis is that the usage of dedicated approximate hardware, exploiting multiples levels of approximation, might enable an optimized exploration of the trade-off between energy and QoS. In general, multimedia applications present natural resiliency to errors due to the limited perception of image details by the human visual system (HVS) (BAI et al., 2014). Thus, hardware acceleration employing approximate computing can explore such a characteristic, being a suitable technique to improve the energy efficiency of complex applications like video coding systems.
- **RH3** - The reduction of energy consumption maintaining the application QoS at acceptable range can be addressed by exploring application-specific proper-

ties/behavior with a run-time management. We hypothesize that using heuristic-based and machine learning-based energy/QoS control can efficiently exploit the application characteristics in order to maximize the energy savings while keeping reasonable QoS. The main idea is to dynamically find at run-time regions more suitable for approximate computing (more error-resilient), applying more aggressive approximation in these regions, based on an exhaustive study of the application behavior employing heuristics and machine learning (ML).

Considering the discussed research questions and research hypotheses, we can notice that many research opportunities remain open. The listed hypotheses have led to the goals of this thesis, as discussed in the next section.

1.2 Goal and Novel Contributions

The main goal of this thesis is to ***research and propose architectural solutions that lead to energy-efficient and high-performance application-specific systems with scalable real-time support for error-resilient applications.***

The main contributions of this thesis are summarized in the following.

- Scalable multi-tiled NoC topology: we propose a NoC organized in multiple neighbor Tiles to allow the NoC effective size to scale according to throughput requirements, i.e., the number of active processing elements varies according to performance requirements.

- Design-time instantiation and placement algorithms: we developed algorithms targeting the tiling definition, the PEs instantiation based on application behavior, and the placement pattern, defining the PEs placement in order to reduce communication distances and to guarantee availability of distinct PEs in different tiles configurations.

- Run-time task allocation algorithms: we propose dynamic task-mapping algorithms to allocate tasks onto different PEs considering their approximation level. We exploited rule-based and machine learning-based solutions.

- Multi-level approximate hardware design: Approximate hardware design using different levels of approximation were designed to be used as processing elements (PE) within the NoC. Application-specific knowledge regarding the error resiliency is considered for PEs design.

- **S**calable **A**pproximate **N**etwork-on-**C**hip (SApp-NoC): We present SApp-NoC, an energy-efficient real-time multimedia system built on top of a NoC employing hardware acceleration, using multiples levels of approximate computing,

leveraging application-specific properties/behavior through the use of heuristics and machine learning techniques, leading to the proposal of two solutions: Heuristic-based SApp-NoC (**HSApp-NoC**) and Machine Learning-based SApp-NoC (**MLSApp-NoC**). SApp-NoC system represents a proof of concept designed targeting video coding application - more specifically the Fractional Motion Estimation (FME) step - in order to demonstrate the validity of our research hypotheses.

1.3 Thesis Outline

This thesis is organized as follows:

Chapter 2 presents an overview of the background knowledge required to understand this work. General aspects regarding Networks-on-Chip are presented with the basic concepts necessary to understand the schedulabilty analysis. Additionally, the main aspects regarding approximate computing are also presented. Afterwards, basic knowledge related to video coding targeting the HEVC standard is also provided. Finally, fundamentals regarding machine learning focusing on decision trees are summarized.

Chapter 3 introduces the discussion about the related works. Works focusing on approximate computing for complex systems, from general applications to video processing, are discussed. Next, the NoC-based solutions found in literature are presented. Furthermore, hardware designs for video coding are analyzed, embracing solutions from precise to approximate design. Finally, a comparison listing the remaining research challenges is presented.

Chapter 4 shows the top-level methodology for development of our case study solution: Scalable Approximate Network-on-Chip (SApp-NoC). The developed algorithms targeting the instantiation and placement of processing elements are presented. Additionally, the developed content-based run-time energy/QoS-aware task allocation is also discussed.

Chapter 5 presents the Fractional Motion Estimation (FME) Multi-level approximate hardware accelerators design. The filters, interpolator, and search and comparison designs are detailed. Besides, the synthesis and QoS results are discussed.

In **Chapter 6** are presented the main aspects regarding the NoC design and the method for application modeling. The NoC architecture is discussed and the developed Schedulability breakdown NoC Frequency Tracking algorithm (SNFT) is presented.

Chapter 7 discusses the development of the application-aware approximation control. The adopted heuristics and machine learning using training benchmarks are widely discussed while the generation of the resulting decision trees is also provided. The found solution is tested and the QoS results are shown.

Chapter 8 presents a wide discussion about the main results. The experimental setup is shown and the power/energy/QoS results are analyzed. Results regarding the performance analysis, targeting the schedulabilty tests are also discussed.

Chapter 9 presents the final remarks of this thesis. All contributions are summarized with a discussion comparing the main achievements with the thesis goals, pointing to future research opportunities and challenges.

2 BACKGROUND

This thesis aims the research of architectural solutions in order to achieve energy efficiency and high-performance regarding application-specific systems with scalable real-time support for error-resilient applications. Considering the highlighted research questions and the main hypothesis to solve those problems, we have proposed strategies in order to achieve our main goal. Our case study SApp-NoC embraces the answers to the mentioned questions and aims to proof the feasibility of the proposed solutions.

Since the adopted solutions cover different research fields, a detailed background about the main aspects becomes necessary for a better understanding of this work. The top-level platform of our work is based on networks-on-chip in order to provide tools to explore the application parallelism. Thus, in Section 2.1, we revisit the basics about NoCs, analyzing their main general aspects such as NoC topologies and communication behavior among nodes. Since real-time processing is targeted by our solutions, Section 2.2 brings the background about real-time schedulability analysis performed at NoC-based systems, establishing the ways to analyze their capabilities of meeting timing constraints.

The hardware accelerators compounding the processing elements of SApp-NoC solution are developed in multiples levels of approximation in order to achieve power efficiency. In Section 2.3 we discuss the main subjects regarding approximate computing techniques. Besides, SApp-NoC focuses on video coding as main application due to its resiliency to errors (being suitable for approximate computing) and its inherent parallelism (being suitable for NoCs). Therefore, we revisit the main aspects of video coding, necessary for a complete understanding of our work. Finally, in Section 2.6, we show the basics regarding machine learning, presenting the general aspects necessary for the understanding of the algorithms employed for the optimization of the approximation level decision.

2.1 Networks-on-Chip: General Aspects

Networks-on-chip (NoCs) are common architectural templates for processors with several cores, being, most of times, a scalable and configurable network compounding a flexible platform that can be adapted to the needs of different workloads (JANTSCH; TENHUNEN et al., 2003). The applied NoC topology can vary, depending on system needs and module sizes and placement. Different topologies have been proposed in the literature, as presented in Figure 1, like the regular mesh (Figure 1 (a)), the folded torus (Figure 1 (b)), and the irregular mesh-custom topology (Figure 1 (c)) (BOLOTIN et al., 2004).

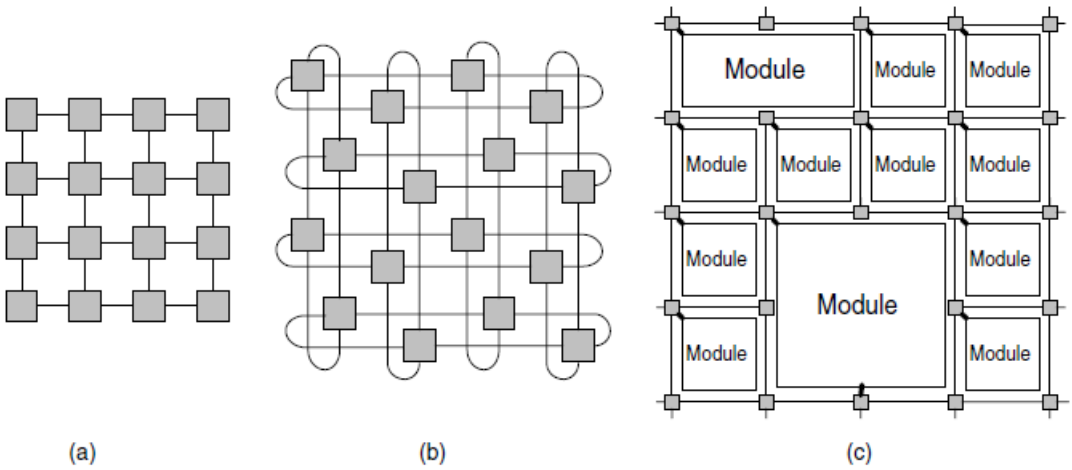

Figure 1 – Examples of NoC topologies: (a) regular mesh; (b) folded torus; (c) irregular mesh-custom topology (BOLOTIN et al., 2004).

Since we have adopted architectural features that are widely used in industry and academia, our focus is put on understanding the widely used 2D mesh topology (BOLOTIN et al., 2004; MORAES et al., 2004; AGARWAL, 2007; SHI; BURNS, 2008; INDRUSIAK, 2014; INDRUSIAK; BURNS; NIKOLIĆ, 2018).

In Figure 2, we show a simplified representation of a simple 3x3 NoC architecture based on a regular 2D mesh topology. All the nodes are interconnected, and each one has a processing element *PE* where the tasks are executed. The *PE* can be, e.g., a central processing unit (CPU), a graphics processing unit (GPU), a hardware accelerator, and so on, where each *PE* is linked to a local cache (blue rectangles attached to *PE*), which stores local information, and a router *r*, which routes the data packets towards the destinations (it can be another *PE*, the off-chip memory, etc.) (INDRUSIAK, 2014). Note that packet is the notation adopted to depict the information crossing the network, widely adopted by literature (BOLOTIN et al., 2004; SHI; BURNS, 2008; INDRUSIAK, 2014; INDRUSIAK; BURNS; NIKOLIĆ, 2018). The communication between the processing elements and the router is made by two unidirectional links (one from *PE* to *r* and other from *r* to *PE*, depicted as orange arrows in Figure 2). The communi-

cation across the network follows an X-Y routing, which is simple and easy to be implemented in the regular topology (SHI; BURNS, 2008). In this work, besides applying the widely used 2D mesh topology, we have also considered the use of wormhole NoCs with priority-preemptive arbitration, widely studied in the literature due to their ability to provide real-time performance (SHI; BURNS, 2008; INDRUSIAK; BURNS; NIKOLIĆ, 2018), since wormhole buffer overhead is much smaller than other approaches like store-and-forward (SAF). Furthermore, link allocation of wormhole switching networks is more efficient than circuit switching approaches, once NoC links are only allocated on the segments of the path where there is data ready to be transferred, i.e., there is no necessity to reserve the complete path of a packet (INDRUSIAK, 2014).

Considering a wormhole switching network, the data are encapsulated into a packet format, further divided into a number of fixed size flits (data words) for transmission. The header flit takes the routing information and governs the route. With the advance of the header along the specified path, the remaining flits follow the same path in a pipeline way. When the header flit encounters a link already in use, it is blocked until the link becomes available (NI; MCKINLEY, 1993). The router is based on priority-preemptive virtual channels (VCs) for flow control (DALLY et al., 1992) as a way to guarantee more predictability. The usage of VC technique decouples the resource allocation by providing multiple buffers for each physical link in the network. Each of these buffers is considered as a virtual channel, holding one or more flits of a packet (SHI; BURNS, 2008).

Each packet has a different priority assigned, allowing packets with higher priority preempt the ones with lower priorities. See in Figure 2 that in each input port there is a FIFO buffer storing the incoming flits of packets arriving through different VCs. The routing and flow controller decides the correct output port for each packet, according to its destination. A credit-based approach based on BJERREGAARD; MAHADEVAN

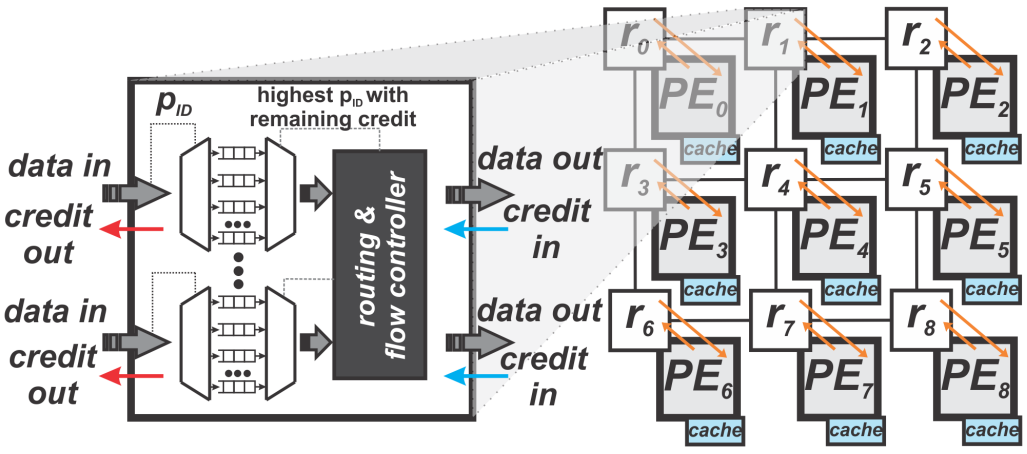

Figure 2 – Example of a mesh 2D NoC 3x3 architecture with a priority-driven VCs router detailing (adapted from INDRUSIAK (2014)).

(2006) was applied, ensuring the forwarding of the data only when there is enough space in the next router's VC. A given flit of a given packet will be sent through its respective output port if it has the highest priority among the other packets being sent out through the same port, and if it also has remaining credits (i.e., there is buffer space available on the respective buffer of the neighboring node connected to that output port). When the highest priority packet cannot send data because it is blocked elsewhere in the network, the next highest priority packet can access the output link (INDRUSIAK, 2014).

As a way to determine if application's tasks being executed and communicating over a specific NoC can fulfill the application's required timing constraints, it is necessary to perform a schedulability analysis, presented in the next section.

2.2 Real-Time Systems: Schedulability Analysis

In general, embedded systems typically have to fulfill timing constraints that are related to their application domain and usage scenarios (INDRUSIAK, 2014). Video processing is a classical example of a real-time application, ranging from scenarios presenting soft timing constraints, like an user watching a video at a streaming platform (when the loss of some frames would not be a huge problem), to safe-critical scenarios, so called hard real-time scenarios, like robotic surgeries or a satellite coupling.

Usually, timing constraints are specified as the deadline to perform specific functions, i.e., the maximum time an application has to execute a set of tasks. For example, a Full-High Definition (FHD) video recorder must be able to capture, compress, and store 30 video frames in a second in order to deliver a real-motion sensation to the user, so there is an expectation that this constraint has to be met by the system in all possible scenarios. Therefore, embedded systems designers must be able to evaluate which design alternatives can fulfill those constraints and, for safety-critical applications, guarantee real-time behavior (INDRUSIAK, 2014). This process which determines the performance estimation of the applications is called *schedulability analysis*.

A NoC-based system is schedulable if, and only if, all its tasks and communicating flows meet the deadlines. It can be determined by applying a performance estimation tool over the application. According to KIASARI; JANTSCH; LU (2013), performance estimation tools can be classified in simulation models and mathematical models. On the one hand, simulation tools are flexible and accurate, however, the complexity of modern SoCs imposes a limitation on what can reasonably be simulated. It also imposes a difficulty to draw conclusions from the simulation results regarding how to adapt the system hardware and its programming, and how to determine the worst-case behavior. On the other hand, mathematical (analytical) models can fairly estimate the desired performance metrics very early on the design phase, in a fraction of the time

that simulation would take (KIASARI; JANTSCH; LU, 2013; INDRUSIAK, 2014).

In this work, we apply the end-to-end schedulability tests proposed in INDRUSIAK (2014), with the improvements of INDRUSIAK; BURNS; NIKOLIĆ (2018). Such methods are based on classic Response Time Analysis (RTA) (AUDSLEY et al., 1993) and on NoC traffic flow schedulability analysis (SHI; BURNS, 2008). The basics on the methodology for modelling the system is given in the following. Next, the end-to-end schedulabilty tests are detailed.

2.2.1 System Model and Notation

A co-design flow of embedded systems, which consists of a set of steps starting with the specifications of requirements and ending with the hardware/software integration into silicon chips (EHRLICH; RADKE, 2013), is of utmost importance nowadays. The co-design modeling phase allows designers to explore the design space, making the best architectural choices in order to meet user requirements, platform and application constraints during the development phase (SMEI; JEMAI; SMIRI, 2017).

In this work, we follow a high level of abstraction modeling based on Sporadic Task Model (SHI; BURNS, 2008; INDRUSIAK, 2014), where an application can be modeled as a taskset defined in (1):

$$\Gamma = \{\tau_1,\ \tau_2,\ ...,\ \tau_n\} \tag{1}$$

Where each task τ_i is a 6-tuple, defined in (2) as follows:

$$\tau_i = \{C_i,\ T_i,\ D_i,\ J_i,\ P_i,\ \varphi_i\} \tag{2}$$

The members of τ_i are respectively the worst-case computation time C_i, the period T_i (minimum inter-release time interval), the deadline D_i, the release jitter J_i (time between the request of a task and it starts to be processed), the priority P_i, and a message φ_i sent by τ_i (immediately after it finishes its computation), defined as a 3-tuple φ_i, depicted in (3):

$$\varphi_i = \{\tau_d,\ Z_i,\ K_i\} \tag{3}$$

Where τ_d is the destination task, Z_i is the message's size and K_i the maximum release jitter (total time the packet takes to reach the destination, including preemption and/or interference). A sporadic task-chain X is an ordered subset of Γ, which can be defined as follows.

$$X = \{\tau_1,\ \tau_2,\ ...,\ \tau_x\} \tag{4}$$

In (4) each task τ_j sends a message to a subsequent task τ_{j+1} in X. In this case,

all tasks within X have the same period and deadline. The final task of every task-chain must be the empty set Ø, conceived as a task outside the taskset scope being evaluated, since for the adopted assumption in this work loops are not allowed.

Still folowing what is stated in SHI; BURNS (2008) and INDRUSIAK (2014), the modelling of the NoC platform (an example was presented in Figure 2) is composed of a set of processing elements Π, a set of switches (routers) Ξ, a set of unidirectional links Λ (between the router and its correspondent processing element and its neighbors routers), depicted in (5), (6), and (7), respectively.

$$\Pi = \{PE_0,\ PE_1,\ ...,\ PE_m\} \tag{5}$$

$$\Xi = \{r_0,\ r_1,\ ...,\ r_m\} \tag{6}$$

$$\Lambda = \{\lambda_{PE0r0},\ \lambda_{r0PE0},\ \lambda_{r0r1},\ \lambda_{r1r0},\ ...,\ \lambda_{PEmrm},\ \lambda_{rmPEm},\ \lambda_{rmr(m-1)},\ \lambda_{r(m-1)rm}\} \tag{7}$$

Where $m = M - 1$ and *M* is the total number of nodes in the NoC. When looking to Figure 2, e.g., if we consider the links connected to node 1, there are two unidirectional links connecting the router to the processing element (λ_{PE1r1} and λ_{r1PE1}), and six unidirectional links connecting the router to other routers (λ_{r1r0}, λ_{r0r1}, λ_{r1r2}, λ_{r2r1}, λ_{r1r4}, and λ_{r4r1}).

The packets are forwarded in the NoC from source to destination according to a routing algorithm. INDRUSIAK (2014) defines a function for routing, e.g. for a communication between PE_0 and PE_1, depicted as

$$route(PE_0,\ PE_1) = \{\lambda_{PE0r0},\ \lambda_{r0r1},\ \lambda_{r1PE1}\} \tag{8}$$

Function $route(PE_0,\ PE_1)$ in (8) denotes a subset of Λ used to transfer packets from core PE_0 to core PE_1. A route includes links connecting the source and destination *PEs* to their respective switches, and all the links between switches along the way. Furthermore, INDRUSIAK (2014) also defines the cardinality of a route as $|route(PE_0,\ PE_1)|$, informally referred as its *hop count*. For example in Figure 2, considering the communication between PE_2 and PE_8, we can define a routing function as:

$$route(PE_2,\ PE_8) = \{\lambda_{PE2r8},\ \lambda_{r2r5},\ \lambda_{r5r8},\ \lambda_{r8PE8}\} \tag{9}$$

Thus, from (9) it can be observed that $|route(PE_2,\ PE_8)| = 4$ for most commonly used routing algorithms (e.g. like the ones prioritizing the minimum distance between

two *PEs*).

Task mapping is a critical part of the design of multicore systems. It defines which application tasks should be mapped onto which processing element (INDRUSIAK, 2014; INDRUSIAK; BURNS; NIKOLIĆ, 2018), i.e., where each task will be executed. The mapping can be defined as a surjective function depicted in (10), which denotes the processing element onto which a task is mapped.

$$map(\tau_i) = PE_j \tag{10}$$

The inverse of this function is defined in (11) and returns all tasks mapped onto a given PE.

$$map^{-1}(PE_j) = \{\tau_i,\ ...,\ \tau_n\} \tag{11}$$

Likewise, the mapping of a message is given as

$$map(\varphi_i) = route(map(\tau_i),\ map(\tau_d)) \tag{12}$$

In (12) it is informed the route of the packets of φ_i across the NoC. The inverse of this mapping function represents all messages traversing a given link, and it is presented in (13).

$$map^{-1}(\Lambda) = \{\varphi_i,\ ...,\ \varphi_n\} \tag{13}$$

Once the mapping of all tasks of Γ is defined, it is possible to calculate the basic communication latency L_i of every message φ_i, which represents the time spent by the message to be completely transferred from its source to its destination, assuming no contention over the NoC links (i.e. as if the message is the only one using the NoC). The actual value of L_i depends on implementation-specific characteristics of the NoC (e.g. link width, time required for a packet header to cross a router, and for a flit to cross a link) (INDRUSIAK, 2014). A common formulation for L_i is given by INDRUSIAK (2014):

$$L_i \ = \ |map(\varphi_i)| \cdot \ l_{hop} + (|map(\varphi_i)| - 1) \cdot \ l_{router} + (Z_i/width) \cdot \ l_{hop} \tag{14}$$

The first term of (14) represents the time taken by the packet header to traverse all the NoC links, expressed as the product of the message *hop count* and the latency l_{hop} for the header to traverse a single link. The second term represents the time taken by the packet header to traverse all NoC routers, being expressed as the product of the number of routers along the path (which is usually the number of hops minus one in most direct networks) and the latency l_{router} for the header to traverse a router. The third term represents the time it takes for the packet payload to follow the header in

a wormhole fashion all the way to the destination, expressed by the message length Z_i (in bits) divided by the link *width*, i.e., the number of bits encapsulated into a single flit, resulting in the number of payload flits of the message, multiplied by the single link latency l_{hop} (INDRUSIAK, 2014).

2.2.2 End-to-End Schedulability Tests for NoC-Based Multicores

From the application domain, we can derive the timing constraints of a given application task-chain, which establish the *end-to-end timing constraint* or the *end-to-end deadline*, e.g., every frame of a video must be processed in 33 ms or less. These constraints need to be met by specific components of the application (i.e. chains of communicating tasks) (INDRUSIAK, 2014). In order to establish whether all task-chains of an application have their end-to-end deadlines met by a particular NoC-based platform configuration, we have to perform the *end-to-end schedulability test* (INDRUSIAK, 2014; INDRUSIAK; BURNS; NIKOLIĆ, 2018). Such test considers the end-to-end latency of each task of a task-chain, i.e., the time a processing element takes to execute a task (*computation latency*) plus the time the NoC takes to transfer all data produced by a task to the next one on the chain (*communication latency*). As aforementioned, the schedulability tests adopted in this work are based on classic Response Time Analysis (RTA) and on NoC traffic flow schedulability analysis.

2.2.2.1 Schedulability of tasks over a processing element

Firstly, the utilization of each *PE* must be verified to check if the processing element is able to handle all tasks allocated in it, given as follows:

$$\sum_{\tau_i \in map^{-1}(PE_a)} \frac{C_i}{T_i} \leq 1 \tag{15}$$

In this example, where C_i and T_i are the worst-case computation time and the period of a task τ_i, respectively, the mapped tasks on processing element PE_a are verified, checking if all of them will be executed until the period ends. Although a given *PE* may execute all the mapped tasks, it may not be able to execute all of them within their deadlines. Response Time Analysis (RTA) (AUDSLEY et al., 1993) techniques can evaluate how much the interference from higher priority tasks can delay the completion time of task τ_i:

$$R_i = C_i + \sum_{\forall\ \tau_j \in hp(\tau_i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j \right) \tag{16}$$

Equation (16) is defined in INDRUSIAK (2014) and calculates the worst case response time R_i of τ_i, where the function $hp(\tau_i)$ denotes the set of all tasks that can preempt τ_i, i.e., the ones mapped onto the same processing element having higher as-

signed priorities. Formally, $hp(\tau_i)$ includes every task $\tau_j \in \Gamma$ where $map(\tau_i) = map(\tau_j)$, and $P_i < P_j$. Thus, by applying what is stated in (16) it is possible to find the longest possible time interval between the release of τ_i and its termination, by adding the computation time C_i of τ_i with the computation times C_j of all releases of tasks τ_j (that could preempt it). Since R_i appears in both sides of (16), iterative solutions like proposed in AUDSLEY et al. (1993) are necessary. Strategies employing RTA has been widely used to test schedulability of uniprocessor and statically mapped multiprocessor systems with fixed priorities (AUDSLEY et al., 1993; SHI; BURNS, 2008; KIASARI; JANTSCH; LU, 2013; INDRUSIAK, 2014; INDRUSIAK; BURNS; NIKOLIĆ, 2018).

In NoC-based systems the communication latency introduced by the NoC when tasks are accessing external memory or exchanging messages are extremely dependent on the adopted task mapping, the communication behavior of the application, and the network congestion. This leads to high variability in communication latencies, which can be of the same order of magnitude of the computational time C_i of the tasks (or even higher) (INDRUSIAK, 2014). Therefore, the schedulability tests of NoC-based systems must also consider the analysis of communicating tasks, as presented in the following.

2.2.2.2 Schedulability of communicating tasks

Many works in literature have been addressing the problem of schedulability tests of communicating tasks for wormhole NoCs, built upon RTA investigations for off-chip interconnection networks (KIM et al., 1998; SHI; BURNS, 2008; KIASARI; JANTSCH; LU, 2013; INDRUSIAK, 2014; XIONG et al., 2017; INDRUSIAK; BURNS; NIKOLIĆ, 2018). The main concern of these investigations is the analysis of the influence of direct and indirect interference of packets with higher priorities over a given packet traversing a NoC, towards finding more precise models for real-time schedulability analysis of communicating tasks.

Considering the routes of any two packet flows φ_i and φ_j, we can define a contention domain $cd_{i,j}$ as a ordered set of links shared by these flows: $cd_{i,j} = route_i \cap route_j$. A direct interference group G_i^D of φ_i is the set of flows presenting higher priority than φ_i, sharing with it at least one network link (i.e. a non-empty contention domain) (KIM et al., 1998):

$$G_i^D = \{\varphi_i \in \Lambda \mid P_i < P_j,\ cd_{i,j} \neq \emptyset\} \tag{17}$$

Likewise, the indirect interference group G_i^I of φ_i can be defined as the set of flows that are not in G_i^D, but interferes with at least one flow in that set (i.e. interferes with the flows directly interfering with φ_i, but not directly with φ_i itself) (KIM et al., 1998;

INDRUSIAK; BURNS; NIKOLIĆ, 2018):

$$G_i^I = \{\varphi_k \in \Lambda \mid \varphi_k \in G_j^D,\ \varphi_j \in G_i^D,\ \varphi_k \notin G_i^D\} \tag{18}$$

The worst case latency S_i of a packet φ_i transmitted over the considered NoC can be found in (19), adapted from XIONG et al. (2017):

$$S_i \ = \ L_i \ + \sum_{\varphi_j \ \in \ G_i^D} \left\{ \left\lceil \frac{S_i + K_j + K_j^I}{T_j} \right\rceil \ \cdot \ \left(L_j + I_{ji}^{down}\right) \right\} \tag{19}$$

The value of S_i can be found by adding the basic latency L_i of φ_i and the latencies L_j of all releases of packets φ_j that could preempt it. It is worth to mention that the release jitter of φ_j can also influence, many times it can preempt φ_i. In (19) two types of release jitter are considered: K_j, which is caused by the execution of the task τ_j (that releases φ_j), and K_j^I, which is caused by aforementioned indirect interference. The value of K_j is the maximum amount of time elapsed between the start of the period of φ_j and its actual release, and since by definition a packet is released immediately after its source task τ_j has finished computation, $K_j \ = \ R_j$. From SHI; BURNS (2008), $K_j^I \ = \ S_j \ - L_j$.

Furthermore, XIONG et al. (2017) have identified that downstream indirect interference, represented by the term I_{ji}^{down} in (19), may cause a single packet of φ_j to directly interfere on φ_i by more than its basic latency L_j. It happens when φ_j suffers interference from any packet φ_k, not interfering with φ_i, and shares links with φ_j downstream from the links it shares with φ_i, addressing a problem referred to as multi-point progressive blocking (MPB) analysis. In this situation, every time φ_j is blocked by φ_k, it can allow φ_i to flow through the network and potentially overtake φ_j flits that had already blocked it earlier (INDRUSIAK; BURNS; NIKOLIĆ, 2018). For a better understanding on this issue, look at the simple example in the following, adapted from INDRUSIAK; BURNS; NIKOLIĆ (2018).

Consider a situation with only three flows φ_i, φ_j and φ_k, shown in Figure 3. Assume that φ_i and φ_j present larger periods and longer packets (therefore larger L_i and L_j) than φ_k. Also assume that the releases of φ_k are not in phase with the others. The priority order is $P_i \ < \ P_j \ < \ P_k$. In Figure 3(a), φ_i and φ_j are released at the same time from PE_0, and the higher priority φ_j gains access to the network, blocking φ_i.

In Figure 3(b), a packet of φ_k is then released, interfering with φ_j (downstream from its contention domain c_{ij} with φ_i). Since φ_k has the highest priority, the flits of φ_j are stopped from using the link between routers 3 and 4, generating *backpressure* on all subsequent flits of that packet of φ_j, forcing them to stay buffered along the route (depicted as stacked dots) all the way to the source in PE_0. Note that now φ_j flits stop using the links shared with the route of φ_i, thus φ_i becomes the highest priority flow

with buffer credits and the routers start transmitting its flits.

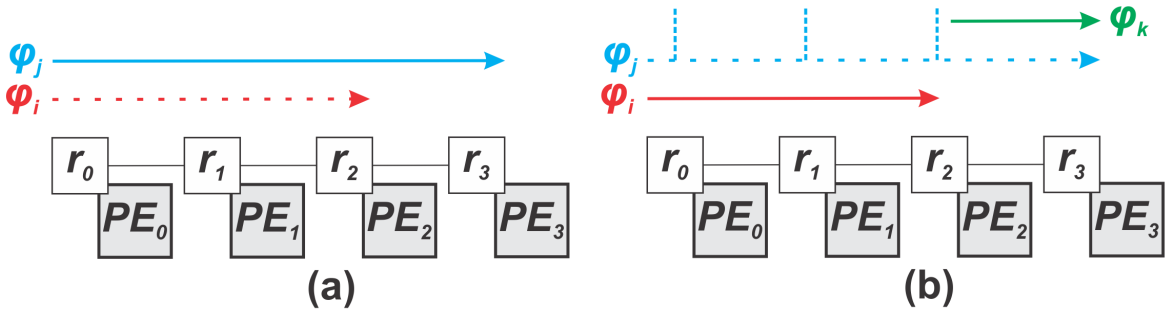

Figure 3 – Downstream indirect interference (adapted from INDRUSIAK; BURNS; NIKOLIĆ (2018)).

When φ_k ends, the scenario goes back to the situation demonstrated in Figure 3(a), with only φ_j flowing through the network. However, before new flits of φ_j can be released from node 0, the buffered flits (due to the blocking imposed by φ_k) must first make way and release the backpressure along the route for upcoming flits from φ_j. This is a key point to the MPB problem, first identified by XIONG et al. (2017) and further improved by INDRUSIAK; BURNS; NIKOLIĆ (2018): these buffered flits of φ_j, which have already caused interference on φ_i when first released out of node 0, will cause interference again on φ_i, and, as a consequence, will delay φ_i by more than L_j (zero-load latency of φ_j). This effect is called as *buffered interference*, which in turn causes MPB.

For calculating the influence of downstream indirect interference I_{ji}^{down} in (19) we have followed exactly what is stated in INDRUSIAK; BURNS; NIKOLIĆ (2018), applying the so called Indrusiak Burns Nikolic (IBN) analysis:

- When computing downstream indirect interference caused by flows that do not suffer from both upstream and downstream interference, I_{ji}^{down} is calculated as follows:

$$I_{ji}^{down} = \sum_{\varphi_k \in G_{I_i}^{down_j}} \left\{ \left\lceil \frac{S_j + K_k}{T_k} \right\rceil \cdot min(bi_{ij},\ L_k + I_{kj}^{down}) \right\} \tag{20}$$

  where bi_{ij} is the maximum buffered interference over the contention domain cd_ij, defined as:

$$b_{ij} = buf(\Xi) \cdot linkl(\Xi) \cdot |c_{ij}| \tag{21}$$

  where $buf(\Xi)$ informs the FIFO buffer size and $linkl(\Xi)$ informs the amount of time taken by a router to transmit a flit over any of its links (assuming no contention).

- When computing downstream indirect interference caused by flows suffering from upstream interference, which is the set $G_{I_i}^{up_j}$, defined as the set of flows $\varphi_k \in G_i^I$ that interfere with the flows $\varphi_j \in G_i^D$ before φ_j interferes with φ_i, I_{ji}^{down} is calculated according to XIONG et al. (2017) as follows:

$$I_{ji}^{down} = \sum_{\varphi_k \in G_{I_i}^{down_j}} I_{kj} \tag{22}$$

Hence, the worst case response time of each flow (communication latency) is calculated by (19), applying the corresponding evaluation of I_{ji}^{down} according to each scenario, by employing (20) or (22). Thus, the worst case end-to-end response time of a task τ_i is given by $EER_i = R_i + S_i$, where R_i and S_i are, respectively, its worst-case computation response time and its worst case communication latency (INDRUSIAK, 2014), showed in Figure 4, which also presents a graphical visualization of other aforementioned variables. Its end-to-end schedulability is verified by checking whether $EERi \leq D_i$.

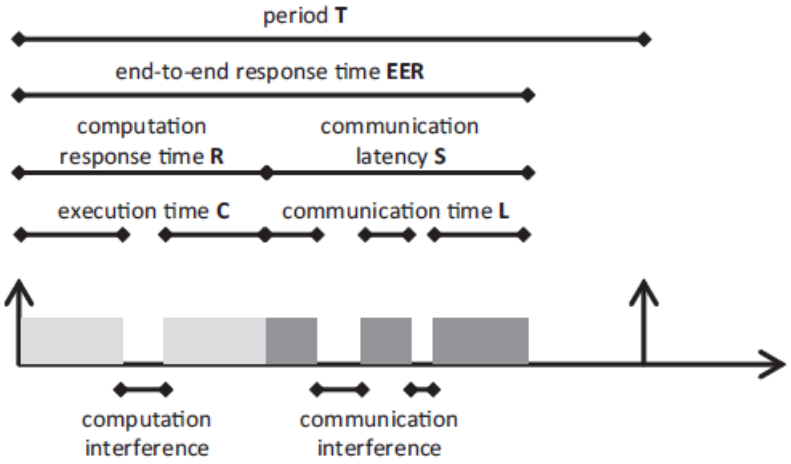

Figure 4 – End-to-end response time of a communicating task (INDRUSIAK, 2014).

2.3 Approximate Computing

The concept of *approximate computing* has gathered lot of research attention as an alternative to improve performance/energy efficiency by compromising the quality of the applications within tolerable ranges (VENKATARAMANI et al., 2015). The main idea is to exploit the application resilience to errors in order to reduce or simplify computation at software and/or hardware level (LIU; LOMBARDI; SHULTE, 2020). According to VENKATARAMANI et al. (2015), approximate computing lies on the notion that a unique, a golden result, simply does not exist, but a range of answers could be acceptable and even the best algorithms are not perfect, sometimes users are conditioned to

accept good-enough results (e.g., recognition problems). Approximate computing can be applied at different levels of abstraction layers, from devices to systems including hardware (devices, circuits, and architectures), software, algorithms, and programming languages (LIU; LOMBARDI; SHULTE, 2020).

At hardware level, approximate arithmetic circuits (adders, multipliers, dividers, etc.) have been widely studied based on the general principle of significance guided design (fewer resources are provided for less significant parts with lower complexity). Following this principle, both logic reduction/pruning and voltage scaling for probabilistic CMOS have been applied (LIU; LOMBARDI; SHULTE, 2020). Furthermore, regarding approximate architectural systems, e.g., some solutions can explore the usage of approximation in data storage and other solutions can even discard some calculation steps when targeting hardware accelerators design.

At software and algorithm levels, LIU; LOMBARDI; SHULTE (2020) also explain that one of the most effective approximate techniques is precision scaling, e.g. by reducing the bit-width word size, which leads to both computation and storage resources saving. At a higher level, a program can even skip some tasks and memory accesses in multicore architectures (GOIRI et al., 2015).

A common feature regarding approximate computing is the exploration of suitable applications, which can tolerate the approximation leading to a complexity/energy reduction without significant quality loss. For instance, some video applications are naturally error-resilient due to the limited perception of image details by the human visual system (HVS) (BAI et al., 2014). Thus, approximate computing can explore such a characteristic, being a suitable technique to improve the energy efficiency of complex applications like video coding systems. However, it raises a new challenge regarding the QoS of multimedia applications, since approximate computing introduces simplifications in the applications that may diminish QoS.

An interesting alternative to solve such an issue could be the exploration of the *nature of the application*, leading to *different levels of approximation and energy/performance efficiency*, with different levels of error resiliency. On the one hand, when applying approximate computing solutions, application steps presenting higher error resiliency contribute to smaller losses on QoS. On the other hand, application steps showing smaller tolerance to errors, could introduce remarkable degradation on QoS if approximate computing is applied. Thus, the challenge of controlling dynamically the use of approximate computing solutions, targeting the optimization of the trade-off between energy consumption and error resiliency, is of utmost importance since more aggressive approximation can be applied at suitable application steps improving energy efficiency (more error resilient), with minimum impact over QoS, and less aggressive or even no approximation can be applied at other application steps (less error resilient).

The application resiliency to approximation is usually data dependent (XU; MYTKOWICZ; KIM, 2015) and can be used to express where approximation is feasible and how it may impact the results, however, determining approximation-suitable data is not a trivial task, as well as determining the level of approximation that provides good trade-off between computation/energy and QoS. Therefore, the level of approximation (if any) to be employed depends on the evaluation of data features, which is not a trivial task, since several parameters present different impacts on QoS.

In order to illustrate this data dependency, Figure 5 shows an example of a picture processed by two different interpolation FIR (Finite Impulse Response) filters: Figure 5 (a) with 8-tap FIR filters and Figure 5 (b) with approximate 6-tap FIR filters.

We know from signal processing (SMITH, 1997) that an ideal interpolation in continuous time can be described as a *sinc* function to predict the values of unknown variables regarding a given distribution. When it comes to digital systems (discrete time), the *sinc* function can be represented as a FIR filter with M coefficients. Since the *sinc* function is infinite, it must be truncated with a constrained window at some point, targeting its practical implementation. The larger the window the higher the number of FIR coefficients (taps). On the one hand, a precise filter aim achieving a given frequency response, on the other hand, an approximate filter aim at being as equal as possible to the original one. Let analyze the images results.

In red it is detailed a heterogeneous region (hair detail) where it is possible to notice quality degradation, posed by the approximate filtering (observe the blocking artifacts) in comparison with the precise filter. Considering the homogeneous region (floor), detached in blue, quality difference between approximate and precise solutions is not noticeable. From this case study, one can observe that homogeneous regions are more error-resilient, i.e., a simplified filter performs well, and heterogeneous regions are less error-resilient, i.e., employing better filters is mandatory to avoid quality degradation.

When analyzing the impacts of the taps reduction, one can notice that it leads

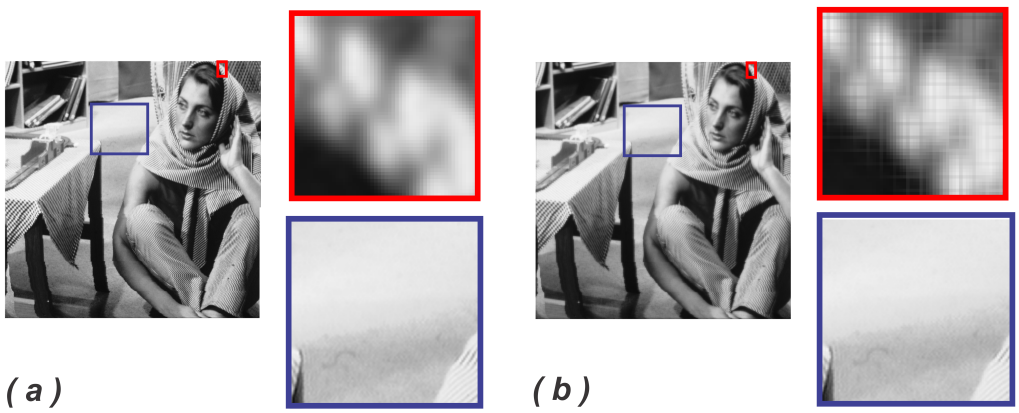

Figure 5 – Picture (public domain) encoded with: (a) original filters (b) approximate filters, detailing homogeneous (blue) and heterogeneous (red) regions.

to image quality losses. In order to investigate this reason, observe in Figure 6 the frequency response of the original filter and in Figure 7 the frequency response of the approximate filter. As expected, the FIR filter behaves like a low-pass filter, with no gain. However, when analyzing the approximate filter, it can be observed a ripple effect in the pass band, which leads to losses and consequent image quality degradation (see Figure 5 the introduced block artifacts).

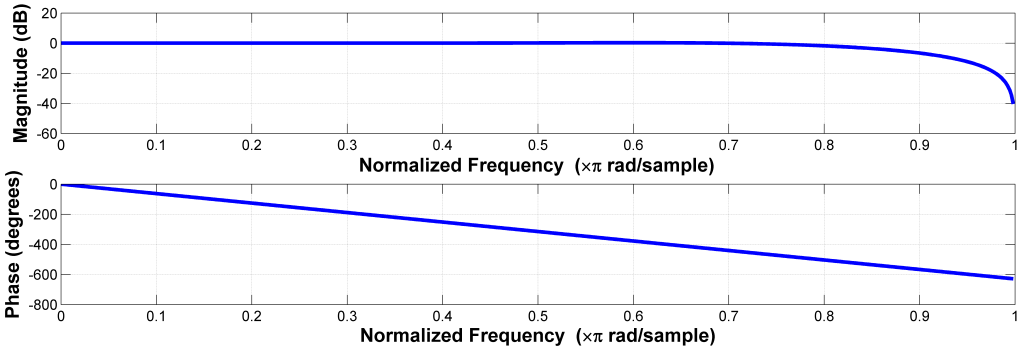

Figure 6 – Frequency response of a precise FIR filter.

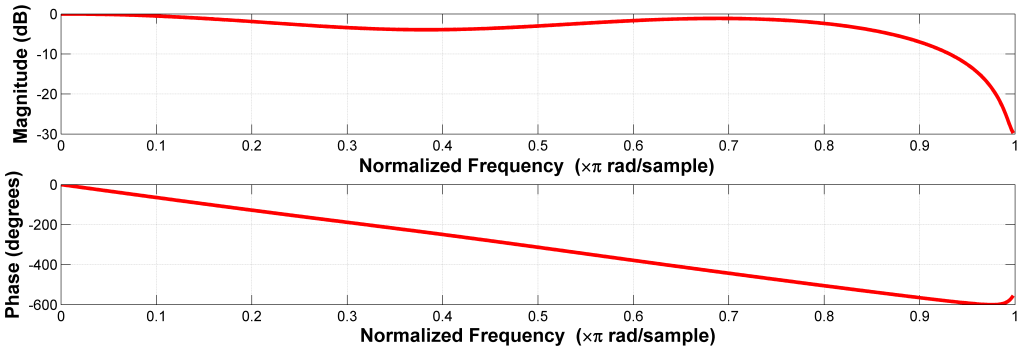

Figure 7 – Frequency response of an approximate FIR filter.

The verified quality degradation changes the objective image quality encoded with the approximate FIR. From the point of view of video coding, e.g., the insertion of approximate computing can lead to sub-optimal choices during the encoding process, which would lead to an overall loss on QoS. In the next section the basics about video coding are provided, allowing a better understanding on how approximate computing can be applied for this application.

2.4 Video Coding: High Efficiency Video Coding (HEVC)

2.4.1 Basic Concepts

A digital video is a sequence of digital static images (called *frames* or *pictures*), presented sequentially to the viewer at a given temporal rate to give a motion sensation, high enough to ensure a smooth transition-free visual perception. In general,

considering the human visual system (HVS), a frame rate to ensure a smooth motion perception is around 30 frames per second (fps) (RICHARDSON, 2003). Modern video applications have introduced more demanding requirements, increasing the need for higher frame rates, requiring up to 120 fps in order to provide real motion sensation for digital videos (SZE; BUDAGAVI; SULLIVAN, 2014).

These static images are digitally represented by a two-dimensional matrix of *pixels* (numerical representation of picture elements), with horizontal dimension *W* (width) and vertical dimension *H* (height), *WxH*, referred as the spatial video resolution.

Each pixel stores the color and luminosity information of its corresponding position within each frame. There are several color spaces defining such a numerical representation of the pixel properties, like widely used RGB (Red, Green and Blue) and YCbCr (Luminance, Blue Chrominance and Red Chrominance). Normally, video coding applications are based on the YCbCr color space, since the human visual system is much more sensible to luminance than chrominance (color) information, and YCbCr color system was conceived in order to take advantage of such a characteristic by allowing a sub-sampling of the chrominance information (RICHARDSON, 2002).

In fact, the sub-sampling of chrominance channels can be seen as the primary video compression, since the discarded information are imperceptible for the human visual system and it reduces the amount of data required to represent the video. The most common adopted sub-samplings configurations are the 4:2:2 and the 4:2:0, which have, respectively, two Cb and two Cr samples for each four Y samples, and only one Cb and one Cr sample for each four Y samples (RICHARDSON, 2003). The HEVC standard supports the 4:4:4 configuration, when no sub-sampling is applied, and the mentioned sub-sampling configurations (4:2:2 and 4:2:0). Since 4:2:0 configuration is widely employed researchers, being the most used sub-sampling configuration, we have considered 4:2:0 along all experiments presented in this thesis.

Video coding/compression seeks to reduce the amount of data considered redundant in the representation of the image or video information. Data that do not contribute with new visual information, relevant to the representation of the image, is considered redundant. There are basically three types of redundancy to be explored: spatial redundancy, temporal redundancy and entropic redundancy (GHANBARI, 2003). Depending on the application, video coding standards explore these redundancies in order to reach the highest compression ratios keeping reasonable video image quality.

2.4.2 General Aspects

The High Efficiency Video Coding (HEVC) (SULLIVAN et al., 2012; ISO/IEC-JCT1/SC29/WG11, 2013; SZE; BUDAGAVI; SULLIVAN, 2014) standard has been released in 2013 by the Joint Collaborative Team on Video Coding (JCT-VC), a joint effort project of the Video Coding Experts Group (VCEG), from the ITU-T (Interna-

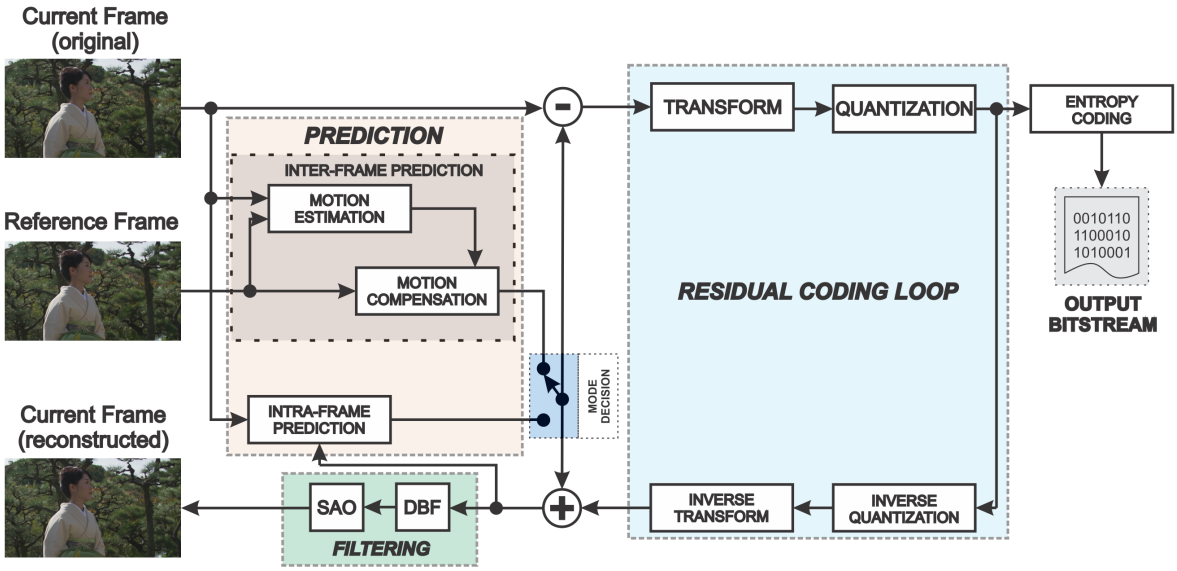

Figure 8 – HEVC block diagram.

tional Telecommunication Union - Telecommunication Standardization Sector), and the Moving Picture Experts Group (MPEG), from the ISO/IEC (International Organisation for Standardisation and the International Electrotechnical Commission), in order to replace older standards, as an alternative to improve the overall delivered compression ratio and support HD (High Definition) and UHD (Ultra-High Definition) resolutions. Indeed, HEVC is capable of reaching up to 40% higher compression efficiency when compared with the previous H.264/AVC (Advanced Video Coding) encoder (VANNE et al., 2012), delivering the same objective video quality levels (GROIS et al., 2013). Such requirements make current video coding standards even more complex, for instance HEVC is up to 18 times more complex than H.264/AVC in average (GROIS et al., 2013) when considering the same video quality, which leads to higher energy consumption/power dissipation regardless the system where the coding process is being implemented. Such an increase on energy consumption is due to the new features of the HEVC, like the use of modern tools and techniques, introduced in order to address those demanding requirements.

HEVC employs the widely used hybrid coding compression model, based on the encoding of residues, composed of the following signal and data processing elements: (i) intra and inter-frame prediction - exploiting spatial and temporal redundancy, respectively; (ii) de-correlating transform; (iii) quantization; (iv) entropy coding; and (v) in loop filtering, composed by the filters SAO (Sample Adaptive Offset) and DBF (Deblocking Filter). Figure 8 shows the operational block diagram of HEVC standard, where we can observe the hybrid coding model.

Firstly, the frames of the input video are split into equally-sized block-shaped regions, called Coding Tree Units (CTUs). The first frame of the input video is always

encoded using only intra-frame prediction, since there are no frames previously encoded to be used as reference in inter-frame prediction. The remaining frames may use both intra and inter-frame prediction.

The intra-frame prediction uses spatial neighboring samples in order to predict the block being coded. During the inter-frame prediction, each block from previously encoded frames is predicted by the Motion Estimation (ME) and Motion Compensation (MC) steps. ME seeks to find the most similar block in the reference frame when compared with the current block being encoded. Motion Vectors (MVs) obtained from the ME process determine the relative location of the best prediction block within the reference frame and are used in the MC process to reconstruct the block. The mode decision control selects the best prediction mode (inter or intra).

The difference between the current block and the predicted block generates the so called residues, which are used as input of the Residual Coding Loop (RCL). During the RCL, direct transform is applied over the residues, converting the samples values from the spatial to the frequency domain, in order to de-correlate the residue and concentrate the signal energy in a few low-frequency coefficients. This process makes the information unnecessary to human visual perception more evident, which may be later attenuated. HEVC employs at transform step the integer transforms Direct Sine Transform (DST) and Direct Cosine Transform (DCT) (SZE; BUDAGAVI; SULLIVAN, 2014).

After transform stage, quantization step is applied over transformed coefficients, which are attenuated, with stronger attenuation over high-frequency coefficients that are not perceptually relevant (considering the human visual system) (GONZALEZ; WOODS, 2010). This attenuation normally generates sparse matrices of coefficients (with several zero values). It is worth to notice that the quantization inserts unrecoverable losses in the residual data. The strength of the quantization, i.e., how aggressive the quantization must be, is controlled by the Quantization Parameter (QP). Increases on QP values lead to higher quantization strength, which in turn leads to more losses during the encoding process. Therefore, higher losses are generally associated to higher compression ratios. For example, the QP is frequently used to adapt the required bandwidth to transmit the output bitstream over an unstable communication channel (VIZZOTTO et al., 2012).

Finally, at the end of the encoding flow, the entropy coding applies data compression algorithms over all generated data (residues and other video control information). The entropy coding step aims to reduce the generated data representation redundancy (SZE; BUDAGAVI; SULLIVAN, 2014). HEVC applies the Context-Adaptive Binary Arithmetic Coding (CABAC) algorithm at entropy coding step (SZE; BUDAGAVI; SULLIVAN, 2014). The output is commonly referred as *output bitstream*, which is sent for properly transmission or storage.

Since the quantization stage generates irreversible losses of information, the coded frame will be different from the original frame after reconstruction at the decoder side (RICHARDSON, 2003). For this reason, the HEVC encoder replicates the decoder processing loop, guaranteeing that both the decoder and the encoder use the same reference samples for intra/inter-frame prediction, since the encoder discards the original frame after being processed and stores the reconstructed frame. Therefore, despite feeding the entropy coding, the quantized residue is fed to the inverse quantization and inverse transform, allowing the reconstruction of the residual information, which, added to the predicted samples, generates the reconstructed samples. As can be observed in Figure 8, this result is delivered to the Deblocking Filter (DBF) and next to the Sample Adaptive Offset (SAO), in order to smooth out artifacts caused by block-wise processing and quantization. In this way, the encoded frame becomes the reference for both encoding and decoding.

2.4.2.1 Video Partitioning Structures

According to MCCNANN et al. (2014), in contrast to schemes presented by previous video encoding standards like H.264/AVC, HEVC employs a video compression scheme based on the partitioning of encoded blocks into a highly flexible hierarchy, which allows the use of large blocks and multiple partitioning levels for prediction and transform blocks, as well as new coding tools. All of these improvements make HEVC a very efficient video encoder.

This video compression flexible hierarchy includes the partitioning of a frame into square-shaped blocks called Coding Tree Units (CTUs) and three other block concepts: Coding Unit (CU), Prediction Unit (PU) and Transform Unit (TU) (MCCNANN et al., 2014). This division into different concepts allows each one to be optimized in the best possible way.

Each CTU consists of a block of luminance samples together with two other chrominance blocks. The sizes of the chrominance blocks will depend on the adopted color sub-sampling (RICHARDSON, 2003). The maximum size of a CTU (and most usual) is 64x64 samples of luminance, which also corresponds to the maximum size of a CU, but a CTU can also have sizes 32x32, 16x16, or 8x8.

The CTU can be composed of one or more Coding Units (CUs), which are used in both types of prediction and always assume square shapes with 2Nx2N size, where N can be 4, 8, 16, or 32. Therefore, the value range of a CU is at least 8x8 samples of luminance up to the maximum CTU size. The recursive division of CTU into CUs allows a variation from small to large block sizes, composing a quadratic tree (quadtree) formed by blocks of CU (SZE; BUDAGAVI; SULLIVAN, 2014).

Figure 9 shows the illustration of a frame divide into CTUs and a division of a CTU into CUs. It can be seen that in the detailed CTU, the first depth presents the initial

CTU 64x64. This is the first level of the quadtree, however, each 64x64 block can still be subdivided into four 32x32 blocks, making up the second quadtree level. Therefore, recursively, each block can continue being subdivided until the fourth depth, in which the CUs will have size 8x8. Note that according to encoding control decisions, the CTUs assume different levels within the frame. Normally more heterogeneous regions tend to split the CTU in more levels, and more homogeneous regions tend to make few (or none) divisions.

During the prediction step (intra or inter), the CUs are divided into base blocks called Prediction Units (PUs). Differently from CUs, PUs allow non-square shapes for the blocks. Hence, since the blocks can be partitioned into different shapes, a more precise picture detailing can be obtained. In Figure 10 it is presented the eight different modes for partitioning a CU into PUs supported by the HEVC: four symmetric and four asymmetric partitions. The values of N can be 4, 8, 16, or 32. Division NxN is only allowed during intra-frame prediction (if N is equal to 4). On the one hand, during the intra-frame prediction, only NxN and 2Nx2N partitions are enabled, on the other hand, during the inter-frames prediction, both symmetric partitions (2NxN, and Nx2N) and asymmetric partitions (2NxnU, 2NxnD, nLx2N and nRx2N) are allowed. Note that asymmetric partitions are available only if N is bigger than four.

The TU is the base block unit used to the processes of Transform and Quantization. TUs are always square-shaped blocks of size NxN, where N can have the same values

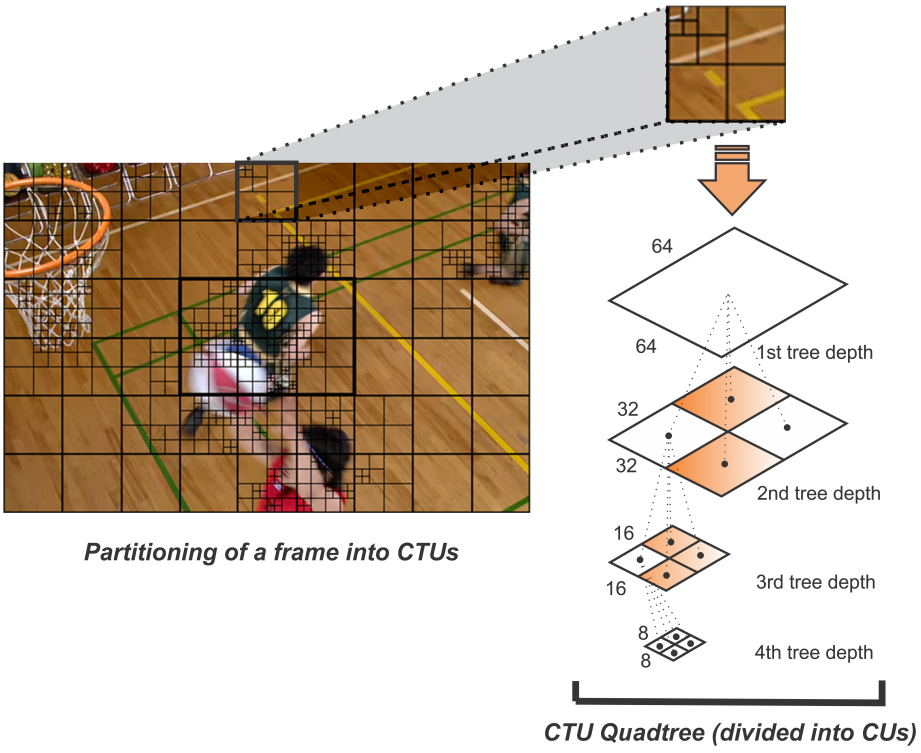

Figure 9 – Division of a frame into CTUs and a CTU in CUs (adapted from ZHOU; ZHOU; CHEN (2013)).

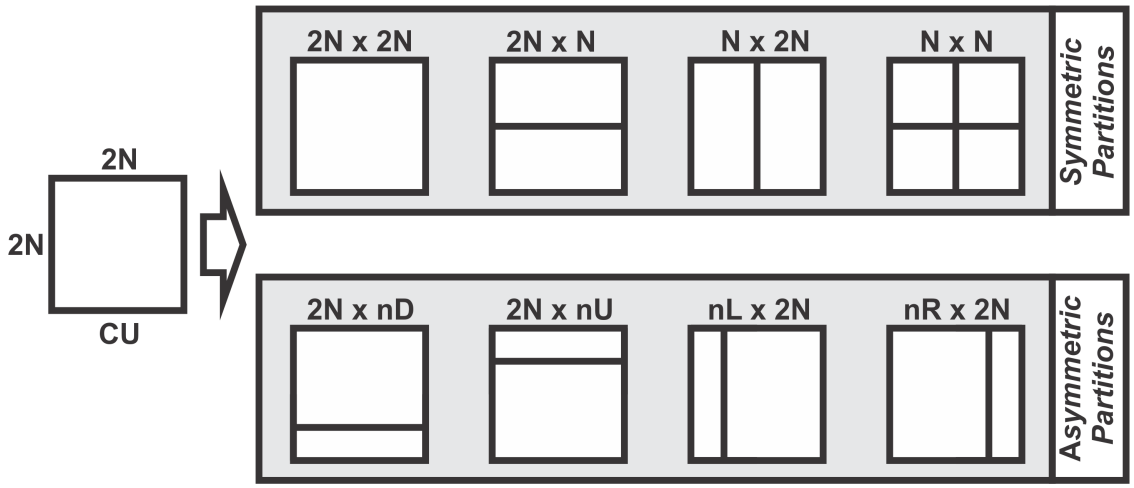

Figure 10 – Division of a CU into all possible PUs allowed by HEVC (adapted from MCCNANN et al. (2014)).

applied to the PUs (4, 8, 16, and 32). Each CU can have one or more TUs so that the TUs can also be disposed as a quaternary tree structure having the CU as root, so called Residual Quadtree (RQT).

In spite of partitioning frames into blocks, HEVC also allows the division of the video sequence into other hierarchy levels. For instance, in the HEVC, a video sequence is divided into a fixed-size Group of Pictures (GoP), which in turn is a set of frames. According to the previous discussion, the frames are divided into CTUs, which in turn are partitioned into CUs. At proper steps, the CUs are further partitioned into PUs and TUs, performing the pursuit to reach the best coding efficiency. It is worth to notice that a GoP is able to use as references only frames within the same GoP. In Figure 11 we show an example of a video divided into GoPs containing four frames each. The concepts of frames I and B will be further explained.

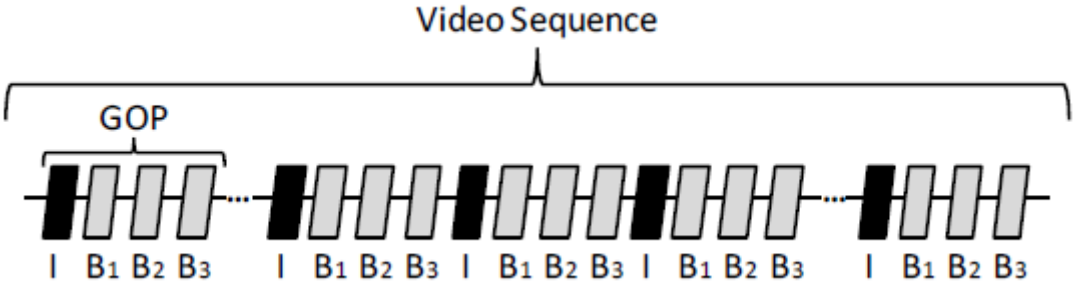

Figure 11 – Division of a frame into GOPs (CORRÊA, 2014).

2.4.2.2 Common Test Conditions and Temporal Configurations

The specification of HEVC standard (ISO/IEC-JCT1/SC29/WG11, 2013) describes the adopted syntax and decoding procedures. The researchers are free to implement

any changes at the encoder side, since the generated *bitstream* remain compliant with the syntax and decoding rules established by the standard.

In order to guide tests using the HEVC, as a way to allow a fair comparison among research works, the JCT-VC stipulated a document that establishes the so called Common Test Conditions (CTC) (BOSSEN, 2013), when dealing with the HEVC reference software, the HEVC Test Model (HM) (BOYCE, 2014). The CTCs document defines six classes of video sequences (from A to F), presenting different resolutions and motion/texture patterns, which should be used when performing tests using the HEVC standard. The CTC also states that simulations must use four different QP levels: 22, 27, 32 and 37.

In addition, CTC document also establishes the structures of temporal prediction, which are divided into three types: *All Intra* (AI), *Low Delay* (LD) and *Random Access* (RA), which establish the way in which frames are distributed and referenced in a GOP.

In AI configuration all pictures in the video sequence are encoded as Instantaneous Decoding Refresh (IDR) pictures, which are images containing only intra-frame prediction. When encoding video according to this configuration, inter-frames prediction is not allowed.

There are two types of LD configuration, type P (unidirectional prediction) or type B (bi-directional prediction). In both conditions, only the first frame of the sequence is encoded as an intra-frame. In the *Low Delay B* configuration, all successive frames are encoded as GPB (Generalized P and B Picture) frames. The GPB frames are B pictures that only allow using reference frames that appear before the current frame in display order. B frames can use reference frames from two different lists (composed of the previously stored reference frames), using two reference frames together to generate the prediction. P frames use only reference frames from one list. In the *Low Delay P* configuration, all inter-frames are encoded as type P (MCCNANN et al., 2014). Figure 12 presents the graphical representation of a *Low Delay* configuration. The indices above each frame indicate the encoding order. In the *Low Delay* setting, the coding order is in the same order used for displaying.

When considering RA configuration, a hierarchy structure composed of bi-predicted frames (B frames) is used in the encoding process. One IDR frame is always periodically inserted in order to refresh the reference lists. The frames located between two IDR frames, considering the viewing order, are encoded as B frames. GPB frames are used in the lowest level of the temporal layer. The second and the third temporal layers are composed by referenced B frames. Finally, the highest level of the temporal layer contains only non-referenced B frames (i.e., they are not used as reference for any other frame). The QP value related to each inter-frame is obtained from an offset (which depends on the temporal layer), summed to the QP of the intra-frame. It is worth to mention that the viewing order in Random Access configuration is not the

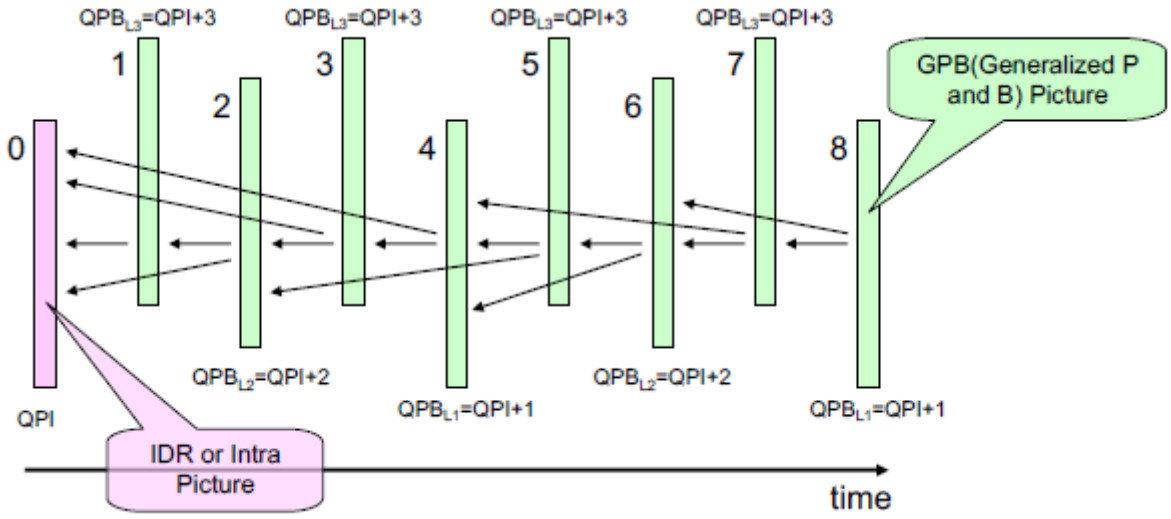

Figure 12 – Graphical presentation of LD configuration (MCCNANN et al., 2014).

same used in the encoding process. In Figure 13 is presented a graphical representation of RA configuration, where the numbers associated to each frame represent the encoding order.

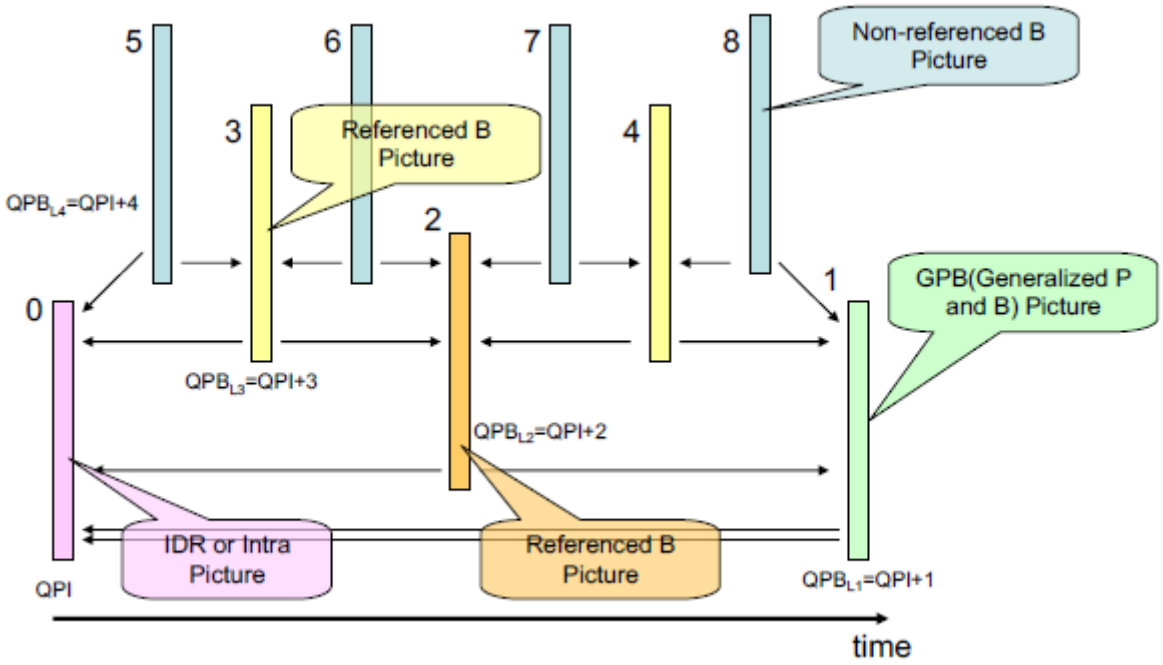

Figure 13 – Graphical presentation of RA configuration (MCCNANN et al., 2014).

2.4.2.3 Rate-Distortion Cost

Based on what was explained so far, one can notice that there are several possible combinations to encode a single CTU, like different prediction modes (intra or inter), partition sizes, transforms sizes, QPs, and so on. In order to determine the

best encoding configurations, a decision called Rate-Distortion Optimization (RDO) is implemented by the video encoders.

When analyzing the rate-distortion (RD) cost of a given configuration being tested, the *rate* metric is related to the size (number of bits of the generated output *bitstream*, and the *distortion* represents the objective quality provided by such a configuration. Equation (23) presents the RD cost mathematical definition, where λ is the lagrangian parameter that weights the trade-off between distortion (D) and rate (R) (λ depends on the adopted QP).

$$RD_{cost} \ = \ D \ + \ \lambda \cdot R \tag{23}$$

The distortion is obtained through the application of any of the distortion metrics (discussed in the next section) over the reconstructed blocks, for each particular configuration. The rate is obtained after the transform, quantization and entropy coding steps. During prediction, e.g., since the RD cost of each CU must be calculated during the CTU quadtree decision, the transforms and quantization steps need to be called many times just to decide the best prediction mode. In a similar way, during the RQT decision, the entropy coding is executed several times in order to decide the best TU size.

Considering that, one of the most prominent optimization problems regarding video compression algorithms is the minimization of the RD cost, which leads to the maximization of the coding efficiency, and, for every decision made during the encoding process, a repercussion (positive or negative) in the final coding efficiency will be verified. Therefore, optimal coding efficiency is achieved when every possible decision during the encoding process is evaluated considering the RD cost calculation, which leads to a huge computational effort spent by video encoders.

2.4.2.4 Distortion Metrics

During the video coding process any standard introduces distortion in the compressed videos. In order to measure the objective video quality, different metrics can be used. Normally, the objective metrics are based on direct comparisons between pixels from different pictures or blocks, like the original and the reconstructed image.

The most used and known objective distortion metric is the Peak Signal-to-Noise Ratio (PSNR) (RICHARDSON, 2003), which can be applied at different levels: from a block or frame to a entire video. The PSNR is represented in decibels (dB), which equation is defined in (24).

$$PSNR_{dB} \ = \ 20 \ \cdot log_{10} \left(\frac{MAX}{\sqrt{MSE}} \right) \tag{24}$$

MAX is the maximum value that a sample can assume ($2^n - 1$ in HEVC, where n is

the number of bits per sample) and MSE is the Mean-Squared Error for the image or block, a similarity criterion calculated according to (25).

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (R_{i,j} - O_{i,j})^2 \tag{25}$$

In (25), m and n are the picture dimensions, O and R represent the original and reconstructed samples (luminance or chrominance), respectively. MSE metric expresses the difference (distortion) between the samples from two frames or blocks in an objective way, thus we can also consider MSE itself as a distortion metric. When looking for low-complexity distortion metrics, we can detach the Sum of Absolute Differences (SAD) (KHUN, 1999), widely used in video encoders, especially in hardware implementations of video coding modules. SAD is computed as shown in (26) and its variables are the same appearing in (25).

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |R_{i,j} - O_{i,j}| \tag{26}$$

2.5 Fractional Motion Estimation

During the inter-prediction, at the motion estimation (ME) step, temporal redundancy between two or more frames is exploited. As aforementioned, the frame is divided into smaller blocks (PUs).

For each block being coded, there is a search for the best matching block within the reference frame, constrained at a search window (SW). This search considers a similarity criterion, usually the Sum of Absolute Differences (SAD). The ME block diagram is presented in Figure 14. ME is divided into two steps: integer ME (IME), which is presented in Figure 15, and fractional ME (FME), our main case study, which is presented in Figure 16.

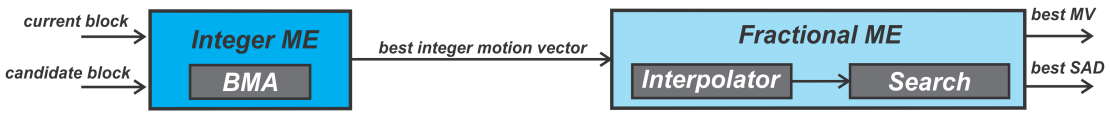

Figure 14 – Motion estimation steps.

To find the best integer matching, a block matching algorithm (BMA) needs to be applied. Many types of BMA may be used, the HEVC reference implementation employs the widely used Test Zonal Search (TZS) algorithm, which is a multi-step algorithm that adapts to data properties and performs the search up to 23x faster at equivalent quality when compared to the exhaustive Full Search (FS) algorithm (PURNACHAND; ALVES; NAVARRO, 2012). During this step, the IME compares the current block (we

want to predict) with candidate blocks within the reference frames in order to find the most similar.

TZS is implemented considering four stages: **prediction**, which moves the search window to the region with the highest probability of matching (around the collocated block in Figure 15 for simplicity); **first search**, performed around the central point, assuming an expanding diamond shape (two expansion levels are shown on Figure 15); **raster**, performed only when the previous step fails (a reasonable block matching is not found); and **refinement**, which improves the result found in the previous stages. When the best integer matching is found, a motion vector (MV) is generated pointing to the selected block. This information is then sent to the next step, the FME.

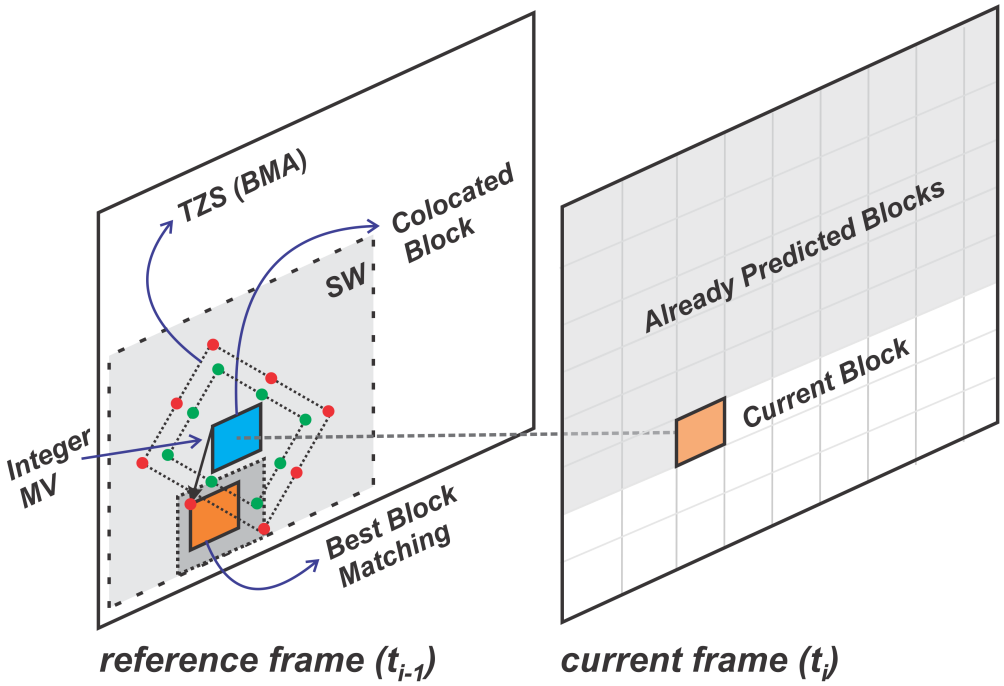

Figure 15 – IME details.

In order to improve the coding efficiency, FME is applied over the integer block found at IME, allowing the matching in sub-pixel positions, with 1/2 or 1/4-pixel precision. The FME consists of two steps: the interpolation and the search steps, as shown in Figure 14. During interpolation, the calculation of the fractional samples is performed using finite impulse response (FIR) filters with 7 or 8 taps, whose coefficient values are presented in Table 1. These filters, following a nomenclature stated in AFONSO et al. (2013), are called UP, MIDDLE or DOWN, according to the samples they calculate. Table 1 also shows which sample each filter calculates.

Figure 16 presents the example of a detailed FME behavior when evaluating an 8x8 block. In Figure 16 (a) we show the best 8 x 8 candidate selected at IME, with its sample borders necessary for the interpolation calculation (compounding a 15 x 15 reference matrix in the case of 8 x 8 blocks). The FME tries to refine the motion, by

Table 1 – HEVC FME Interpolation Filter Coefficients

Filter	***Coefficients***	***Evaluated Samples***
UP	{-1, 4, -10, 58, 17, -5, 1}/64	*a, d, e, f, and g*
MIDDLE	{-1, 4, -11, 40, 40, -11, 4, -1}/64	*b, h, i, j, and k*
DOWN	{1, -5, 17, 58, -10, 4, -1}/64	*c, n, p, q, and r*

also searching in fractional positions. These positions do not exist in the image grid, but their values can be estimated by interpolation, describing the sample values if the motion was smaller than an integer sample precision.

Observe in Figure 16 (b) the detailing of the 8x8 integer block (depicted as orange boxes). The interpolation is divided into two parts: (i) the *horizontal filtering*, which performs the calculation of samples *a*, *b*, and *c* (depicted as yellow boxes), using neighboring integer samples (A) in a row of the reference matrix, and (ii) the *vertical filtering*, which performs the calculation of samples *d*, *h*, and *n* (depicted as red boxes), using neighboring integer samples in a column, and the calculation of the remaining samples (depicted as blue boxes), using the previously pre-calculated fractional samples (*a*, *b*, and *c*). In Figure 16 (c) we present the behavior of the horizontal filtering process. Each filter tap is multiplied by a given sample. The sum of these multiplications is further divided per 64, in order to guarantee no gain after the filtering. For each 8 x 8 block, fifteen fractional blocks also sizing 8 x 8 are generated, depicted in Figure 16 (d), which are composed by samples *a*, *b*, *c*, *d*, and so on. Generalizing, fifteen n x m blocks (considering that the block being coded has n x m dimension) containing the calculated samples are generated for each integer n x m block. During search step, these blocks are evaluated and compared, finally, the best fractional block is appointed (in terms of applied similarity criterion, e.g. SAD).

As could be seen, the Fractional Motion Estimation is a very computation/power demanding step, being responsible for up to 60% of total HEVC encoding effort (GRELLERT; BAMPI; ZATT, 2016). For instance, the encoding of a single frame of an UHD 4K video generates more than 120 million fractional samples (1/2 and 1/4 pixel) only for the 8x8 blocks. When we analyze other block sizes, considering only square-shaped blocks (16 x 16, 32 x 32, and 64 x 64), this number grows exponentially, reaching more than 10 billion samples to be calculated regarding a single 4K frame. This huge number led the researchers to think in strategies in order to reduce the computational effort while simultaneously achieve the required performance.

On the one hand, approximate computing emerges as an interesting solution in order to reduce the computational effort/power demands of FME. The simplification of some steps at FME may lead to sub-optimal choices but can introduce huge power savings. On the other hand, the entire process does not present data dependency

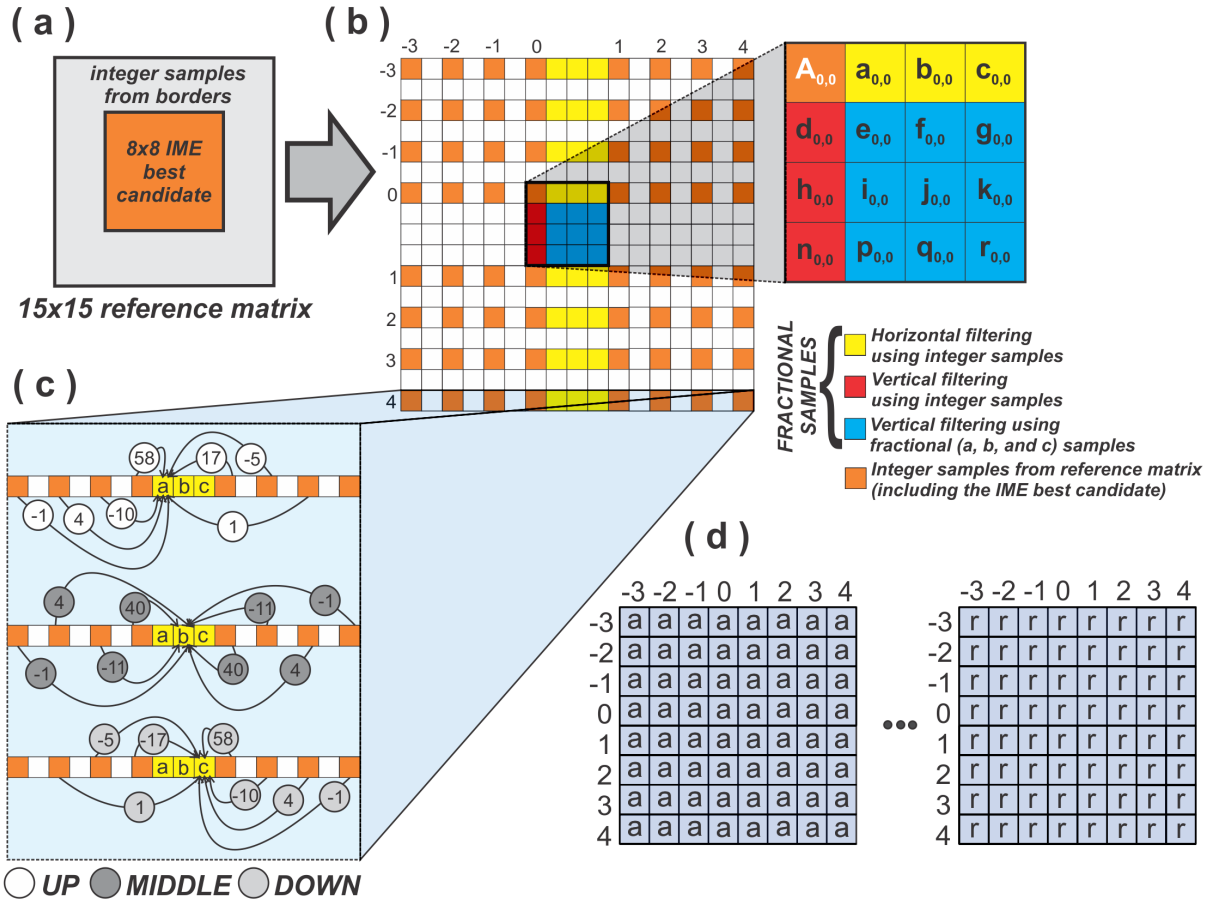

Figure 16 – FME detailing for an 8x8 block: (a) integer best candidate block within the reference matrix, (b) fractional samples from horizontal and vertical filtering details, (c) filtering behavior from UP, MIDDLE and DOWN, and (d) fifteen fractional blocks regarding one integer block.

between spatial neighbor blocks, i.e., all blocks evaluated by the FME can be processed in parallel, bringing an interesting opportunity for parallelism exploration.

2.6 Machine Learning: Decision Trees

The concept of *Knowledge Discovery from Data* (KDD) (FAYYAD et al., 1996) has gained lot of research attention recently, being currently applied to several knowledge areas, such as medicine, market management, biology, and image processing. KDD systems aim to extract information from both structured and unstructured sources, by using strategies based on machine learning algorithms.

Machine learning (ML) is a branch of study in artificial intelligence that is concerned with designing algorithms capable of obtaining knowledge from observations (RUSSELL; NORVIG, 2002). Regarding computational problems, such knowledge is represented as a model that estimates the output of a task based on some indicators. This definition is intentionally broad, because it shows that machine learning can be

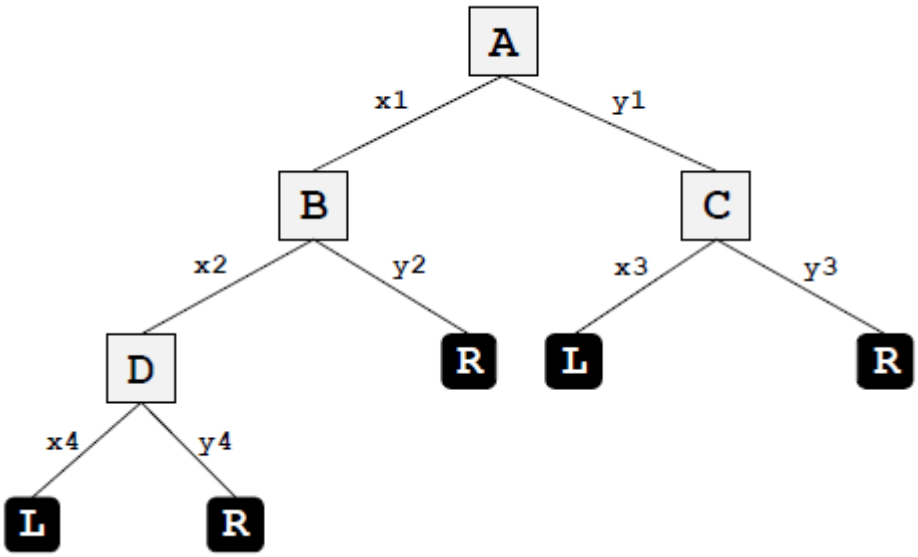

Figure 17 – Example of a decision tree (CORRÊA, 2014).

applied to many cases, as long as errors are allowed to some degree (GRELLERT, 2018). Techniques based on ML are used to determine the value of dependent variables by looking at the value of some attributes in the data set, identifying regularities and building generalisation rules that can be expressed as models (CORRÊA, 2014).

In most cases, the training process is performed offline using the entire training set. If the model generalizes well, a good performance is expected when it is tested against an evaluation set, which must be different from the training one.

There are several methods of machine learning available in the literature, varying according to their efficiency, complexity and applicability (CORRÊA, 2014). Decision trees (QUINLAN, 2014) are widely used machine learning algorithms, where a dependent variable can assume one among a finite number of outcomes.

According to CORRÊA (2014), when building decision trees, observations on a set of training data are mapped into arcs and nodes, as demonstrated in Figure 17. Inner nodes (A, B, C, and D) represent the tested variables (attributes), while the arcs are the possible values that the attributes can assume. In a binary classification tree the attributes can assume two results (e.g. x2 and y2 for attribute B in the given example). The leaves of decision trees are the values that the class attribute can assume and represent the possible outcomes of the whole decision process. In the example given in Figure 17, the possible outcomes are L and R.

Decision trees usually achieve high prediction accuracy after trained for a set of problems, being very easy for human beings understanding, thus being simple to implement. Normally, there are many efficient algorithms to build them from training data, like the widely used C4.5 (QUINLAN, 2014). Decision trees are very practical and can

handle both categorical and numerical values, executing predictions very fast. The prediction process adds negligible extra computational complexity to the application (CORRÊA, 2014).

2.7 QoS Metrics for Video Coding

There are several metrics in the literature targeting the coding efficiency assessment of digital videos. The coding efficiency can be used to measure the QoS of video coding applications. These metrics can employ two approaches for analysis, considering a subjective or objective assessment. Metrics based on subjective assessment take into account the perception of the spectators, when watching both the original and a modified video. Objective approaches use mathematical models to compare the original and the modified video.

For objective analysis of video QoS, metrics based on bit-rate and PSNR are widely used. However, the trade-off between compression and image quality cannot be accurately verified if a individual analysis of these metrics is performed. In order to address such an issue, the BD bit-rate (BD-BR), or simply BD-Rate, and the BD Peak Signal-to-Noise Ratio (BD-PSNR) metrics are used, which are based on the Bjontegaard Difference (BD) (BJONTEGAARD, 2001). The BD-BR can be interpreted as the percentage variation of the bit-rate between a *reference* configuration and a *test* configuration, considering videos with the same objective quality after encoding. Thus, a positive BD-BR means loss of *test* encoding performance in relation to *reference*, a negative BD-BR indicates gain, i.e., an encoding that resulted in a lower bit-rate video, but with the same objective quality. The BD-PSNR can be interpreted as the variation in decibels of PSNR between the *reference* and the *test* condition, considering videos that have the same bit-rate. Thus, a negative BD-PSNR value indicates loss of quality while a positive value represents an improvement in video quality, both of which having the same bit-rate.

To obtain these metrics, configurations *reference* and *test* are coded for four different Quantization Parameters (QP), producing eight pairs (PSNR, bit-rate). These pairs are used to interpolate two curves through a third order interpolation function, generating two Rate-Distortion (RD) curves (BJONTEGAARD, 2001), presented in Figure 18. The area between the curves is integrated using the Y axis as a reference for the BD-BR (Figure 18 (a)) and the X axis as a reference for the BD-PSNR (Figure 18 (b)). The equation for calculating BD-BR and BD-PSNR are given in (27) and (28), respectively. REF and TEST in (27) and (28) represent the PSNR or bit-rate values for the curves, regarding the *reference* and *test* encoding configurations, respectively. Variables A and B, used as integration limits, represent the second minimum and the second maximum PSNR values considering the eight pairs (bit-rate, PSNR), for the BD-BR calculation,

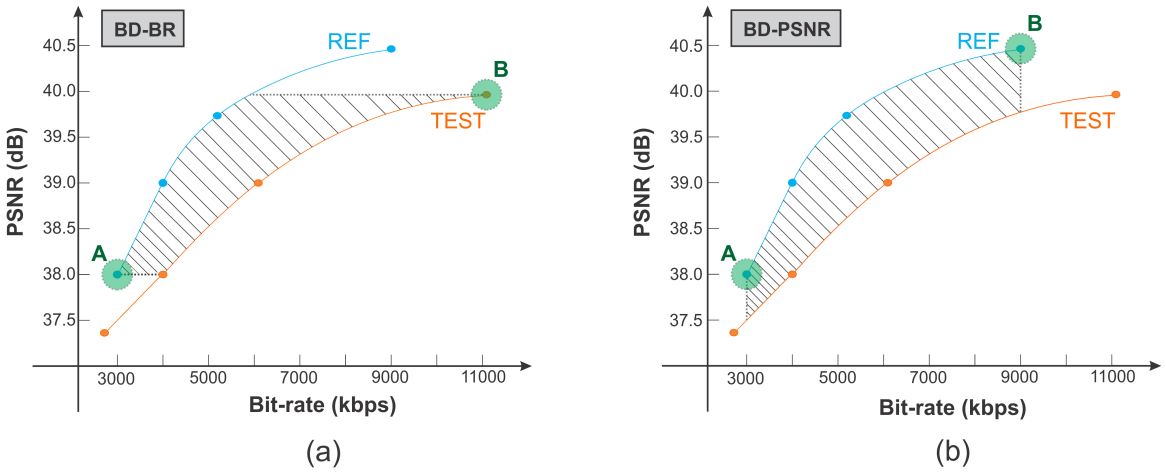

Figure 18 – RD curves employed on (a) BD-BR and (b) BD-PSNR calculation.

and they also represent the second minimum and the second maximum bit-rate values considering the eight pairs (bit-rate, PSNR), for the BD-PSNR calculation.

$$BR_{BR} = \frac{\int_A^B (REF_{PSNR}(y) - TEST_{PSNR}(y))\, dy}{B - A} \tag{27}$$

$$BR_{PSNR} = \frac{\int_A^B (REF_{bitrate}(x) - TEST_{bitrate}(x))\, dx}{B - A} \tag{28}$$

2.8 Summary

In this chapter we have revisited several branches of the literature, extremely important for a better comprehension of this work. Basic fundamentals regarding communicating NoC and real-time schedulability analysis were provided. A background regarding video coding concepts, ranging from basic aspects to specific features of HEVC encoder was also provided. We revisited the adopted evaluation metrics and focused on more detailed explanation about some HEVC encoder steps, exploited by this work, like the inter-prediction module. Finally, basic comments about machine learning were provided, detailing the widely used decision trees, also exploited in this work.

3 RELATED WORKS

In this thesis, several computer science research branches were revisited to support our main contributions and innovations. In order to visualize the insertion of this work in current state-of-the-art, we have analyzed several works in the literature. This chapter aims to organize this comparison, showing the main challenges that are still unsolved by current related works. In Section 3.1, we analyze many works proposing approximate computing solutions, from general purpose complex systems to video coding applications. Section 3.2 brings the discussion about NoC-based solutions, exploring the seek for energy efficiency in homogeneous and heterogeneous NoCs. In Section 3.3, it is contemplated the study about works developing hardware acceleration for video coding, specifically for our target case study, the FME. A discussion about precise and approximate hardware is also provided. Finally, in Section 3.4, we summarize related works contributions, listing and discussing the remaining research challenges.

3.1 Approximate Computing for Complex Systems

As aforementioned in this thesis, approximate computing has being applied by researchers to simplify the processes, aiming a reduction on energy, computational effort, external memory accesses, and so on, trying to insert the smallest errors on the application QoS. Due to its versatility, approximate computing can be applied over several applications, from general complex systems to video processing (our main focus), allowing the energy consumption reduction while maintaining the overall QoS of the application at tolerable ranges.

3.1.1 General Purpose Complex Systems

There are a wide range of works applying approximate computing for general complex systems available in the literature. BOYAPATI et al. (2017) proposes an approximate data framework for communicating network-on-chip architectures. The authors propose APPROX-NoC, a data approximation framework for NoCs conceived in order

to alleviate the impact of heavy data communication stress on NoCs, by leveraging the error tolerance of applications. They claim a reduction on the transmission of approximately similar data in the NoC, by delivering approximated versions of precise data to improve the data locality for higher compression rate. The proposed framework uses an approximation engine to adapt the error control logic, approximating a given data block to the nearest compressible reference data pattern. By applying this solution, APPROX-NoC is capable of reaching expressive reduction on communicating latency under a small penalty on application error.

YIN et al. (2020) propose the design of a dynamic range of approximate logarithmic multipliers (LM) for machine learning applications. In their work, the approximation is realized by truncating the operation, compensating the adder results and using the inherent approximate characteristics of the LM. The proposed circuit level design is able to deliver expressive energy savings, when compared with precise solutions, introducing small error on application. GUESMI et al. (2020) introduce a defensive approximation strategy, enhancing the security of Convolutional Neural Networks (CNNs) through approximate computing usage. They implemented an approximate CNN hardware accelerator, built upon an aggressively approximate floating point multiplier, which injects data-dependent noise within the convolution calculation. By following this strategy, the authors claim that their approximate computing model is capable to maintain the same level of classification accuracy, without retraining requirements and reducing resource utilization and energy consumption of the CNN.

Approximate computing is also explored by other research areas, such as IoT. In GHOSH; RAHA; MUKHERJEE (2020), it is proposed the design of a low power hardware prototype for algorithm implementation, built on a real-time microcontroller-based IoT platform that operates as an end-to-end Wireless Body Sensor Node (WBSN) system in real time, which is an IoT-based health system for monitoring patients outside the hospital environment. Their experimental results have shown significant system-level energy improvement at negligible impact on signal quality.

It can be observed that many works in the literature apply approximate computing focusing on suitable applications (the ones presenting resilience to errors). Regarding such a scenario, the video coding application emerges as a potential candidate for approximate computing. Indeed, there are many works in the literature exploring these aspects, as detailed in the following.

3.1.2 Video Coding

When it comes to approximate computing for video coding, different strategies can be employed exploring the approximation at different levels, such as the ones based on hardware simplifications (approximate arithmetic circuits, approximate storage, reduction on logic circuits, etc.), or software/algorithmic simplifications (exploring the preci-

sion scaling, with the skip of some tasks or memory accesses). Furthermore, there is a wide range of solutions addressing different parts of the encoder steps, such as the inter prediction (EL-HAROUNI et al., 2017; PRABAKARAN et al., 2019; PORTO et al., 2020), intra prediction (PASTUSZAK; ABRAMOWSKI, 2015), transforms (CHATTERJEE; SARAWADEKAR, 2019), and in-loop filters (PRAVEEN; ADIREDDY, 2013).

In EL-HAROUNI et al. (2017) it is proposed a novel approximate architecture for energy-efficient motion estimation in the HEVC encoder. Their solution employs different SAD accelerators with accurate and heterogeneous multi-level approximation modes for different block sizes, allowing the exploration of the trade-off between energy savings and video quality. Still, PRABAKARAN et al. (2019) propose an approximate multi-accelerator tiled architecture for energy-efficient motion estimation, also targeting HEVC inter prediction steps. In such work approximate adders were developed in order to build the SADs within the inter-frame prediction. Their design was conceived in a configurable way, being able to achieve power, latency, and/or energy savings. Furthermore, PORTO et al. (2020) also focus on HEVC inter prediction step, proposing a fast and energy-efficient approximate motion estimation architecture employing imprecise adders within the SAD calculation. They have designed the entire motion estimation architecture, embracing both IME and FME while introducing the developed approximate adders at SAD steps.

When looking for other encoding steps, PASTUSZAK; ABRAMOWSKI (2015) propose an algorithm and architecture design targeting the intra-frame prediction in the HEVC encoder. They developed a computationally-scalable algorithm and its hardware architecture in order to support the intra HEVC encoding. In their design, approximate computing is applied in order to explore the trade-off between throughput and coding efficiency by simplifying rate-distortion decisions. CHATTERJEE; SARAWADEKAR (2019) develops an approximated core transform architectures for HEVC using a decomposition method based on the Walsh Hadamard Transform (WHT). This work focuses on the transform step, being able to reach huge energy savings when compared with precise solutions. In PRAVEEN; ADIREDDY (2013) it is proposed an analysis and approximation of SAO (Sample Adaptive Offset) estimation for CTU-level HEVC encoder, thus targeting the in-loop filtering step. This work proposes two methods that are very suitable for designing pipelined architectures, including both software and hardware solutions.

3.2 NoC-Based Solutions

NoC-based systems present an interesting alternative to explore the inherent parallelism of the applications, since the processing of a given application can be split over several processing elements. In addition, they also bring scalabiity for the pro-

cessing due to the ability of variation on the number of processing elements, which becomes essential when dealing with applications presenting different throughput (e.g. video processing, which can present different spatial resolutions and frame rates). Due to these features, many works in the literature have been employing the use of NoC-based solutions in order to explore parallelism and provide scalability.

3.2.1 Energy-Efficient NoC Solutions for General Applications

There are several works in the literature aiming for energy efficiency on NoCs for general applications (ZHAN et al., 2013; CLARK et al., 2018; ALI et al., 2018; ZHENG; LOURI, 2019; TARIQ; WU; ABD ISHAK, 2020; WANG et al., 2020). These works propose energy-efficient NoC designs targeting real-time embedded systems running general applications.

In ZHAN et al. (2013) it is developed an energy-efficient NoC for real-time embedded systems through slack optimization. They propose a methodology to minimize the NoC's energy consumption without violating the deadlines of real-time applications, developing a formal approach based on network calculus to obtain the worst-case delay bound of all packets. Thus, they produce a safe estimation of the number of cycles that a packet can be delayed while meeting its deadline, which is so called worst-case slack. Based on that, they developed an optimization algorithm in order to trade the slacks for lower NoC energy, assigning different voltages and frequencies to different routers to reduce energy, while meeting the deadlines for all packets. CLARK et al. (2018) also develop a solution based on DVFS for NoC energy efficiency following a learning-based approach. In this work, the authors have applied machine learning techniques in order to enable energy-performance trade-offs at reduced overhead cost, in order to control the DVFS system.

ALI et al. (2018) investigate contention and energy-aware real-time task mapping on NoC-based heterogeneous MPSoCs. This architecture considers DVFS-enabled processors, while contention and energy-aware static mapping for real-time Directed Acyclic Graph (DAG) tasks with individual deadlines and precedence constraints is also studied. In ZHENG; LOURI (2019) it is proposed an energy-efficient network-on-chip design using reinforcement learning. They explore the dynamic interactions among power gating, DVFS, and system parameters, in order to learn the critical system parameters. Thus, the usage of Artificial Neural Network (ANN) is introduced, aiming a efficiently implementation of the state-action table required by reinforcement learning.

TARIQ; WU; ABD ISHAK (2020) propose an energy and memory-aware software pipelining streaming applications on NoC-based MPSoCs. They explore the problem of energy-aware scheduling of real-time applications, modelled by conditional retiming task graphs in order to minimize the total energy consumption, while meeting memory capacity constraints. In WANG et al. (2020) it is proposed an efficient task mapping

scheme targeting many-core systems. Since application task mapping has a significant impact on the efficiency of many-core system computation and communication, the authors proposed WAANSO solution, a scalable framework that incorporates a wavelet clustering-based approach to cluster application tasks, reaching significant energy consumption reduction.

3.2.2 NoC Solutions for Video Coding

3.2.2.1 Employing GPPs

Many works in current literature focus on video coding over NoCs (ALIKHAH-ASL; RESHADI, 2016; PENNY et al., 2019a; MA et al., 2015; MENDIS; AUDSLEY; INDRUSIAK, 2017), exploring latency improvement (ALIKHAH-ASL; RESHADI, 2016), performance evaluation targeting the meet of timing constraints (PENNY et al., 2019a), and dynamic and static task-allocation algorithms for video encoders (MA et al., 2015; MENDIS; AUDSLEY; INDRUSIAK, 2017; MENDIS; INDRUSIAK, 2016).

In ALIKHAH-ASL; RESHADI (2016) it is proposed a XY-Axis and distance-based NoC mapping, by proposing a low complexity heuristic algorithm for the application mapping onto NoC to improve latency, targeting the MPEG-4 encoder. In our previous work (PENNY et al., 2019a), we make a performance evaluation of HEVC Residual Coding Loop (RCL) mapped onto a NoC-based embedded platform. A set of analysis exploring the combination of different NoC sizes and task mapping strategies were performed, showing for the typical and upper-bound workload cases scenarios when the application is schedulable and meets the real-time constraints.

MA et al. (2015) implement a MVC (Multiview Video Coding) decoding on homogeneous NoC, investigating the routing strategies. They analyze which is the most suitable switching for the routers: a circuit or wormhole switching, by performing a comparison on decoding speed of the whole system, link utilization, and delay between these switching strategies. Their case study considers the decoding of an eight-view QVGA (Quarter Video Graphics Array - 320 x 240 pixels) onto a 4x4 mesh NoC at 30 fps, leading to the selection of circuit switching for routing.

Furthermore, MENDIS; AUDSLEY; INDRUSIAK (2017) proposed a dynamic and static task allocation for hard real-time video stream decoding on NoCs, targeting the MPEG-2 decoder. They describe two application and platform-aware run-time task mapping strategies, which attempt to decrease the end-to-end response-time of the video stream decoding jobs. Besides, MENDIS; INDRUSIAK (2016) present a run-time and low-communication overhead clustering-based HEVC tile to PE mapping scheme, taking into account the workload and task blocking behaviour. They have illustrated how a frame-level task graph can polynomially grow due to tile partitioning and inter-prediction. The proposed task-mapper clusters dependent tasks together on to the same or neighbouring PEs, reducing the inter-task data communication, leading to

significant reduction in data communication overhead and increased mean PE *idle* periods, also resulting in reduced energy consumption.

3.2.2.2 Employing Hardware Accelerators

NoC-based systems with GPPs as processing elements might not address the necessary performance to meet real-time constraints posed by energy/time demanding applications like video coding. In order to adress such an issue, hardware acceleration becomes essential, and its usage as processing elements in NoC-based environments has been addressed as an interesting solution by many works (NOURI; GHAZNAVI-YOUVALARI; NURMI, 2018; POURABED; NOURI; NURMI, 2018; PENNY et al., 2019b).

NOURI; GHAZNAVI-YOUVALARI; NURMI (2018) propose a design and implementation of multi-purpose accelerator on heterogeneous multi-core architecture, targeting the HEVC transforms: Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST). They employ the usage of template-based Coarse-Grained Reconfigurable Arrays (CGRAs) as accelerators on a Heterogeneous Accelerator-Rich Platform (HARP), arranged in a 3x3 NoC mesh topology, reaching enough throughput to deal with FHD videos at 30 fps. In POURABED; NOURI; NURMI (2018) it is proposed a similar solution, but focusing on the acceleration of the HEVC inverse transforms: Inverse Discrete Cosine Transform (IDCT) and Inverse Discrete Sine Transform (IDST). They also employ template-based CRGAs as accelerators on HARP, delivering throughput to handle FHD videos at 30 fps.

In our previous work (PENNY et al., 2019b), we have proposed a design space exploration of HEVC RCL mapped onto NoC-based embedded platforms, combining traditional CPUs and hardware accelerators as processing elements (PEs). We considered the encoding of FHD (1920 x 1080 pixels) and 4K (3840 x 2160 pixels) videos at 30 fps, when running at CPUs and hardware accelerators, respectively. A set of analysis exploring the combination of different NoC configurations and task mapping strategies were performed, with the development of a search algorithm called SNFT (Schedulability breakdown NoC Frequency Tracking algorithm). SNFT will be further introduced in this thesis, as a way to find the minimum NoC frequency for each configuration which becomes the system fully schedulable, showing for different workload cases scenarios when the application is schedulable and meets the real-time constraints.

3.3 Hardware Designs for Video Coding

Considering the severe constraints posed by video coding, especially when dealing with embedded battery-powered systems, dedicated hardware acceleration is manda-

tory, as widely demonstrated in current literature. The adoption of this strategy can improve both energy and performance efficiency of the applications, enabling the power-aware real-time processing of very demanding applications like video coding.

3.3.1 Precise Designs for FME

Many works in the literature propose hardware designs to deal with very complex tasks of the HEVC encoder, such as the inter-frame prediction. In AFONSO et al. (2016) it is developed a hardware design for the whole FME (interpolator + search), capable of reducing memory communication and computational effort by the use of only four sizes of squared-shaped prediction units (PUs), instead of using the 24 sizes supported by HEVC motion estimation. Besides the reduction of memory accesses and computational effort, it does not support other PU sizes and significantly affects the coding efficiency. HE et al. (2015) propose a high-throughput architecture for HEVC FME, also involving all FME steps. Besides reaching complexity reductions, it does not show a complete evaluation of impacts of the adopted complexity-reduction strategy whereas significantly affects coding efficiency. Furthermore, in PENNY et al. (2015) it was proposed a real-time and high-throughput architecture targeting the HEVC codec. In spite of the development of efficient hardware design, supporting all PU sizes, they focus only on the motion compensation (MC) step. LUNG; SHEN (2019) also presents the design and implementation of a highly efficient fractional motion estimation for the HEVC encoder. Despite obtaining high throughputs by the use of reconfigurable datapaths, neither of aforementioned works apply approximate computing solutions in their architectures in order to reduce both energy consumption and external memory communication.

3.3.2 Approximate Designs for FME

Many works introduce approximate computing solutions to develop architectures targeting specific accelerators for HEVC inter-prediction blocks. DIEFY; SHALABY; SAYED (2015) have proposed the development of efficient architectures for HEVC luminance interpolation filters. It applies approximate computing by changing the number of adders in the design, resizing the filter coefficients according to a proposed scaling factor modification to reduce hardware implementation complexity. In PALUMBO et al. (2016), it is developed a run-time coarse-grained adaptation solution to guarantee energy reduction, in constraint-aware or user-defined situations, while introducing a controllable quality degradation by approximate computing, targeting MC HEVC filters. Although they have obtained expressive gains in performance and area, with small quality degradation, they only act at the interpolation filters, not involving the whole interpolator unit. Moreover, they focus only on the motion compensation tool, at the decoder side, not targeting the FME step. Furthermore, SAU et al. (2017) proposed

an approximate hardware solution for the HEVC fractional pixel FPGA interpolator with reconfigurable and multi-frequency approximate computing. It guarantees a tunable interpolation system offering the energy-quality trade-off. However, it implements the architecture only on FPGA and does not present QoS (in terms of coding efficiency) results.

KALALI; HAMZAOGLU (2018) and SILVA; SIQUEIRA; GRELLERT (2019) propose the hardware implementation of approximate HEVC FME interpolation filters, exploring the idea of changing the coefficients of the interpolation filters in order to achieve complexity/energy reduction, under a small penalty on coding efficiency. In our previous work (PENNY et al., 2020), we have presented a low-power and memory-aware hardware architecture for the HEVC FME interpolator, proposing the development of two novel hardware designs for the interpolation filters, so called Approximate Unified FME Filters (AUFF). These solutions exploit the usage of approximate computing at both algorithmic and data levels, leading to a reduction in dissipated power and memory bandwidth. Therefore, although many works in the literature address the development of approximate hardware designs for the fractional motion estimation, many research opportunities are still open for novel contributions.

3.4 Summary and Challenges

The works in the literature aiming energy efficiency on NoCs for general applications (ZHAN et al., 2013; CLARK et al., 2018; ALI et al., 2018; ZHENG; LOURI, 2019; TARIQ; WU; ABD ISHAK, 2020; WANG et al., 2020) are unable to exploit the application specific characteristics to deliver high-throughput and energy efficiency. In turn, the works found in current literature that focus on video coding over NoCs (ALIKHAH-ASL; RESHADI, 2016; PENNY et al., 2019a; MA et al., 2015; MENDIS; AUDSLEY; INDRUSIAK, 2017), although proposing application-specific solutions, do not focus on energy consumption, work at the decoder side (less complex than the encoder) targeting low resolutions, and/or consider older video coding standards. NoC-based systems with hardware accelerators as processing elements for video coding are explored in NOURI; GHAZNAVI-YOUVALARI; NURMI (2018); POURABED; NOURI; NURMI (2018); PENNY et al. (2019b) but are not able to give energy/performance scalability (since approximate computing was not applied) and/or they do not reach enough throughput to deal with UHD videos. Regarding approximate computing for NoCs, despite providing significant energy reduction, related work BOYAPATI et al. (2017) focuses only on the NoC communication, not changing processing elements neither applying approximate computing over them. When it comes to approximate dedicated hardware accelerators for video coding, there is a wide range of solutions including different steps, such as inter-prediction (EL-HAROUNI et al., 2017; PRABAKARAN et al.,

2019; PORTO et al., 2020), intra-prediction (PASTUSZAK; ABRAMOWSKI, 2015), transforms (CHATTERJEE; SARAWADEKAR, 2019), and in-loop filters (PRAVEEN; ADIREDDY, 2013). However, none of these works makes a precise evaluation of application's resilience to errors and/or do not focus on high energy demanding steps like the Fractional Motion Estimation. Furthermore, none of them considers the employment of NoC solutions, performance/energy scalability, and support to multiple QoS levels.

Summarizing this discussion about related works, one can notice that many research opportunities remain open in different research fields, regarding the real-time processing of complex applications, guiding the development of this thesis, as follows:

- *Exploiting the inherent parallelism of the applications to provide high-throughput*;
- *Supporting scalability for distinct operation demands by employing different NoC sizes*;
- *Exploring approximate computing for multimedia hardware designs by deploying energy-efficient processing elements*;
- *Dealing with the trade-off between energy and QoS by leveraging application-specific properties/behavior*.

4 SCALABLE APPROXIMATE NETWORK-ON-CHIP (SAPP-NOC)

This thesis aims to research and propose architectural solutions leading to power-efficient and high-performance application-specific systems, allowing scalable real-time support for error-resilient applications. In order to achieve this goal, we follow a strategy exploiting *parallelism* to provide scalability and performance improvement, *hardware acceleration* and *approximate computing* to provide high-performance with power efficiency, and also by exploring *application-specific behavior* in order to keep reasonable QoS.

The parallelism is exploited by building the solution onto a NoC-based system, capable to explore the application inherent communication parallelism and to provide scalability across different demanding throughput. Approximate computing is exploited by the development of hardware accelerators employing the simplifications at different levels of approximation. Furthermore, the application QoS is kept at reasonable values by leveraging application-specific behavior, exploring error-resilient areas/steps by the employment of heuristics and machine learning-based solutions. All these assumptions lead to the development of the **S**calable **App**roximate **N**etwork-**o**n-**C**hip – SApp-NoC – used as our main case study and focusing on the HEVC FME step. SApp-NoC is presented in Figure 19 (a), as well as an overview of the main contributions of this work. We have tagged in Figure 19 (a) each contribution with a small circle having the corresponding chapter/section number where such a contribution is detailed. Furthermore, Figure 19 (b) shows the adopted methodology framework of this thesis.

Our NoC is organized in multiple neighbor Tiles to allow the NoC effective size to scale according to throughput requirements, i.e., the number of active processing elements varies according to video resolution and frame rate. The processing elements of the NoC are based on hardware acceleration, which could accelerate any application step, but in our case study targets the HEVC fractional motion estimation, presenting multiple levels of approximation. Hardware accelerators were named Approximate FME Filters - $FAPP_j$, where $j = [0,\ 1,\ 2,\ 3]$, varying the approximation level from precise (0) to most aggressive (3) (see Section 4.1 and Chapter 5 for a better detailing).

The hardware design takes into account the application behavior, focusing on low QoS degradation. In Chapter 6 details about the NoC architecture, application modeling, and **S**chedulability **N**oC **F**requency breakdown **T**racking algorithm (SNFT) algorithm are presented. At design time, algorithms are proposed to define the PE type (i.e., the approximation level), amount (Section 4.2) and placement (Section 4.3). Finally, at run-time, tasks are smartly allocated on SApp-NoC, following application behavior-based statistics (Section 4.4, whose fundamentals are described in Chapter 7).

Following what was stated in the background (Chapter 2), all the nodes are inter-

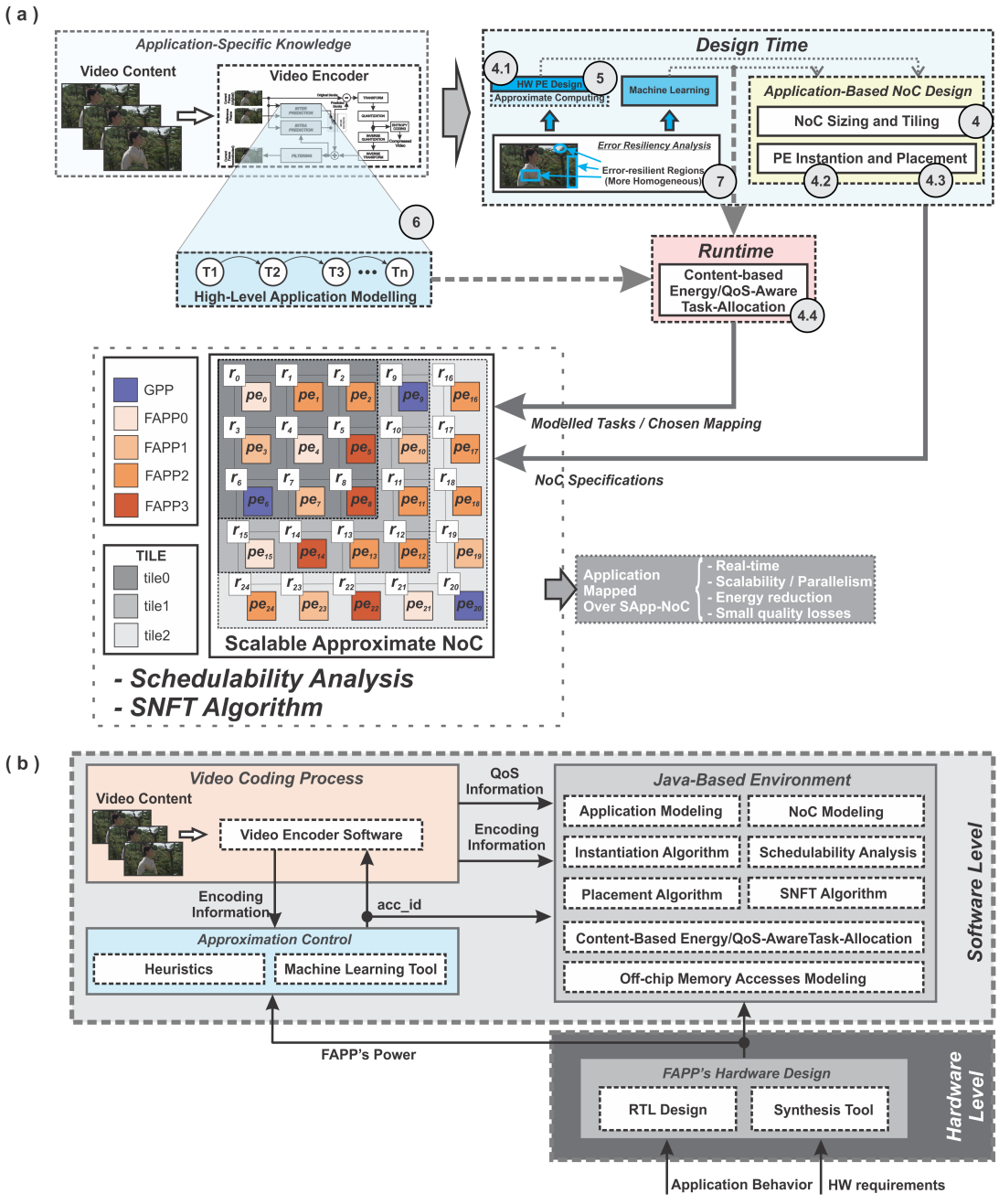

Figure 19 – Detailed methodology: (a) overview of the novel contributions of this work and (b) adopted framework.

connected, and each one has a processing element *PE*, linked internally to a local cache (omitted in the Figure 19 (a) for simplicity), which stores local information, and a router *r*, which routes the data packets towards their destinations. The communication between processing elements and the router is made by two unidirectional links (one from *PE* to *r* and other from *r* to *PE*). In this work, according to what was previously mentioned in Chapter 2, we have applied the widely used square-shaped 2D-mesh topology, considering wormhole NoC with priority-preemptive arbitration, widely studied in the literature due to its ability to provide resources for hard real-time guarantees. In order to determine whether application tasks being executed and communicating over SApp-NoC can fulfill the required timing constraints, the schedulability analysis, also presented in Chapter 2, was applied.

In Figure 19 (b) we present the adopted development framework. Our contributions are present at both software and hardware levels. At hardware level, we follow a Register Transfer Level (RTL) design, making the FME hardware development and employing a synthesis tool to estimate the power and chip area. At software level, the video encoder reference software is employed to analyze the behavior of the proposed simplifications, verifying the resulted QoS information for each proposed scenario. In addition, encoding information are also gathered from the encoder software, in order to allow the usage of approximation control. Such a control is also performed at software level by employing heuristics and a machine learning tool. The generated approximation control algorithm is inserted in the encoder software code, also aiming the analysis of the QoS behavior. All the other contributions are made employing a Java-based environment, were the application (HEVC FME), the NoC, and the off-chip memory communication are modeled in a high level of abstraction. All the other proposed algorithms and the schedulability analysis are performed considering the same Java-based environment.

Considering the adopted square-shaped 2D-mesh topology, with $n \times \ n$ available nodes, in order to define a proper sizing, we need to find an n such as $PE_{Active} = n^2 - m$ (where m is the number of nodes used for memory access, see Section 4.2 for details) is the number of processing elements that achieves real-time processing given a target throughput ($TPut$). It is worth mentioning that we consider all *PEs* to have the same processing performance regardless the approximation level (refer to Chapter 5 for details of the hardware design). To guarantee scalability across distinct throughput demands, we adopt the strategy of neighbor active topologies, here named Tiles, where a larger Tile contains the smaller Tiles. For instance, consider a system demanding two target throughputs $TPut_0$ and $TPut_1$ where $TPut_1 > TPut_0$. If the NoC size $n_0 \times n_0$ is enough to guarantee $TPut_1$, the same Tile size can be used to achieve real-time processing for both throughput scenarios. Otherwise, if $n_0 \times n_0$ is not enough to guarantee $TPut_1$, an n_1 such as $n_1 > n_0$ (i.e., $n_1 = n_0 + k$) needs to be used.

Two neighbor Tiles must be used if scalability is desired, where $Tile_0$ is sized n_0 x n_0 and $Tile_1$ sized $n_1 \times n_1$, i.e., $(n_0 + k) \times (n_0 + k)$. The NoC size required for the higher defined throughput defines the maximum NoC size ($Tile_1$ in this example).

To size and tile our system we have considered five throughput demands targeting FHD and UHD resolutions and frame rates: ($TPut_0$) FHD (1080p) at 30 fps (frames per second); ($TPut_1$) FHD (1080p) at 60 fps; ($TPut_2$) 4K UHD (2160p) at 30 fps; ($TPut_3$) 4K UHD (2160p) at 60 fps; and ($TPut_4$) 4K UHD (2160p) at 120 fps. We have considered the operational frequency from the smallest throughput as the basic PE frequency (f_{PE}), i.e., all PEs within the NoC will have the same frequency. Besides, considering a given NoC with $n \times n$ available nodes, and also considering a minimum NoC size 3x3, the number of PEs dedicated for memory access is $m = n - 2$.

Let consider that each throughput demand $TPut_i$ has an operational frequency f_i. The relation between f_i and f_{PE} inform us how much faster is the application having $TPut_i$ when compared with the basic PE frequency. Considering this simple relation, and also considering the addition of an extra PE to ensure the desired performance, the number of nodes N_i having frequency f_{PE}, required to perform $TPut_i$, is given as follows.

$$N_i = \left\lceil \frac{f_i}{f_{PE}} \right\rceil + 1 \tag{29}$$

Therefore, the number of active PEs will be equal to N_i. However, we need to size our NoC by finding its dimension n. Since $PE_{Active} = n^2 - m$, and $m = n - 2$, the number of active PEs can be expressed as

$$PE_{Active} = n^2 - n + 2 \tag{30}$$

Substituting (30) in (29):

$$n^2 - n + 2 = \left\lceil \frac{f_i}{f_{PE}} \right\rceil + 1 \tag{31}$$

If we simplify (31) we find the equation depicted in (32), which describes the dimension n of the square-shaped NoC being designed.

$$n^2 - n + \left(1 - \left\lceil \frac{f_i}{f_{PE}} \right\rceil\right) = 0 \tag{32}$$

The positive root of (32), given in (33), is the solution that informs the desired NoC dimension n for any given throughput scenario making the same adopted assumptions (number of nodes reserved for memory access, minimum NoC 3x3, and square-

shaped NoC).

$$n = \left\lceil \frac{1 + \sqrt{4 \cdot f_i / f_{PE} - 3}}{2} \right\rceil \tag{33}$$

As a result, considering the proposed throughput, we obtained $n = 1$, for FHD@30; $n = 2$, for FHD@60; $n = 3$, for 4K@30; $n = 4$, for 4K@60; and $n = 5$, for 4K@120. Since we considered 3x3 as the minimum NoC size, the proposed NoC-based system has three neighbor Tiles: $Tile_0 = 3 \times 3$, $Tile_1 = 4 \times 4$, and $Tile_2 = 5 \times 5$. Considering our case study, each PE was designed individually with a target frequency focusing on the minimum target throughput, i.e., FHD at 30 fps. Note that in order to address higher throughput the parallelism provided by the NoC infrastructure becomes essential.

Considering our case study, the methodology employed for the development of the proposed system can be expanded to any application allowing multi-level approximation. The hardware accelerator for the FME, one of the most power/processing hungry step in HEVC encoder (remember that a single 4K frame considering only square-shaped blocks need to calculate more than 10 billion samples and also that the FME is responsible for up to 60% of total encoding time in the HEVC (GRELLERT; BAMPI; ZATT, 2016)), is designed exploring approximate computing techniques to achieve power efficiency. The approximation is proposed at multiples levels by performing an error resiliency analysis in order to find more suitable regions to apply the approximation. Such analysis is done in two ways: by employing heuristics to decide the approximation level, leading to the development of the solution called Heuristic-based SApp-NoC (**HSApp-NoC**), and by employing decision trees (based on machine learning techniques), leading to the development of the solution called Machine Learning-based SApp-NoC (**MLSApp-NoC**). The developed hardware is employed as processing elements compounding a Network-on-Chip topology, to enable parallel processing featuring multiple neighbor Tiles to provide scalability. An algorithm for the multi-level approximate *PEs* instantiation based on the average selection of each FAPP, obtained as a consequence of the adopted heuristics/ML solution, is proposed in Section 4.2. Furthermore, an algorithm for the multi-level approximate *PEs* placement in order to reduce communication distances and guarantee availability of distinct *PEs* in different Tiles configurations is also proposed in Section 4.3. At run-time it is proposed in Section 4.4 a task-mapping algorithm to distribute the tasks between different multi-level approximate *PEs* favoring power reduction whereas avoiding QoS degradation. The application-specific step (FME) is modelled at high level of abstraction. The so called **S**chedulability **N**oC **F**requency breakdown **T**racking algorithm (SNFT) is also proposed aiming to find the optimal NoC communication frequency (discussed in Chapter 6) and a schedulability analysis (discussed in Chapter 2) of the generated mapping is performed to guarantee the meeting of real-time constraints.

4.1 Processing Elements Design

The processing elements of SApp-NoC were conceived having two natures: general purpose processors (*GPPs*) or hardware accelerators. The *GPPs* are employed in PEs used to process any kind of task, specifically in SApp-NoC they are used only to process tasks related to memory communication (reading or writing). The hardware accelerators are employed in PEs used to process the target application (in our case study the HEVC FME). The design of these PEs considers both precise and approximate solutions, which were named Approximate FME Filters (FAPP). As aforementioned, $FAPP_j$, where $j = [0,\ 1,\ 2,\ 3]$, varies the approximation level from precise (0) to most approximate (3). As will be further discussed in Chapter 5, we have named the FME filters as $FAPP_j$. A given FME architecture (interpolation + search) implements only one type of FAPP for filtering when performed at SApp-NoC. It means that the level of approximation (if any) is always the same for a given FME architecture (the variation of the approximation internally to the FME engine is not allowed). Each PE of SApp-NoC (except the *GPPs*) is a unit performing the entire FME (precise of approximate), which means that only one type of FAPP is used at a given PE. Hence, for simplicity, we have also named the PEs as FAPPs, e.g., a PE FAPP1 will perform the entire FME for a given block, using FAPP1 during the FME filtering steps. Chapter 5 details the PEs design and discusses the main aspects about the developed architectures.

4.2 Processing Elements Instantiation Algorithm

Given the sizing and tiling of the NoC, and given the number of levels of approximation supported by the multi-level approximate *PEs* (four in our case study), it is necessary to instantiate these processing elements within SApp-NoC, establishing the number of units for each PE ($FAPP_0$, $FAPP_1$, $FAPP_2$, and $FAPP_3$). For that, we observe the application and measure the percentage of occurrences ($occur_{PE_x}$) each type of PE is called. Note: the algorithm responsible to select the level of approximation is presented in Chapter 7, so the percentage of PEs ocurrences vary for HSApp-NoC and MLSApp-NoC but the instantion algorithm is the same. Additionally, one restriction applies: at least one instance of each *PE* type must exist within the smallest NoC Tile in order to allow any approximation level even for less demanding scenarios. In addition, the *PEs* design must be done considering if the individual PE would deal with the smallest throughput alone, i.e., each PE is designed considering the necessary frequency for FHD@30fps.

The instantiation pseudo-algorithm presented in Figure 20 starts by instantiating the memory access units (line 3). General purpose processors (*GPP*) are used to control the memory access. For NoCs up to 3x3 ($n \leq 3$), one single *GPP* is enough ($num_{GPP} = 1$). However, every time *n* is incremented, a new *GPP* is required, i.e., for

$n \geq 3$; $num_{GPP} = n - 2$. After that, for each Tile, the percentage of call occurrences for the approximate *PE* types are sorted in crescent order (line 5). Thus, the less called *PE* types are instantiated first to the available *PEs*, respecting a simple proportional relation defined according to what is given in line 8, as follows:

$$num_{PE_x} = \left\lceil \left(available_{PE} \cdot \frac{occur_{PE_x}}{100} \right) \right\rceil \tag{34}$$

This process is repeated for all *PE* types except for the most used one (lines 6-13). The last *PE* type, the most called one, is instantiated to all remaining nodes available within the NoC (line 11). This algorithm guarantees the instantiation of at least one *PE* of each type if the number of total *PEs* in the NoC (nodes) is larger than the number of *PE* types. This is always true for SApp-NoC.

1: ***determine the number of PEs*** $(occur_{PE[\,]}, n)$
2: $available_{PE} = n \times n$
3: $num_{PE[0]} = num_{GPP} = max(1,\ n-2)$
4: $available_{PE} = available_{PE} - num_{GPP}$
5: $occur_{PE} = sort(occur_{PE[\,]},\ 'crescent')$
6: **for** $(x = 1,\ x \leq PE_{TYPES}, x++)$ **do**
7: **if** $(x \neq PE_{TYPES})$ **then**
8: $num_{PE[x]} = ceil(available_{PE} \cdot ocur_{PE[x]}/100)$
9: $available_{PE} = available_{PE} - num_{PE[x]}$
10: **else**
11: $num_{PE[x]} = available_{PE}$
12: **end if**
13: **end for**
14: **return** num_{PE}

Figure 20 – Instantiation algorithm pseudo-code.

In our particular case study, the statistical distribution (deeply discussed in the following chapters) is as follows:

- **HSApp-NoC**: $occur_{FAPP3} = 82.26\%$, $occur_{FAPP1} = 8.94\%$, $occur_{FAPP0} = 5.35\%$, and $occur_{FAPP2} = 3.45\%$. For $Tile_0 = 3 \times 3$, $num_{GPP} = 1$, $num_{FAPP0} = 1$, $num_{FAPP1} = 1$, $num_{FAPP2} = 1$, and $num_{FAPP3} = 5$. For $Tile_1 = 4 \times 4$, $num_{GPP} = 2$, $num_{FAPP0} = 1$, $num_{FAPP1} = 2$, $num_{FAPP2} = 1$, and $num_{FAPP3} = 10$. For $Tile_2 = 5 \times 5$, $num_{GPP} = 3$, $num_{FAPP0} = 2$, $num_{FAPP1} = 2$, $num_{FAPP2} = 1$, and $num_{FAPP3} = 17$.

- **MLSApp-NoC**: $occur_{FAPP2} = 35.72\%$, $occur_{FAPP3} = 27.02\%$, $occur_{FAPP0} = 20.62\%$, and $occur_{FAPP1} = 16.64\%$. For $Tile_0 = 3 \times 3$, $num_{GPP} = 1$, $num_{FAPP0} = 2$, $num_{FAPP1} = 2$, $num_{FAPP2} = 2$, and $num_{FAPP3} = 2$. For $Tile_1 = 4 \times 4$, $num_{GPP} = 2$, $num_{FAPP0} = 3$, $num_{FAPP1} = 3$, $num_{FAPP2} = 5$, and $num_{FAPP3} = 3$.

For $Tile_2 = 5 \times 5$, $num_{GPP} = 3$, $num_{FAPP0} = 4$, $num_{FAPP1} = 4$, $num_{FAPP2} = 10$, and $num_{FAPP3} = 4$.

Note that different FAPP occurrences incur into different numbers of instantiated FAPPs. The number of instantiated FAPPs for all considered scenarios can be observed in Figure 23, presented in Section 4.4.

4.3 Processing Elements Placement Algorithm

Since we have determined the number of each *PEs* in the NoC, we use the placement pseudo-algorithm presented in Figure 21 to place the *PEs* in the NoC. Note that, similar to the instantiation algorithm, the different solutions (HSApp-NoC and MLSApp-NoC) only impact on the placement algorithm decisions, being the same algorithm applied over the both of them. For that, some constraints are taken into account. First, any *GPP* must be at a NoC border and its position must be determined in order to maximize the average distance among *GPPs* (line 3), favoring a better packet flow distribution within the SApp-NoC. Once the best positions are defined, the *GPPs* are assigned to the correspondent positions (line 4) in the previously initialized NoC structure (line 2). For SApp-NoC specific case, the Tile sizes range from 3x3 to 5x5 and $num_{GPP} = 3$. Thus, the best *PEs* to place the *GPPs* are 06, 09, and 20 (see Figure 19).

1: ***determine PE placement*** $(num_{PE[\,]},\ n)$
2: $initNoC\ (NoC[\,],\ n)$
3: $pos[\,] = findPositions\ (num_{PE[0]},\ NoC[\,],\ 'maximize')$
4: $place\ (pos[\,],\ NoC[\,],\ 0)$
5: **for** $(x = 1,\ x \leq PE_{TYPES}, x++)$ **do**
6: $\quad pos[x] = findPositions\ (num_{PE[x]},\ NoC[\,],\ 'minimize')$
7: $\quad place\ (pos[\,],\ NoC[\,],\ x)$
8: **end for**
9: **return** $NoC[\,]$

Figure 21 – Placement algorithm pseudo-code.

Following the same order as the instantiation algorithm, the less used/instantiated *PEs* are placed first aiming at minimizing the average distance to the *GPPs* (lines 6-7). The rationale behind prioritizing the placement of less instantiated *PE* types is as follows: since there will be less *PEs* of that specific type (in some cases a single one) the run-time task-mapping algorithm will have few options for allocation, thus, to compensate that, we minimize the distance to *GPPs* to provide more and shorter communication routes. Considering our specific case study, the *GPPs* are placed first, then $FAPP_2$, $FAPP_0$, $FAPP_1$, and $FAPP_3$ are placed for HSApp-NoC; and after *GPPs*,

$FAPP_1$, $FAPP_0$, $FAPP_3$, and $FAPP_2$ are placed for MLSApp-NoC. The distance metric is calculated adding the distance in X-axis to the distance in Y-axis (since a 2D-mesh XY is applied). The $findPositions(\ )$ algorithm scans all NoC positions to find where *GPPs* are, always a position is found, FAPPs less used are placed near to such point. Given the small number of less used FAPPs it is enough to guarantee their positions as close as possible to *GPPs*. Finally, the most used *PE* type is placed at all remaining available *PEs*. SApp-NoC's final *PEs* placement is shown in Figure 19.

4.4 Content-Based run-time Energy/QoS-Aware Task Allocation

Our run-time energy/QoS-aware task allocation algorithm leverages video content properties to reduce energy consumption by allocating tasks to approximate *PEs* (FAPPs) whereas reducing quality degradation. For that, we exploit the application knowledge, following decisions based on heuristics for HSApp-NoC and building a decision tree based on Machine Learning in order to decide the best approximation level for MLSApp-NoC. The heuristics consider the TZS behavior and the decision tree leverages many encoding parameters to make the best decision. The whole study regarding the approximation control is presented in Chapter 7. Considering our case study, the heuristic decision (HSApp-NoC) and decision tree (MLSApp-NoC) are performed over each block before the FME, thus every block arriving at the FME to be processed carries with itself the information about the selected approximation level (*approx_level*). Figure 22 details the allocation algorithm where *approx_level* denotes the approximation level that must be employed for a given block being coded. Each task in the *tasks* set is an *FME* task for a 64x64 block (see Chapter 6 for application modeling comprehension).

```
1:  determine task allocation (tasks,  approx_level)
2:  for task in tasks do
3:     if task = mem_access then
4:        allocate task at GPP
5:     else if task = FME && approx_level = 0 then
6:        allocate task at FAPP_0
7:     else if task = FME && approx_level = 1 then
8:        allocate task at FAPP_1
9:     else if task = FME && approx_level = 2 then
10:       allocate task at FAPP_2
11:    else if task = FME && approx_level = 3 then
12:       allocate task at FAPP_3
13:    end if
14: end for
```

Figure 22 – Task allocation algorithm pseudo-code.

Whenever a task arrives at SApp-NoC, it will have a flag informing the approximation level that must be used. Then, the task is allocated to an available *PE* of the proper approximation level, according to the algorithm presented in Figure 22. In a simple way, in cases where the task is the FME and *approx_level = 0*, the task is allocated to one $FAPP_0$ *PE*. When the task is the FME and *approx_level = 1*, the task is allocated to one $FAPP_1$ *PE*. In cases where the the task is the FME and *approx_level = 2*, the task is allocated to one $FAPP_2$ *PE*. Finally, for cases where the task is the FME and *approx_level = 3*, the task is allocated to one $FAPP_3$ *PE*. All tasks being mapped will consider a crescent order of availability within the *PEs* of the same type. A task never is allocated twice in a *PE* while all *PEs* of same type have not been allocated already. Memory related tasks (reading and writing) are always allocated in *GPPs*. In this case, the NoC is subdivided into regions of mapping, guaranteeing sub-regions of traffic flows, as showed in Figure 23.

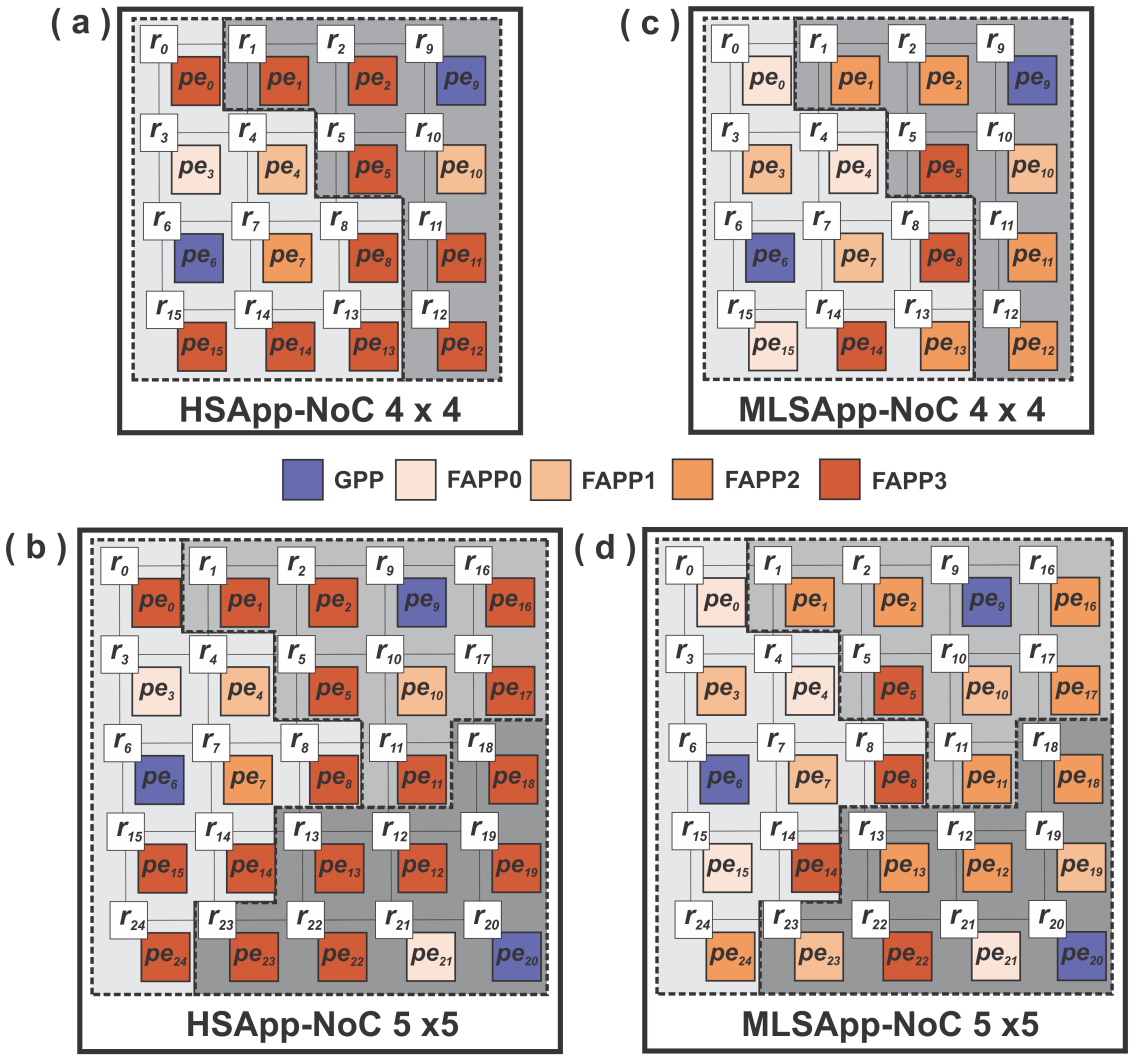

Figure 23 – Sub-regions for task allocation considering (a) HSApp-NoC 4x4, (b) HSApp-NoC 5x5, (c) MLSApp-NoC 4x4, and (d) MLSApp-NoC 5x5.

In Figure 23, we also show the number of each instantiated FAPP and their placed positions. In Figure 23 (a) and Figure 23 (b), we show the NoC distribution for HSApp-NoC for sizes 4 x 4 and 5 x 5, respectively, and in Figure 23 (c) and (d) we show the NoC distribution for MLSApp-NoC for sizes 4 x 4 and 5 x 5, respectively.

Note that for each *GPP* there is a sub-region containing other *PEs*. The memory related tasks of a *FME* task (reading or writing) must be allocated at a *GPP*. In this case, our allocation strategy guarantees that these tasks are always allocated in the *GPP* within the same sub-region, avoiding longer traffic flows crossing the NoC. Each *GPP* has its own sub-region, concentrating the allocation of memory related tasks of all tasks allocated in the *PEs* within the same sub-region. Obviously, when SApp-NoC sizes 3 x 3 there are no sub-regions to select, since only one *GPP* is available.

4.5 Summary

In this chapter we have summarized the methodology for the development of SApp-NoC and the upcoming solutions HSApp-NoC and MLSApp-Noc, showing the main contributions of this thesis. We discussed how SApp-NoC is sized and divided in tiles. In Section 4.1 the processing elements design was briefly commented, since in Chapter 5 it will be better detailed. Based on the application behavior, in Section 4.2 the instantiation algorithm was proposed in order to instantiate the amount of each PE. A placement algorithm was proposed in Section 4.3 in order to place each PE in SApp-NoC, favoring a better communication of flows across the NoC. Finally, in Section 4.4 the task-allocation algorithm is also developed, conceived in order to guide the mapping of tasks onto the corresponding PEs, according to the desired level of approximation.

5 FME MULTI-LEVEL APPROXIMATE HARDWARE ACCELERATORS

This chapter discusses the design of the HEVC FME multi-level approximate hardware accelerators, used as processing elements of our main contribution SApp-NoC. Since the level of approximation can vary, we will have different architectures, but the methodology of development and the final FME structure is the same, showed in Figure 24.

The HEVC FME multi-level approximate hardware architecture is presented in Figure 24, which shows the interpolator with the filtering and the other FME steps. Note that this is a template architecture, the generic FAPP showed in Figure 24 can be any of the considered FAPPs. The filters are instantiated in parallel within the interpolator architecture to calculate the desired number of samples. The selected parallelism was defined in the Basic Processing Unit (BPU) concept introduced in our previous works (PENNY et al., 2015, 2019), which have demonstrated that any block size could be processed using fixed BPU sizes - e.g., a 16x16 block can be processed using four 8x8 BPUs - and stated that the best BPU size for the HEVC is 8x8. Thus our design contains eight horizontal filters (H-Filter) and eight vertical filters (V-Filter). The input of the architecture is a row of the reference matrix read at every clock cycle, having a variable number of samples (*N* in Figure 24), with *M* input samples on the filters, which depends on the approximation level applied, i.e., which FAPP is operating. Note that the FAPP detailing is present in Section 5.1.

Depending on the proposed design (they are conceived separately, there is not a possibility for configurable FAPP usage in this work), each H-filter can have internally the following distribution (represented as a generic box FAPP in Figure 24): FIR filters UP, MIDDLE, and DOWN, for the FAPP0 and FAPP1 solutions, a FAPP2 or a FAPP3, for the other approximate solutions. They calculate the fractional samples *a*, *b*, and *c*. Furthermore, there is a simple bypass for the integer sample. Note that, eventually, depending on the approximation level chosen, these calculated samples could assume slightly different values when compared with the FAPP0 case (original), leading to losses on coding efficiency verified and presented in the next section.

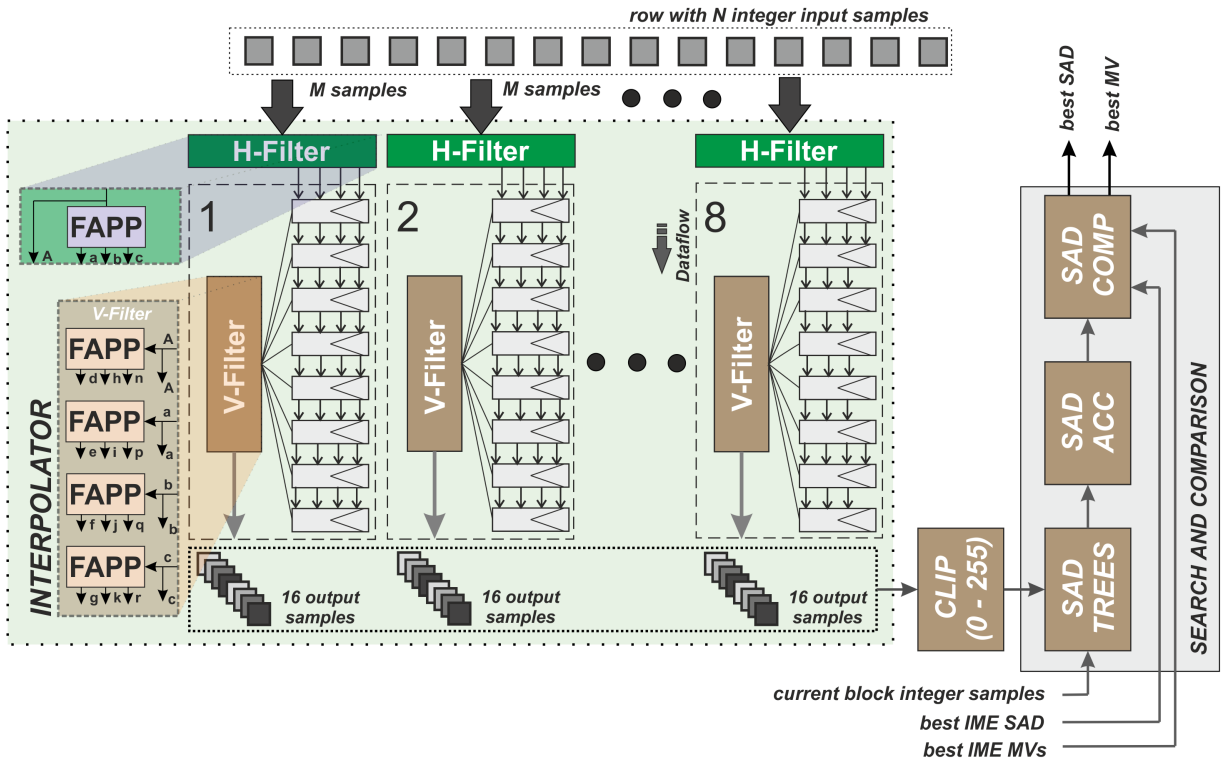

Figure 24 – HEVC FME hardware design.

The outputs of the H-filters feed a register chain having eight positions. On each cycle, each position of the register chain is forwarded. After eight cycles, all positions are filled. Then, all their positions feed vertical filtering. Although V-filters are similar to H-filters, at this time the number of input samples is four times greater, so four FAPPs are needed. Inside these filters, all the other fractional samples are calculated. Furthermore, integer samples (*A*), and the fractional samples (*a*, *b*, and *c*), bypass the filters. The V-filter outputs are always a set of 16 samples (15 fractional and the integer), related to each line from the original block, delivered at each clock cycle. After fifteen clock cycles all samples (integer/half/quarter) from an 8x8 block are delivered by the interpolator. These samples are further used at search and comparison step, presented in the next section.

5.1 Filters Design

In HEVC the fractional samples are calculated using Finite Impulse Response (FIR) filters, with 7- or 8- taps, whose coefficient values as well as calculated samples were presented in Table 1 (Chapter 2). As aforementioned, these filters are called UP, MIDDLE, and DOWN, according to the samples they calculate. Hardware implementation of these filters is named Approximate FME Filters (FAPP). The design following exactly the values defined by the standard is denoted as the **FAPP0** solution, which is presented in Figure 25, showing FAPP0 MIDDLE design, and Figure 26, showing FAPP0

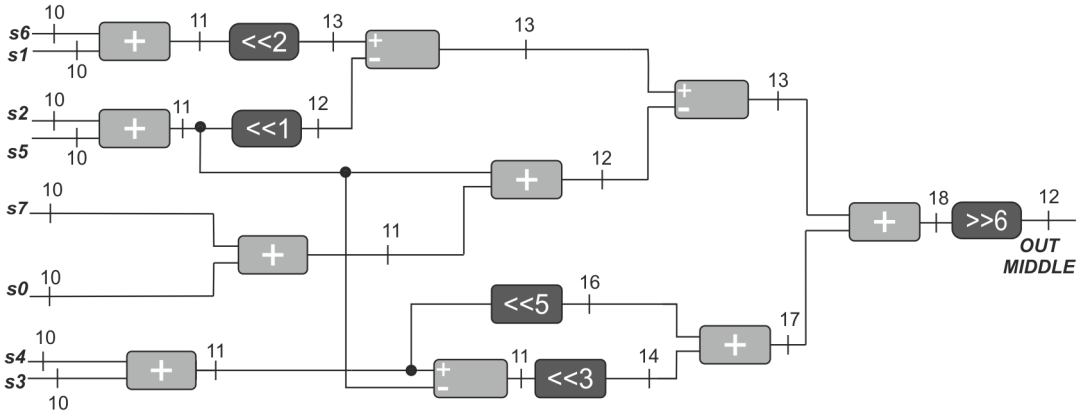

Figure 25 – FAPP0 MIDDLE hardware design.

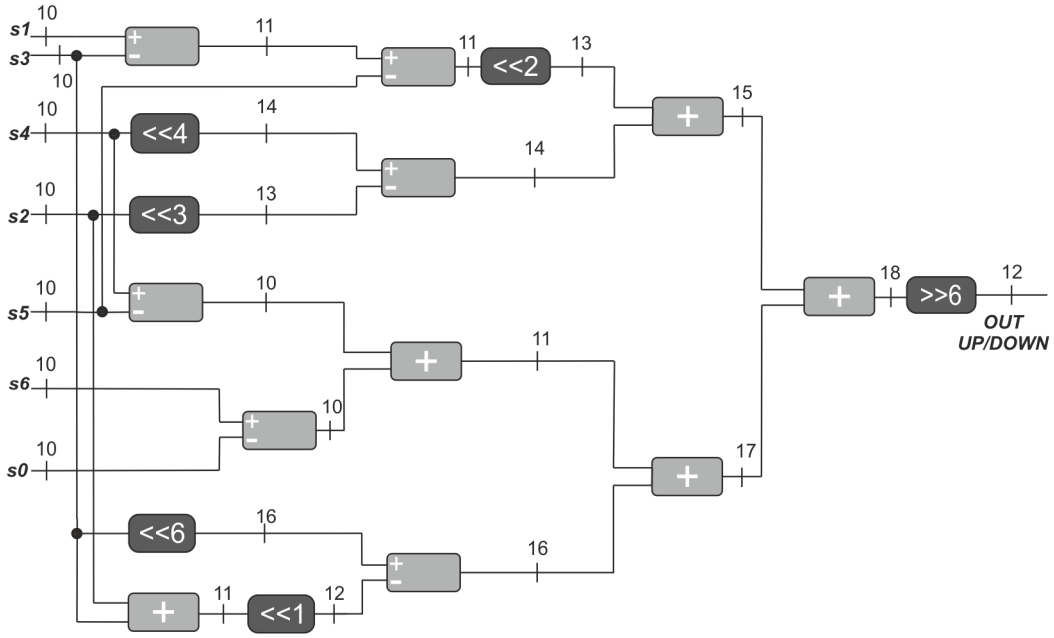

Figure 26 – FAPP0 UP/DOWN hardware design.

UP/DOWN design. Note that since UP and DOWN filters share the same coefficients (in the inverse order), they can share the same hardware design, only changing the input order when necessary, i.e., UP/DOWN filter need to be instantiated twice for each FAPP0, addressing both UP and DOWN filters in a parallel way. All multiplications and divisions were replaced by shifts and adders in order to optimize the hardware design.

The inputs are the integer samples (s_i) regarding the corresponding reference matrix from the block being encoded. The MIDDLE filter is an 8-tap FIR filter, so it requires eight input samples. The same way, UP/DOWN (7-tap) requires seven samples. The adopted hardware design tries to simplify the calculation, sharing coefficients when possible. The bit depth during all steps is depicted in both Figures 25 and 26. The input samples have 10 bit depth in order to handle samples coming from other calculations (e.g. during the vertical filtering when receiving samples from the horizontal

filtering). Considering that we need to instantiate UP/DOWN filter twice within a single FAPP0, plus a MIDDLE filter, in total, FAPP0 design demands 17 shifters and 32 adders/subtractors.

Approximate computing is applied in two distinct levels during the filters design. The algorithm level approximation is performed by changing the coefficients to hardware-friendly values. The data-level approximation is performed by discarding taps of the interpolation filter. The proposed hardware-friendly coefficients (in format of 2^n, represented in hardware by a simple shifter, shifted n times to the left if multiplying or to the right if dividing) are presented on Table 2. Three approximate solutions are proposed: in the first one, called FAPP1 (Figure 27, for FAPP1 MIDDLE, and Figure 28, for FAPP1 UP/DOWN), we have removed both external taps, keeping the other filter coefficients original. In the second, called FAPP2 (Figure 29), we have changed the coefficients into hardware-friendly values, keeping the same number of taps. In the third solution, called FAPP3 (Figure 30), besides changing coefficients values, we have reduced the number of taps from eight to two.

Table 2 – FAPP Coefficients

Filter		***Coefficients***	***Evaluated Samples***
FAPP0	*UP*	{-1, 4, -10, 58, 17, -5, 1}/64	*a, d, e, f, and g*
	MIDDLE	{-1, 4, -11, 40, 40, -11, 4, -1}/64	*b, h, i, j, and k*
	DOWN	{1, -5, 17, 58, -10, 4, -1}/64	*c, n, p, q, and r*
FAPP1	*UP*	{4, -10, 58, 17, -5}/64	*a, d, e, f, and g*
	MIDDLE	{4, -11, 40, 40, -11, 4}/64	*b, h, i, j, and k*
	DOWN	{-5, 17, 58, -10, 4}/64	*c, n, p, q, and r*
FAPP2	*UP*	{ -1, 4, -16, 64, 16, -4, 1 }/64	*a, d, e, f, and g*
	MIDDLE	{-1, 8, -8, 32, 32, -8, 8, 1}/64	*b, h, i, j, and k*
	DOWN	{1, -4, 16, 64, -16, 4, -1}/64	*c, n, p, q, and r*
FAPP3	*UP*	{ 3, 1}/4	*a, d, e, f, and g*
	MIDDLE	{1, 1}/2	*b, h, i, j, and k*
	DOWN	{1, 3}/4	*c, n, p, q, and r*

FAPP 1 design, presented in Figures 27 and 28, follows the same method applied to FAPP0 (requiring two UP/DOWN designs and a MIDDLE design). The only difference is the withdraw of more external taps. Since the removed taps present a small height, one can infer that such changes will have small impact on coding efficiency (QoS) and will present a reduction on the usage of hardware resources, demanding less samples from the off-chip memory. Indeed, a single FAPP1 design requires 17 shifters and 26 adders/subtractors.

Note that, since there is a huge similarity between the coefficients when dealing with designs presenting more aggressive approximation, the filters UP, MIDDLE, and DOWN were merged into a unique filter, when considering FAPP2 and FAPP3 solutions. FAPP2 design is presented in Figure 29, It has eight input samples (the same

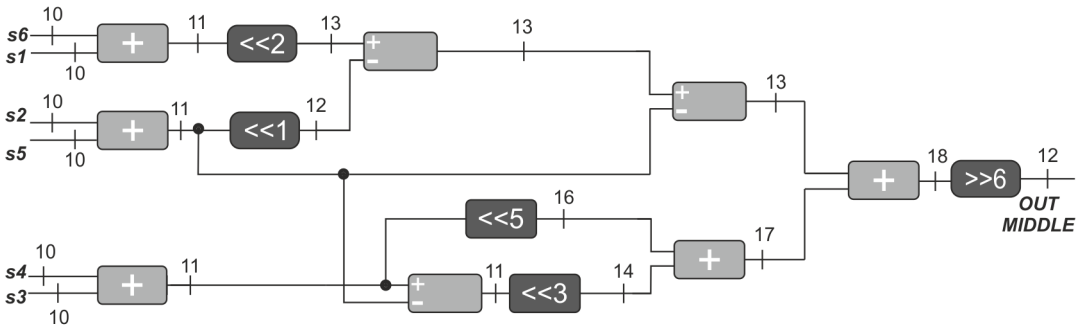

Figure 27 – FAPP1 MIDDLE hardware design.

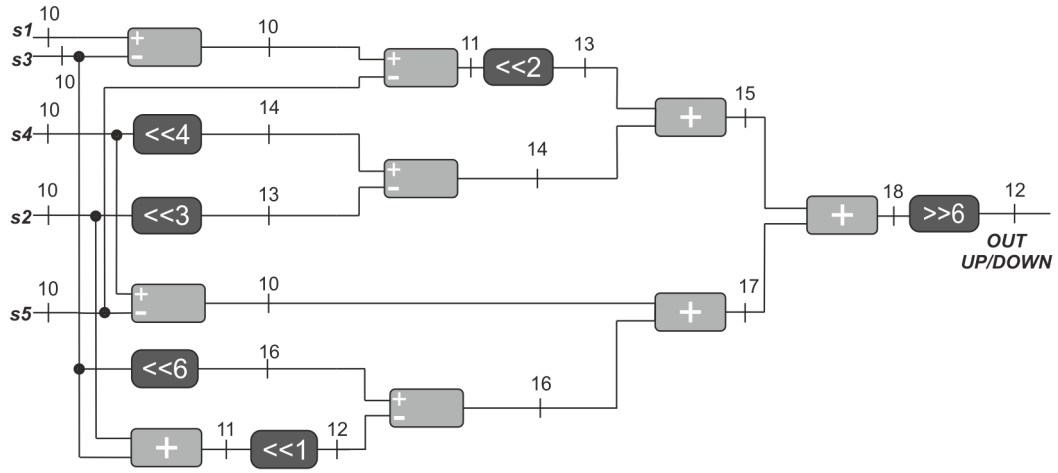

Figure 28 – FAPP1 UP/DOWN hardware design.

from FAPP0). All coefficients are in the form 2^n, transforming the complex task of a multiplication into a simple shifter. Such assumption, along the sharing of subexpressions lead to a hardware optimization, reducing the dissipated power.

FAPP3 design is presented in Figure 30, applying the most aggressive approximation. It has only two input samples, performing simple calculations with only two samples. In FAPP3 design it is reached the highest power saving due to the aggressive adopted simplifications.

All FAPPs deliver the calculated samples of all fractional positions (half and quarter precision). The architectures have outputs with different bit depths, depending on the number of samples (s_i) received in the input and intermediary calculations. Thus, a clipper (omitted in the figures to optimize the representation) keeps the outputs with 10-bit depth.

When considering the processing of 8x8 blocks, in FAPP1 a reduction on the number of samples fetched from the memory of up to 25% is reached, as we have previously showed in PENNY et al. (2020), whereas in FAPP3 a reduction on memory communication of 64% can be observed, as detailed in Figure 31, since less input samples are necessary. Commonly, interpolation filters with n taps require $n - 1$ sample borders, e.g. an 8x8 blocks requires a reference matrix of 15x15 samples (considering a

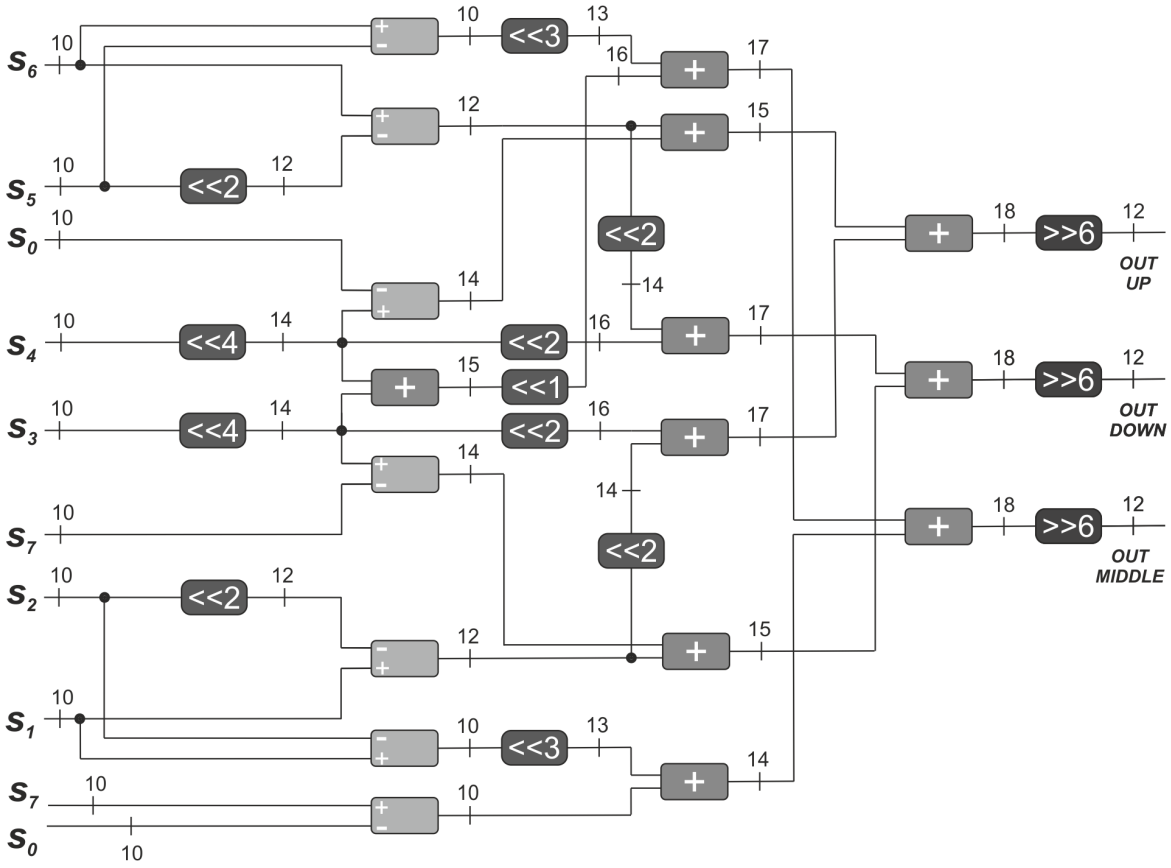

Figure 29 – FAPP2 hardware design.

filter having 8- taps). Observe in Table 3 and Figure 31 that, on the one hand, FAPP0 and FAPP2 require the original number of input samples, once they do not change the number of taps, and, on the other hand, FAPP1 requires an input matrix of 13x13 and FAPP3 requires an input matrix of 9x9. Table 3 summarizes the maximum number of samples required by each FAPP in the input and the correspondent size of its input matrix, as well as the number of operators that are spent by each FAPP (shifters and adders/subtractors). When analyzing Table 3, one can notice that FAPP3 is the one presenting the smallest amount of resource usage, thus we can infer that this FAPP provides the highest power savings, as will be observed in Section 5.3.

It is worth to notice that the approximate solution was applied only at the FME step, which belongs to the encoder side. Since standard compliance is mandatory at the

Table 3 – FAPP inputs and resources detailing

Filter	*Num. Input Samples*	*Input Matrix Size*	*Num. Shifters*	*Num. Add/Sub*
FAPP0	8	15x15	17	32
FAPP1	6	13x13	17	26
FAPP2	8	15x15	14	17
FAPP3	2	9x9	5	5

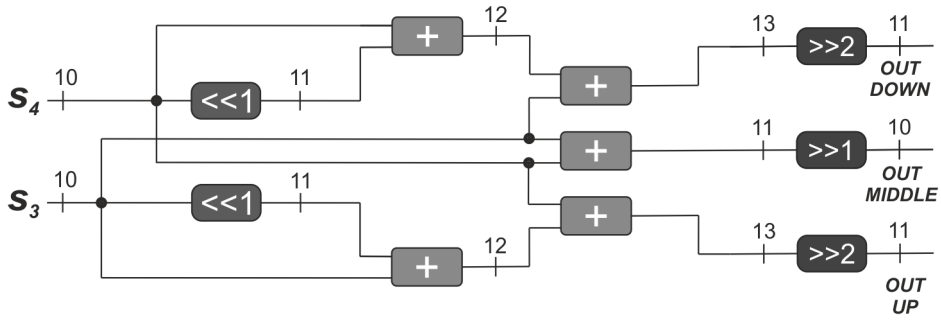

Figure 30 – FAPP3 hardware design.

decoder side, the decoder steps existing inside the encoding flow (e.g., the motion compensation - MC), must not be altered, under penalty of making the encoded video not compliant with the standard or inserting encoder-decoder drifting.

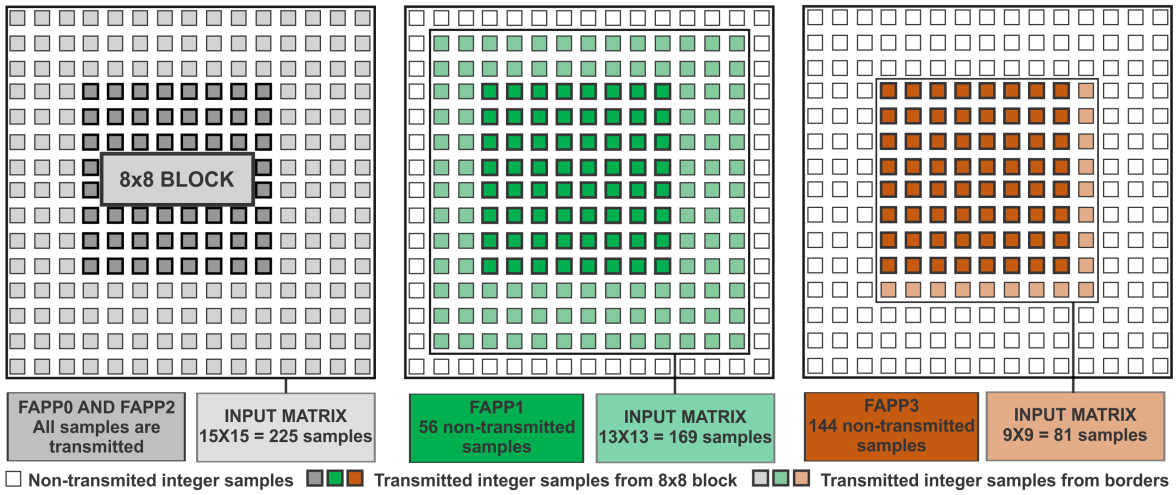

Figure 31 – Reference matrices scenarios for FAPP0, FAPP1, FAPP2, and FAPP3.

5.2 Search and Comparison Design

The calculated fractional samples are used at the Search and Comparison step, detailed in Figure 32 (a), where a similarity criterion is applied in order to determine the best results, i.e., which fractional block is the most similar when compared with the current block (which is being predicted). The proposed architecture uses the Sum of Absolute Differences (SAD) as similarity metric, previously explained in this work.

Once the interpolated samples are clipped (see Figure 24), in order to keep their values between 0 and 255 (range of values assumed by samples regarding the bit depth). The first step of search and comparison is the SAD TREE. During this step, the SAD of all possible 15 blocks, compounded by fractional samples, are calculated. In order to do that, the SAD TREE calculates the SAD between the fractional samples and the current block integer samples (which is being predicted). SAD TREEs employ

three pipeline stages, as shown in Figure 32 (b). First it determines the difference between the samples and obtains the absolute value. Next, all those differences are summed up to determine the SAD.

Next, the calculated SAD values are stored, since the design does not evaluate all samples at the same time. The SAD ACCUMULATOR (detailed in Figure 32 (c)) stores the partial SADs. When all samples from a given block size are already calculated, it forwards the calculated SADs and resets the registers. In the SAD COMPARATOR unit, (detailed in Figure 32 (d)) the calculated SADs are compared with each other and with the SAD of the best IME block, the smallest SAD (best SAD) is then delivered along its motion vectors (MVs - in X-axis and Y-axis) (which points to the position of the block having the best SAD).

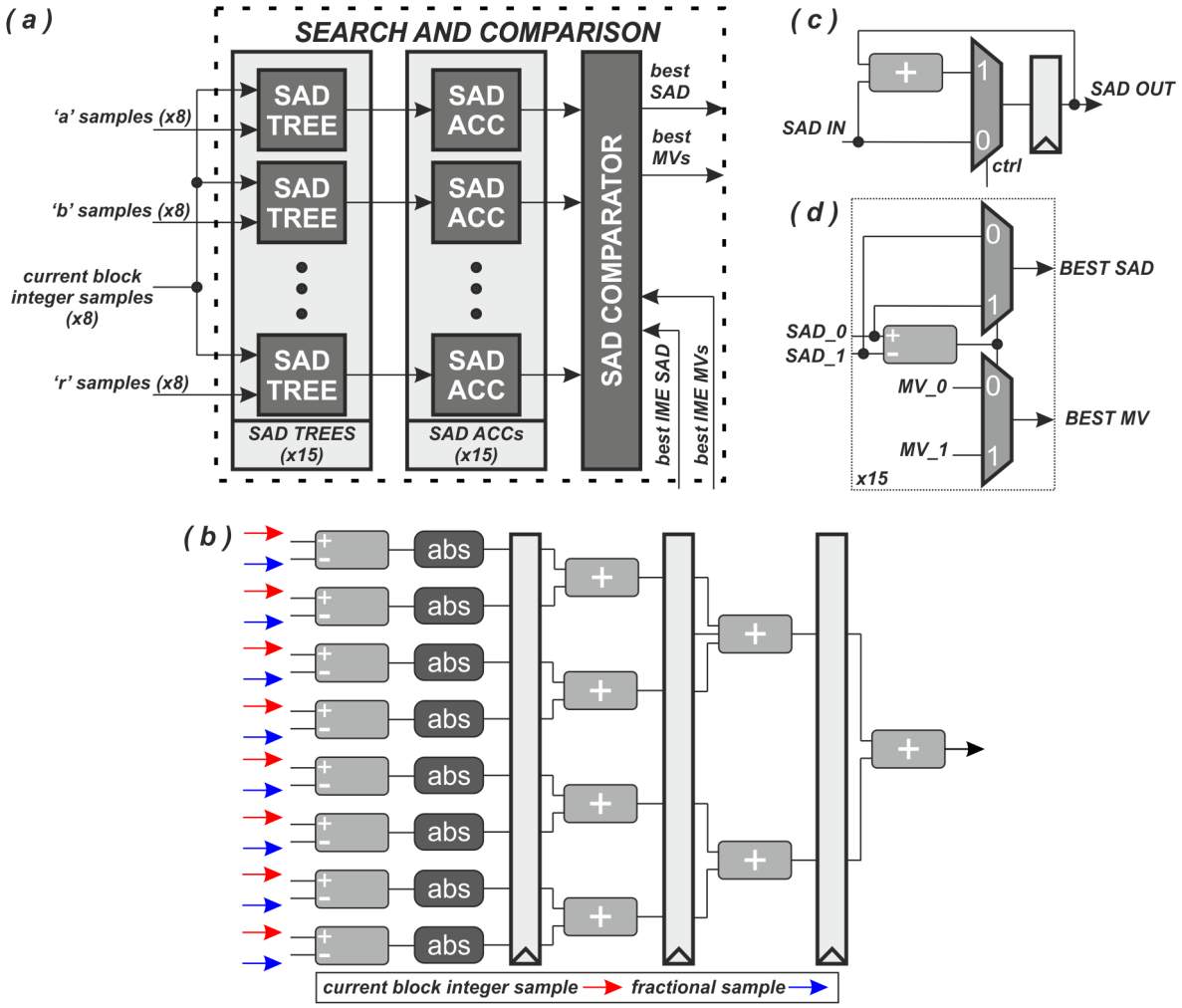

Figure 32 – Search and comparison design: (a) top-level view of SEARCH AND COMPARISON, (b) SAD TREE detailing, (c) SAD ACCUMULATOR detailing, and (d) SAD COMPARATOR detailing.

The temporal behavior when calculating an 8x8 block is quite straightforward, however, when thinking in how to calculate bigger blocks, it is necessary a more careful temporal behavior analysis. Furthermore, since HEVC FME hardware design is our

case study, the knowledge about its behavior is crucial. The case study will consider the processing of 64x64 blocks. In order to calculate the corresponding fractional samples and determine the best SAD and MVs, we use the design focusing in 8x8 blocks, presented in the previous section, in a serial way.

The processing of the 64x64 is split into 8x8 blocks. In order to help this understanding, we present in Figure 33 the temporal diagram for FAPP0 or FAPP2 (worst cases regarding time since more samples need to be evaluated), detailing the behaviors. In Figure 33 (a) we show the necessary clock cycles to process a single 8x8 block. First, there is the latency to fulfill the interpolator register chains (8 cycles), at this point, fractional samples start to be delivered in the output. Thus we have the latency of the SAD TREES (3 cycles), when the first partial SADs start to be stored at the SAD accumulators. From the point it starts, the interpolator takes 15 cycles to calculate all fractional samples. However, the other steps start their operation in parallel, meanwhile the interpolation is happening. After the interpolator ends, the SAD TREEs keep operating for 3 cycles whereas the SAD ACCUMULATOR takes one cycle after SAD TREEs have ended. The SADs are then compared at SAD COMPARATOR, which in turn spends 3 cycles. Therefore, the processing of an 8x8 block in our HEVC FME design takes 22 cycles.

In order to evaluate a 64x64 block, let's consider the partial analysis, presented in Figure 33 (b). In this case, we consider the evaluation of a a block column, i.e., 64x8. Look that when the calculation of an 8x8 finishes, samples already calculated can be used. Therefore, all the column of input matrix (detailed in the figure) is processed. It uses 75 clock cycles, however, when the entire block is being processed, since the samples are still being fed, there is an overlapping of cycles, which in practice means that for each column 71 cycles are necessary, as can be seen in Figure 33 (c).

Such approach can be applied to the other parts of the block, divided into eight columns, as presented in Figure 33 (c). Each of them is processed sequentially, storing the partial SAD values. Finally, they are compared in SAD COMPARATOR, spending a total of 575 cycles. A similar analysis can be applied for FAPP1 and FAPP3 temporal behavior. The only changes are related to the number of samples that need to be read, 69x69 and 65x65 integer samples for FAPP1 and FAPP3, respectively. Therefore, the processing of a 64x64 block with FAPP1 would take 559 cycles and with FAPP3 would take 527 cycles.

5.3 Synthesis and QoS Results

The developed architectures were described in VHDL and synthesized using a 40 nm TSMC standard cell library with 0.9 V (TSMC, 2020) using the Cadence RTL Compiler tool (CADENCE, 2020). The power results were generated using the default tool

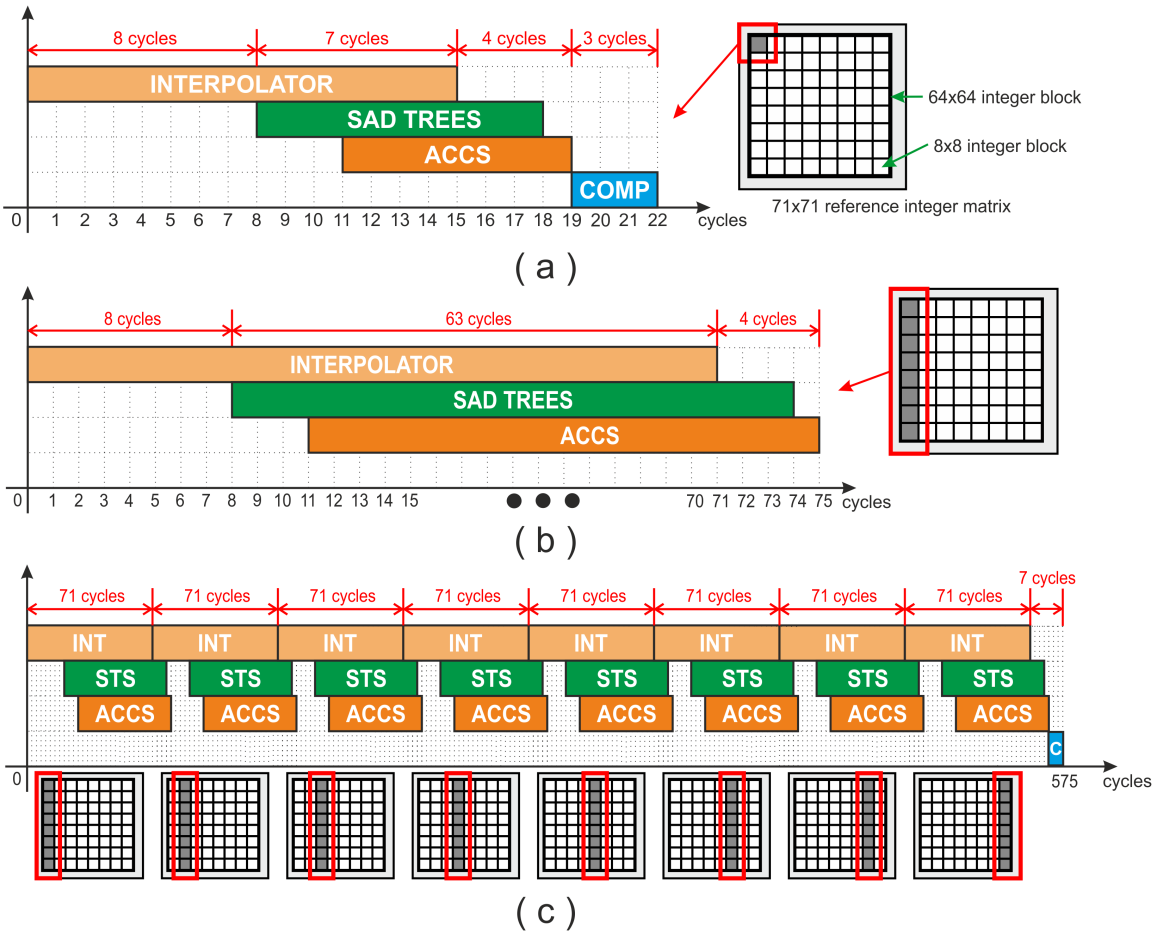

Figure 33 – Temporal diagram of HEVC FME operating with FAPP0 or FAPP2: (a) processing of an 8x8 block, (b) processing of a 64x8 column, and (c) processing of a 64x64 block.

switching activity (20%). The gate count was calculated based on the smallest 2-input NAND (NAND2x1) that represents $0.9408\ (\mu m)^2$ for TSMC (note that even for different technologies the gate count can be fairly compared since it is a normalized metric).

The designed HEVC FME architectures (FAPPs) were evaluated by considering the area and power dissipation results and by focusing on different performances: FHD 1080p@30fps; FHD 1080p@60fps; UHD 4K 2160p@30fps; UHD 4K 2160p@60fps; UHD 4K 2160@120fps; UHD 8K 4320p@60fps; and UHD 8K 4320p@120fps - (8K) resolution (7680x4320 pixels). The minimum throughput (1080p@30fps) was considered in order to guide the design of the processing elements (PEs) of our SApp-NoC solution. In this case, higher throughput is achieved by SApp-NoC by exploiting the inherent parallelism provided. The other throughput are considered to allow a comparison with related works.

From each given scenario we derive the minimum operational frequency in order guarantee the required global throughput (resolution@frame rate), determined by the

relation stated in (35).

$$\frac{Wres \cdot Hres \cdot frame\_rate}{throughput_{FAPP_i} \cdot frequency} = 1 \quad (35)$$

In (35), $Wres \times Hres$ represents the spatial video resolution and $throughput_{FAPP_i}$ represents the number of delivered samples per cycle from $FAPP_i$ interpolator. In our design, $throughput_{FAPP_{02}} = 4.27\ samples/cycle$, $throughput_{FAPP_1} = 4.92 samples/cycle$, and $throughput_{FAPP_3} = 7.11 samples/cycle$. Note that these values consider the processing of the video using only blocks 8x8 (Basic Processing Unit), which is the worst case in terms or processing time since in this case the initial latency presents more prominence face the total time spent. However, a frequency guaranteeing the global throughput for the worst case scenario also guarantees the performance for other scenarios.

In Table 4 we present the synthesis results regarding Application Specific Integrated Circuit (ASIC) solution, targeting a 40 nm technology. In this table we show the synthesis results for all proposed designs: FAPP0 (precise), FAPP1 (approximate), FAPP2 (approximate), and FAPP3 (approximate). The results were generated aiming seven global throughput, ranging from FHD@30fps to 8K@120fps. For each scenario we show the corresponding operational frequency, the static power (*Ps*), the dynamic power (*Pd*), and the total power (*Pt*), which is the sum of *Ps* and *Pd*. Furthermore, area in terms of gate count is also provided.

As expected, FAPP0 presents the highest area and power dissipation values. According to the applied level of approximation, these values are significantly reduced when compared with the precise design. The power savings are present in Table 5. Note that the static power does not present significant changes for different throughput, since it is related to the size of the architecture. The dynamic power increases with the throughput once more intense switching activity is demanded.

In Table 5 we present the power savings provided by each FAPP, for each throughput scenario. In general, the behavior regarding the power savings is quite constant. FAPP1, which provides the less aggressive approximation, delivers an average power saving of 17.025%. FAPP2, a medium case of approximation, delivers an average power saving of 34.970%. The more aggressive approximation solution, FAPP3, provides an average power saving of 64.509%. Moreover, the approximate designs FAPP1, FAPP2, and FAPP3 reduce area in 7.11%, 31.54%, and 56.74%, respectively.

The impact of the proposed approximate solutions in terms of QoS (coding efficiency) were evaluated using the reference software of the HEVC, the HM 16.18 (HEVC Test Model) (BOYCE, 2014), according to the Common Test Conditions (CTCs) (BOSSEN, 2013), established by the HEVC standard. Our simulations took into account 200 frames from each of the 24 recommended video sequences and recom-

Table 4 – Synthesis results for ASIC TSMC 40 nm

Design	*Throughput*	*Freq. (MHz)*	*Ps (mW)*	*Pd (mW)*	*Pt (mW)*	*Gate Count (K)*
FAPP0	*FHD@30fps*	14.570	8.557	6.501	15.058	167.66
	FHD@60fps	29.140	8.557	13.446	22.003	
	4K@30fps	58.220	8.577	25.673	34.250	
	4K@60fps	116.550	8.570	48.617	57.187	
	4K@120fps	233.100	8.560	82.849	91.409	
	8K@60fps	466.200	8.559	139.090	147.649	
	8K@120fps	932.400	8.746	214.217	222.963	
FAPP1	*FHD@30fps*	12.644	7.904	4.572	12.476	155.73
	FHD@60fps	25.288	7.936	10.142	18.078	
	4K@30fps	50.575	7.921	20.438	28.359	
	4K@60fps	101.150	7.935	38.993	46.928	
	4K@120fps	202.300	7.932	69.806	77.738	
	8K@60fps	404.600	7.919	115.130	123.049	
	8K@120fps	809.200	7.718	176.379	184.097	
FAPP2	*FHD@30fps*	14,570	5.650	3.605	9.255	114.78
	FHD@60fps	29.140	5.641	9.303	14.944	
	4K@30fps	58.220	5.662	17.787	23.449	
	4K@60fps	116.550	5.673	32.660	38.333	
	4K@120fps	233.100	5.688	53.391	59.079	
	8K@60fps	466.200	5.684	86.851	92.535	
	8K@120fps	932.400	5.511	135.020	140.531	
FAPP3	*FHD@30fps*	8.750	3.658	2.366	6.024	72.53
	FHD@60fps	17.500	3.641	3.334	6.975	
	4K@30fps	35.000	3.610	8.624	12.234	
	4K@60fps	70.000	3.633	14.809	18.442	
	4K@120fps	140.000	3.621	26.400	30.021	
	8K@60fps	280.000	3.636	52.236	55.872	
	8K@120fps	560.000	3.627	81.279	84.906	

mended Quantization Parameter (QP) – 22, 27, 32, and 37. The coding efficiency (BD-BR) results are present in Table 6 for each FAPP and video sequence. Note that the classes and resolutions of each sequence, established by the HEVC CTC, are also shown in Table 6.

The QoS of approximate solutions presented the same behavior of the power savings. The higher the approximation level, the higher the QoS degradation. Indeed, such a behavior is expected, since we are simplifying the process when comparing with the precise solution. Taking into account the 24 video sequences, in average, FAPP1 presents 0.5591%, FAPP2 presents 1.0746%, and FAPP3 presents 7.2607% of BD-BR increase. In a simple view, the growth on power savings leads to losses on QoS, showing the necessity of an adaptive solution which we explore in this thesis.

Regarding specific cases, some videos require a more detailed analysis. On the one hand, the sequence *BQTerrace* presented the highest values, thus presenting higher QoS losses than the others. It happens for this video due its nature, since it clas-

Table 5 – Average power savings of approximate solutions (FAPP1-3) compared with precise solution (FAPP0)

Design	***Throughput***	***Power Saving (%)***	***Avg. Power Saving (%)***
FAPP1	*FHD@30fps*	17.147	17.025
	FHD@60fps	17.838	
	4K@30fps	17.200	
	4K@60fps	17.939	
	4K@120fps	14.956	
	8K@60fps	16.661	
	8K@120fps	17.432	
FAPP2	*FHD@30fps*	38.538	34.970
	FHD@60fps	32.082	
	4K@30fps	31.536	
	4K@60fps	32.969	
	4K@120fps	35.369	
	8K@60fps	37.328	
	8K@120fps	36.971	
FAPP3	*FHD@30fps*	59.995	64.509
	FHD@60fps	68.300	
	4K@30fps	64.280	
	4K@60fps	67.751	
	4K@120fps	67.158	
	8K@60fps	62.159	
	8K@120fps	61.919	

sified as a high complexity sequence (HCS) in terms of Temporal Index (TI) and Spatial Index (SI) (CASSA; NACCARI; PEREIRA, 2012). Therefore, even small changes on the encoding process can lead to huge quality losses. On the other hand, the sequence *SlideEditing* presented QoS improvement for all FAPPs. In spite of this non-expected result, such a behavior can be explained. This sequence consists of slides presentation being edited, so the background scene remains unchanged for considerable time, which diminishes the impacts of the FME step. For this reason, even FAPP3 solution did not introduce any QoS loss.

A comparison with related works regarding synthesis results, power savings and QoS results is presented in Chapter 8.

Table 6 – QoS results in terms of BD-BR (%) for each FAPP

Class / Resolution	***Sequence***	***FAPP1***	***FAPP2***	***FAPP3***
Class A / 2560x1600	*Traffic*	1.1302	2.1284	13.1047
	PeopleOnStreet	0.2609	0.6756	2.5497
	Nebuta	0.1102	0.6605	2.4316
	SteamLocomotive	0.9605	1.7301	13.2132
Class B / 1920x1080	*Kimono*	0.4477	0.7670	2.9604
	ParkScene	0.8993	1.8404	11.7870
	Cactus	0.3161	0.8166	3.1022
	BQTerrace	2.4352	3.3906	26.6549
	BasketballDrive	0.7345	1.2162	8.7508
Class C / 832x480	*RaceHorsesC*	0.5648	1.1512	9.2376
	BQMall	0.7588	1.1705	7.4362
	PartyScene	0.3498	0.2294	5.5861
	BasketballDrill	0.0835	1.0932	12.2990
Class D / 416x240	*RaceHorses*	0.5192	1.1018	5.7615
	BQSquare	1.0756	0.5914	6.3794
	BlowingBubbles	1.0030	1.1522	8.2223
	BasketballPass	0.4868	0.9432	5.0366
Class E / 1280x720	*FourPeople*	0.1828	0.5003	2.2968
	Johnny	0.8822	1.7669	6.5710
	KristenAndSara	0.3704	0.9022	3.2758
Class F / 1280x720	*BasketballDrillText*	0.3369	1.0562	12.2525
	ChinaSpeed	0.0940	0.2031	0.7612
	SlideEditing	-0.0053	-0.0183	-0.0016
	SlideShow	0.3806	0.8212	3.8887
Average		0.5591	1.0746	7.2607

6 NOC DESIGN AND APPLICATION MODELING

In order to observe to behavior of SApp-NoC when performing the execution of the target application, we have modeled SApp-NoC architecture and the application, according to the background provided in Chapter 2. Following a high level of abstraction, we are able to model the distribution of application tasks over the PEs, analysing the flows across the NoC and the system schedulability, as follows.

6.1 NoC Architecture

The NoC platform follows the architecture described in Chapter 2, with heterogeneous processing elements (GPP, FAPP0, FAPP1, FAPP2, and FAPP3), running priority-preemptive task schedulers, distributed memory, 2D-mesh NoC interconnect with XY dimension routing (3x3, 4x4, and 5x5 in this work), credit-based flow control, 8 virtual channels with 3-flit input buffers per port and priority-preemptive link arbitration.

In our approach, we considered that there is always enough memory space on a given PE. The operational frequency of GPPs was set in 3.0 GHz, exclusively to run memory related tasks. The frequency of FAPPs is based on their hardware synthesis, and it is discussed in the next section. The frequency of the NoC communicating flows can assume a wide range of values (SAYUTI; INDRUSIAK, 2015). In our solution, the definition of this frequency is defined by a proposed algorithm, presented in Section 6.3.

6.2 Application Workload Modeling

We have modeled the application (HEVC FME) in high-level abstraction as a Sporadic Task-Model, as introduced in Chapter 2, and we have considered the processing of only 64x64 blocks. Each step of HEVC FME was considered as one single task: interpolation (*Int*) and search and comparison (*S&C*) calculation, as presented in Figure 34, but for purpose of modeling they are clustered into a FME task. Furthermore, the first and last tasks of the chain are memory-related tasks: *MI* (Memory Input) and *MO* (Memory Output), respectively.

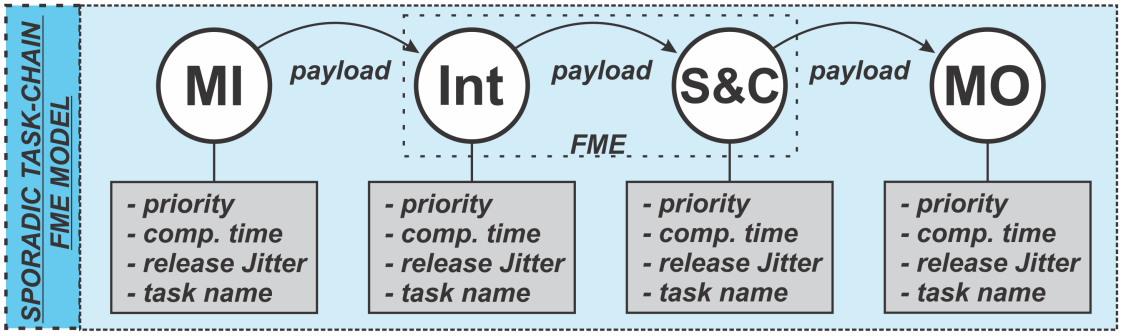

Figure 34 – FME modeled as a Sporadic Task-Chain.

Each task has some characteristics assigned, which will guide its behavior: *priority*, *computation time*, *release jitter*, and a *task name*. In this work, different priorities are assigned to each task, in a crescent order, according to the task creation order. For example, to the first task is given a priority 1, to the second task is given a priority 2, and so on (the value 1 denotes the highest priority and larger integers denote lower priorities). Note that flows have the same priority assigned to its source task. The computation time informs the time the task takes to be executed. The release jitter of tasks are assumed equals to zero, i.e., we consider that any task starts to be executed at the same time it is requested. The task name is a tag, guaranteeing that each task will give a different name in order to be easily identified.

The first task of model presented in Figure 34 represents the modeling of a block being read from off-chip memory until reaches the *Int*. The final task is an empty set, treated as a sink (i.e., computation time equals to zero), where the necessary information is stored for the forward steps of the encoder. The payload, given in flits, are packets containing information about the data being processed, having the size given according to the block size (worst case 64x64 was considered in this work), NoC flit size (depends on NoC topology – 64 bits in this work), and bit word size (8 bits for FHD, 10 bits for UHD). In (36) we show the equation of payload between a memory read (MI task) and the FME task.

$$payload = \frac{Weigth_{ref\_matrix} \times Height_{ref\_matrix} \times bit\_word_{size}}{NoC_{flit\_size}} + 2 \qquad (36)$$

The payload between MI and Int considers the data information, which are the samples of the reference matrix (71x71 for FAPP0 and FAPP2, 69x69 for FAPP1, and 65x65 for FAPP3) multiplied by the bit word size, representing the total data in bits. When dividing by the NoC flit size such an information if then represented in flits. Finally, two flits are added, which are used for the header of the packet. The payload between the FME task and MO needs to embrace the best SAD and best vector information, which, according to our hardware design, requires 21 bits. Thus for this payload we consider the allocation of three flits (one for data and two from the header).

Table 7 – FME memory reads computation time evaluation of 64x64 blocks

Sequence	***QP***	***Mean (ns)***	***Upper Quartile (ns)***
Traffic	*22*	542	553
	27	549	561
	32	550	561
	37	548	561
Kimono	*22*	578	581
	27	586	587
	32	581	585
	37	579	579
RaceHorsesC	*22*	581	582
	27	586	586
	32	560	557
	37	576	577
BlowingBubbles	*22*	558	562
	27	561	572
	32	576	586
	37	580	590
Johnny	*22*	593	608
	27	595	610
	32	590	606
	37	593	607
ChinaSpeed	*22*	600	607
	27	592	600
	32	589	596
	37	592	599
Average		**576.458**	**583.875**

To model *MI* we have analyzed FME step using the reference software of the HEVC, the HM 16.18 (HEVC Test Model) (BOYCE, 2014). We have encoded 64 frames from different video sequences, one from each class - *Traffic*, *Kimono*, *RaceHorsesC*, *BlowingBubbles*, *Johnny*, and *ChinaSpeed* - presenting different resolutions and frame rates, as recommended by the video community benchmarks, in the Common Test Conditions (CTCs) document (BOSSEN, 2013), for each recommended QP (22, 27, 32, and 37), running isolated into an *i7* core with a fixed frequency of 3.0 GHz. The obtained computation times regarding the reading of a block 64x64 during HEVC FME step are present in Table 7. In this table, we show the average computation time (mean in nanoseconds) and the upper quartiles, also expressed in nanoseconds, of the statistical distribution, for each sequence and QP. For the modeling of MI we have considered the average among all upper quartiles, thus applying 583.875 ns as computation time for MI during the application modeling.

The modeling of the FME task follows results from hardware synthesis. The full processing of a 64x64 at FME takes 575 cycles at FAPP0 or FAPP2 (see Figure 33),

559 cycles at FAPP1, and 527 at FAPP3. Therefore, the computation time of FME is given in (37) as follows:

$$comp\_time_{FME} = number\_of\_cycles \times clock\_period \quad (37)$$

In this thesis we adopted the smallest global throughput (FHD@30fps) as reference to design the processing elements. Note that we adopted the same frequency for all PEs in order to have the same clock at all PEs within SApp-NoC, differently from the individual FME designs (where we applied different frequencies according to the level of approximation desired). Thus, since the PE frequency is 14.57 MHz (see Table 4), its period is equals to 68.63 ns, which leads to a computation time at FME, when processing a 64x64 block, of 39.46 μ s when executing FAPP0 or FAPP2, .

6.3 Schedulability breakdown NoC Frequency Tracking algorithm (SNFT)

As previously mentioned, the schedulability of a given NoC-based system can be defined by the analysis of its tasks and flows. The system will be fully schedulable only if the both tasks and flows meet their deadlines (i.e., they are schedulable).

A given application mapped onto a NoC can be fully schedulable or not depending on the NoC frequency. For the same configuration, presenting schedulable tasks, only changing the NoC frequency, a previously unschedulable system can become schedulable. In order to find out the schedulability breakdown frequency, i.e., the minimum frequency that makes the system fully schedulable, we developed the Schedulability breakdown NoC Frequency Tracking algorithm (SNFT) based in (SAYUTI; INDRUSIAK, 2015), presented in Figure 35.

Firstly, it is defined an initial NoC frequency f_0. The application model is created and then the other steps happen inside a loop. An auxiliary variable *test_schedulability* is created to control the loop in order to vefify the system schedulabiity. The application is first mapped onto a NoC, next the schedulability test is performed. Note that to begin SNFT algorithm analysis all tasks need to be schedulable, otherwise there is no breakdown frequency capable of turn the system schedulable. The definition criteria for the next NoC frequency value is different only for the first iteration, being the same for the following steps, and it is based on the frequency values and on the information whether the system is fully schedulable or not, according to Figure 35. The stopping criteria is based on a preset value, establishing an accuracy of 1 MHz. In addition, if even reaching a frequency of 1.2 GHz the system still unschedulable, the loop finishes and, in such a specific situation, the schedulability breakdown NoC frequency is not found.

Inputs: *Application Workload*, *NoC Characteristics*
Outputs: *Schedulability Results*, *Breakdown NoC Frequency*

define f_0, *test_schedulability = 0*, *i = 0*
create the application model
while *test_schedulability = 0* **do**
  map the application over the NoC
  perform schedulability analysis
  if *i = 0* **then**
    if *system is fully schedulable* **then**
      f_{i+1} = f_i - f_i/2
      Δ_f = f_i - f_{i+1}
    else
      f_{i+1} = f_i + f_i/2
      Δ_f = f_{i+1} - f_i
    end if
  else
    if *system is fully schedulable* **then**
      f_{i+1} = f_i - Δ_f/2
      Δ_f = f_i - f_{i+1}
    else
      f_{i+1} = f_i + Δ_f
      Δ_f = f_{i+1} - f_i
    end if
  end if
  if $\Delta_f \leq 10^6$ **or** $f_{i+1} \geq 1.2 \cdot 10^9$ **then**
    test_schedulability = 1
  end if
  i++
end while

Figure 35 – SNFT pseudo-algorithm

7 APPLICATION-AWARE APPROXIMATION CONTROL

Instead of applying approximate solutions for the whole frame, as previously discussed in Chapter 2, we propose applying the approximation only on suitable regions of the frame, i.e., homogeneous regions, in order to keep the video quality whereas reducing the FME power dissipation. The main challenge is to determine these regions beforehand, thus applying the most suitable approximation level at FME. The monitoring of other encoding parameters, measured before FME being executed, can give a trust clue about the current block (homogeneous or not), and their combination can provide a simple decision tree to allow the approximation level prediction. In order to perform the selection of the approximation level, we propose in this thesis two solutions: a heuristic-based solution (HSApp-NoC) and a machine learning-based solution (MLSApp-NoC).

7.1 Heuristic-based Application-aware Approximation Control

Our Heuristic-based Application-aware Approximation Control (HAAC), which compounds the HSApp-NoC solution, leverages video content properties to reduce energy consumption by selecting the approximation level whereas reducing quality degradation. For that, HAAC exploits the application knowledge and considers the TZS behavior during the IME process as a descriptor for the video content. Considering that, generally, the more complex (texture/motion) the data, the more steps/effort are necessary, the TZS behavior may be a good descriptor of frame properties, a fast TZS convergence gives a clue that the block being coded is likely to be homogeneous, thus, being suitable for approximation due to error-resiliency nature, as stated in Chapter 2. On the other hand, when TZS has a hard-to-match block, we assume it is likely to be a heterogeneous block, thus requiring more precise processing (also refer to the discussion in Chapter 2).

Let's define *acc_id* as the parameter which decides the FAPP to process a given block, when *acc_id = 0* FAPP0 must be selected, when *acc_id = 1* FAPP1 must be selected and so on. For HAAC we have adopted as heuristic the behavior of the param-

eter *tzs_step*, which informs the convergence behavior at IME during TZS execution. HAAC is presented in Figure 36.

```
1:  determine acc_id (tzs_step)
2:  if tzs_step = 0 then
3:      acc_id = 3
4:  else if tzs_step = 1 then
5:      acc_id = 2
6:  else if tzs_step = 2 then
7:      acc_id = 1
8:  else if tzs_step = 3 then
9:      acc_id = 0
10: end if
11: return acc_id
```

Figure 36 – Heuristic-based application-aware approximation control pseudo-code.

In cases where the best matching is found in the **first search** step with *two or less* iterations, *tzs_step = 0*, then FAPP3 must be selected (*acc_id = 3*). When the best matching is found in the **first search** step with *more than two* iterations, *tzs_step = 1*, then FAPP2 must be selected (*acc_id = 2*). In cases where the best matching is found in the **refinement** step, *tzs_step = 2*, and, in this case, FAPP1 must be selected (*acc_id = 1*). Finally, for cases where IME search was all the way to the **raster** step, *tzs_step = 3* and FAPP0 must be selected (*acc_id = 0*). The QoS results of this solution are presented in Chapter 8.

7.2 Machine Learning-based Application-aware Approximation Control

In order to improve the QoS results, we propose a deeper analysis of the application behavior to find the most suitable regions within the frame for the employment of approximate computing. Thus, we have developed a decision tree to select the approximation level for each block being processed by the FME, based on machine learning, so called Machine Learning-based Application-aware Approximation Control (MLAAC).

7.2.1 Evaluated Parameters

Some works in the literature (HE et al., 2015; PODDER; PAUL; MURSHED, 2016; MAUNG; ARAMVITH; MIYANAGA, 2016; SILVA; SIQUEIRA; GRELLERT, 2019; MERCAT et al., 2019; LU et al., 2020) give hints of which parameters could affect the FME behavior. The evaluated parameters are presented in Table 8, we choose to evaluate seven parameters, which have the highest probability of affect FME behavior. Besides common parameters like video resolution and quantization parameter. We have also

considered *merge_flag* (informs if the block selected at IME belongs to the merge list candidates - *merge_flag = 1*), *skip_flag* (informs if the block selected at IME is skip - *skip_flag = 1*), *neigh_est* (informs if the neighbor block, previously encoded, has selected IME or FME – 0,1 – Figure 37 details *neigh_est* parameter), and *tzs_step* which was detailed in the previous section. Parameter *acc_id* is the one to be predicted, it defines the approximation level to be applied at FME, thus indicating the FAPP index (0, 1, 2, or 3). Parameter *skip* informs that no movement or residual information were transmitted during inter-prediction, in this case, the motion information is derived from spatial and neighbor blocks. Parameter *merge* consists in deriving the motion parameters from other PU which is based on spatial or temporal neighbors (note that a *skip* is always a *merge*, but a *merge* might not be a *skip*).

Table 8 – Evaluated Parameters

Parameter	***Description***	***Parameter Values***
Wres	*Weight Resolution*	*Integer*
Hres	*Height Resolution*	*Integer*
QP	*Quantization Parameter*	*Integer*
merge_flag	*Merge mode information*	*{0, 1}*
skip_flag	*Skip mode information*	*{0, 1}*
neigh_est	*Neighbour Estimation*	*{0, 1}*
tzs_step	*Convergence of TZS during IME*	*{0, 1, 2, 3}*
acc_id	*Parameter to predict - FAPP selection*	*{0, 1, 2, 3}*

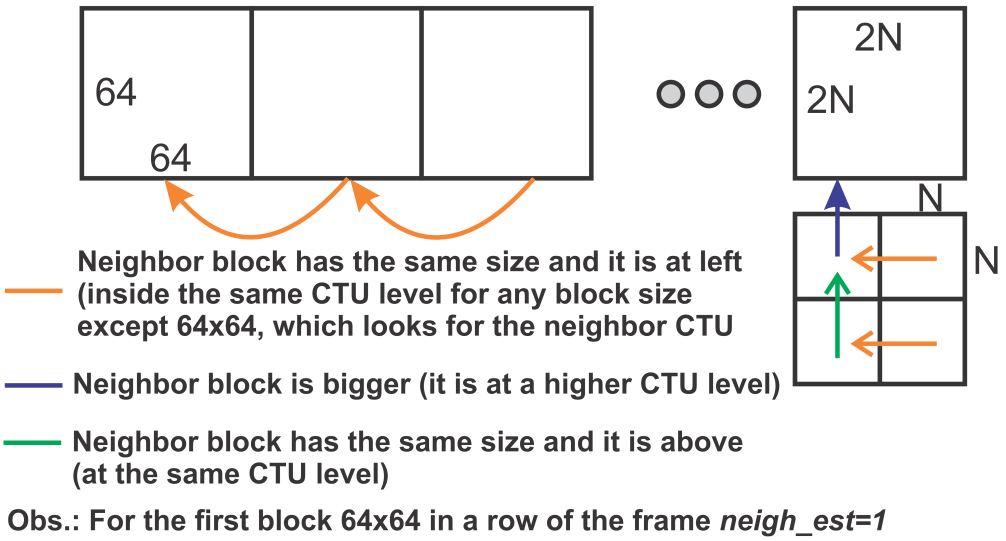

Figure 37 – Neighbour Estimation Parameter.

7.2.2 Evaluation of Normalized RDCosts

The application of supervised Machine Learning algorithms implies the need of feeding the algorithm the best selection for each situation. Thus, in order to allow the application of such techniques, a method determining the best selection needs to be provided. We decided to apply a selection criterion based on the normal distribution of normalized RDCosts for each proposed scenario (FAPP0, FAPP1, FAPP2, and FAPP3).

First, we perform all traditional encoding steps until it reaches the FME. At this point we make a branch, performing all proposed scenarios (precise + approximate) and evaluating their RDCosts. The normalized RDCost ($normRD_i$) of each $FAPP_i$ is then calculated according to (38).

$$normRD_i = RDCostFAPP_i / RDCostFAPP_0 \tag{38}$$

These values are stored for further offline analysis. We have evaluated one video sequence per class of HEVC Common Test Conditions (CTCs) (BOSSEN, 2013): *Traffic*, *BQTerrace*, *PartyScene*, *RaceHorses*, *Johnny*, and *ChinaSpeed*, for all recommended QPs (22, 27, 32, and 37), considering only square-shaped blocks (64x64, 32x32, 16x16, and 8x8), 200 frames (150 for *Traffic* since it is its maximum size), only one reference frame, no bi-prediction, and Random Access (RA) configuration. The measured values resulted in four Gaussian distributions, present in Figure 38, for each recommended QP, informing the Probability Density Function (PDF) values for each scenario.

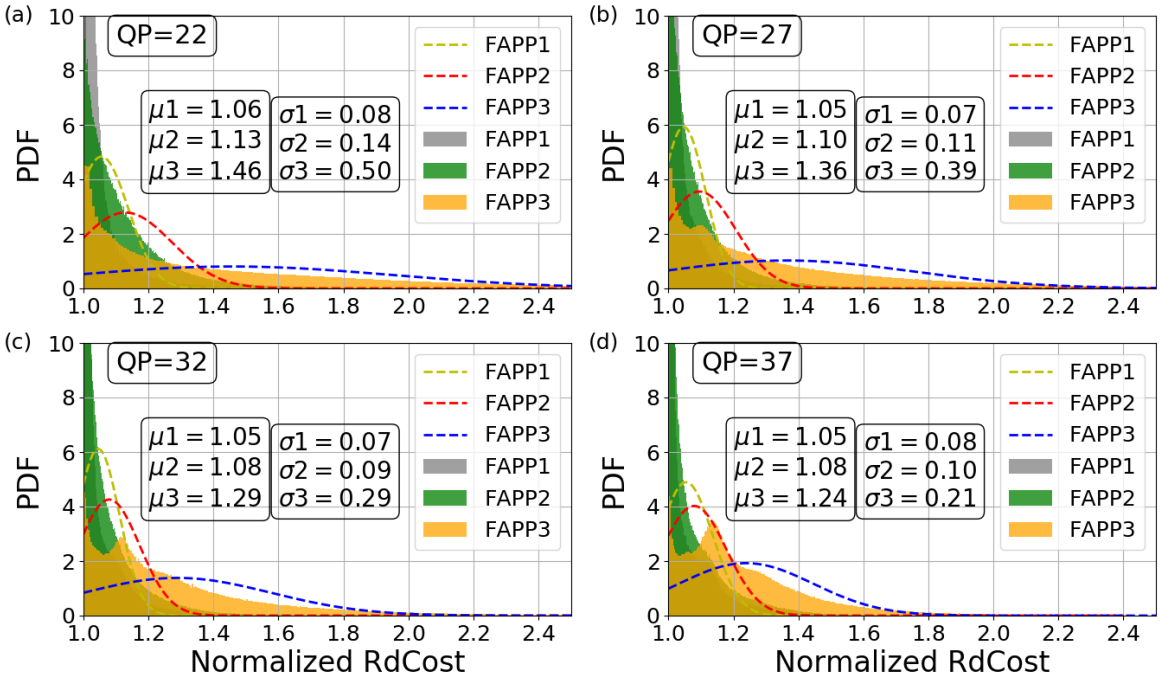

Figure 38 – Gaussian distributions for QPs (a) 22, (b) 27, (c) 32, and (d) 37.

The average and standard deviation values are present in each graphic. The normal distribution curves are plotted in yellow, red, and blue, representing FAPP1, FAPP2, and FAPP3, respectively. Consider the PDF function $f(x)$ defined as follows:

$$f(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{39}$$

In (39) and Figure 38, μ and σ are, respectively, the average and the standard deviation of the statistic distribution. Let consider two PDF functions $f_1(x)$ and $f_2(x)$, the intersection between these curves is found when $f_1(x) = f_2(x)$. The solution of this approach returns a quadratic function:

$$Ax^2 + Bx + C = 0 \tag{40}$$

where

$$A = \frac{1}{2\sigma_2^2} - \frac{1}{2\sigma_1^2} \tag{41}$$

$$B = \frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2} \tag{42}$$

$$C = \frac{\mu_2}{2\sigma_2^2} - \frac{\mu_1}{2\sigma_1^2} + ln\left(\frac{\sigma_2}{\sigma_1}\right) \tag{43}$$

The roots of (40) return the intersection points between two Gaussian functions. The finding of these roots is mandatory in our analysis since the approach we apply in this work depends on these intersection points.

Our criterion considers the intersection points among the curves as control points: (**P1**) FAPP1-2, (**P2**) FAPP1-3, and (**P3**) FAPP2-3. The algorithmic decision is quite simple: for each QP, for each encoding scenario, the normalized RDCosts of all proposed FAPPs are evaluated and then compared. If any of their values are smaller than 1.0 (not showed in the graphics), then such FAPP must be chosen, if more than one FAPP presents this result the FAPP providing higher power saving must be chosen (i.e. FAPP3 $\rightarrow$2$\rightarrow$1). If any of their values are between 1.0 and **P1** (inclusive), then the selected FAPP must be the one providing higher power saving; if none of them is at this interval, then the same evaluation is performed at **P1**-**P2** interval and next at **P2**-**P3**. If no FAPP normalized RDCost values are found at these intervals, so FAPP0 (precise) must be selected. This criterion selection resulted in an average QoS in terms of BD-BR of only 1.15% when compared with precise encoding (FAPP0), with a resulting average FAPP selection of 4.064%, for FAPP0; 18.056%, for FAPP1; 33.411%, for FAPP2; and 44.469%, for FAPP3.

Obviously, such approach is unpractical in real scenarios since it requires the pro-

cessing of the FME five times (once for each FAPP and then for the selected FAPP again). In addition, a power analysis also does not make sense for this approach, due to the enormous overhead that is inserted. However, such approach can guide the development of machine learning-based algorithms in order to make those decisions feasible, like the ones based on decision trees, as will be discussed in the following section.

7.2.3 Generated Decision Tree

The data collected for training considered only ten frames of sequences with higher resolutions (classes A and B) - *Traffic* and *BQTerrace* – a low complexity sequence (LCS) and a high complexity sequence (HCS), respectively, providing a wide range of motion and texture in terms of Temporal Index (TI) and Spatial Index (SI) (CASSA; NACCARI; PEREIRA, 2012), whereas avoiding over-fitting to the data set. Considering the prohibitive amount of data to handle if the full sequences were analyzed, we have made the assumption that ten frames are enough to give a clue about the video behavior in order to build the model. Furthermore, considering the configurations adopted for SApp-NoC, only blocks 64x64 were considered. The evaluated parameters were stored for each FAPP selection.

These collected values were analyzed using the Waikato Environment for Knowledge Analysis (WEKA), which is a free, open-source machine learning tool (HALL et al., 2009). The input for WEKA are the ARFF (Attribute-Relation File Format) files, which contain a plain text describing a list of instances sharing different attributes (in our case the selected encoding parameters) and the class (variable to be predicted). In Figure 39 we show the structure of an ARFF file, using as example the ARFF file employed by this work. It presents two major regions, separated by dashed boxes. The first one is a header section, presenting the name of the relation (class) and the attributes declaration (i.e., name and type of values of that attribute). In the second section it is presented the training data, containing one instance per row and one attribute value per column. When considering the building of decision trees, the last line of the first section informs the class attribute, i.e., the variable for which the ML algorithms will try to find a general (prediction) rule.

We evaluated the relevance of each parameter applying the Information Gain Attribute Evaluation (IGAE) method, which measures the information gain (IG), based on the Kullback-Leibler Divergence (KLD), also known as relative entropy, which measures the divergence between two probability distributions. Therefore, IG indicates how relevant each parameter is for constructing a decision tree model. In our case, it decides which is the level of approximation that must be applied during FME, resulting in the selection of FAPP0, 1, 2, or 3.

The definition of the attributes with more information gain allowed the construction

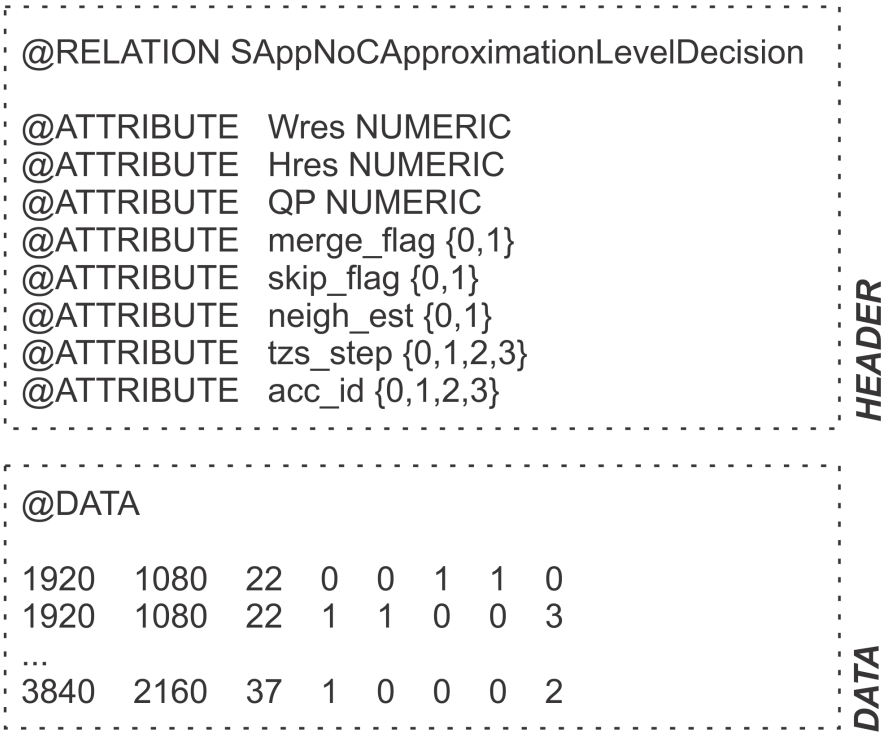

Figure 39 – Example of the structure of an ARFF file.

of the decision tree model for our adaptive FME approximation. The training process considers the application of the C4.5 algorithm, implemented in WEKA framework as the J48 tool. The default configuration was used, considering 0.25 as a confidence factor (used for pruning the trees – the less the value the more aggressive is the pruning). The collected information from the selected video sequences was used for training, evaluating more than 13,000 instances (after a class balancing, since some FAPPs were more selected than the others, which required a re-sampling from the original obtained ARFF file). Firstly, we removed the resolution from the analysis, in order to build a simpler tree (we observed that the generated trees, when these parameters were present, became very deep and complex). Next, C4.5 starts obtaining all input instances, then calculating the IG of each attribute in order to perform the classification (QUINLAN, 2014), which are present in Table 9.

Table 9 – Information Gain Attribute Evaluation for SApp-NoC Decision Tree

Attribute	***IG***
QP	0.06129
merge_flag	0.01039
skip_flag	0.00881
neigh_est	0
tzs_step	0.06324

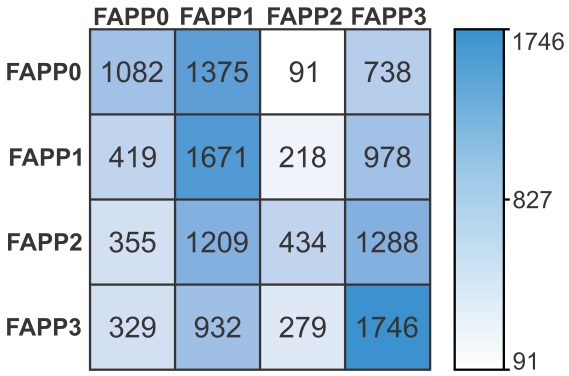

Figure 40 – Confusion matrix of MLAAC decision tree.

After the best attribute selection, it is used to divide the training data into two subsets, being recursively applied to build the tree. Thus, after the training process, the accuracy of the generated trees is observed by applying a 10- fold cross-validation process and observing the percentage of correct decisions, based on the confusion matrix presented in Figure 40, achieving a percentage of 64.33% on correctly classified instances. Note that we have considered any decision which not diminishes the QoS as a correctly classified instance, e.g., if the correct choice would be FAPP3 but FAPP2 is selected instead, we treat as a correct classification, if the correct choice would be FAPP2 and FAPP3 is selected, we treat as an error.

The decision tree for the adaptive selection of the approximation level used at MLSApp-NoC is presented in Figure 41. Note that *tzs_step* is placed at the root since it presents the highest IG. The other selected parameters: *QP*, *merge_flag* and *skip_flag*, guide the further decisions of the tree. Note that they are the ones presenting the highest IGs according to Table 9. These attributes give a reasonable clue about the homogeneity level of the block being coded, allowing improved exploration of power/QoS trade-off.

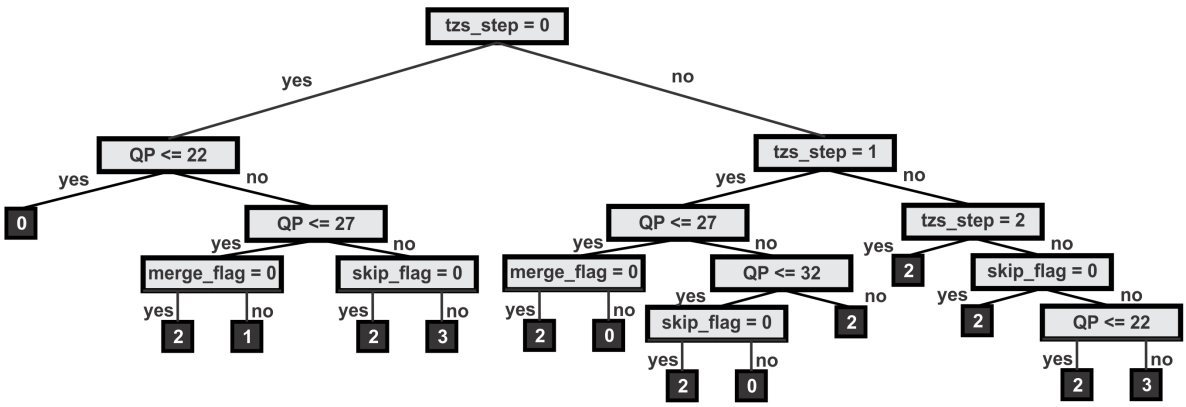

Figure 41 – Generated decision tree - MLAAC

7.2.4 QoS Results for Generated Decision Tree

In order to verify the accuracy of the generated decision tree, we have repeated the same experimental setup presented in the previous section, implementing the decision tree within the inter-prediction code to decide the level of approximation when performing the FME for each block. Note that the overhead inserted by this tree is negligible, since it is a small set of *ifs* and *elses*. However, by this time, we changed the analyzed video sequences in order to provide a different sort of encoding scenarios. We have evaluated one sequence per class of HEVC CTC (BOSSEN, 2013): *PeopleOnStreet*, *Kimono*, *BasketballDrill*, *BlowingBubbles*, *KristenAndSara*, and *SlideEditing*, for all recommended QPs (22, 27, 32, and 37), considering only 64x64 blocks, 200 frames (150 for *PeopleOnStreet*), only one reference frame, no bi-prediction, and Random Access (RA) configuration. The QoS results in terms of BD-BR are present in Table 10. Note that the same encoding scenario was considered for the encoding of the baseline (without any changes, i.e., the precise solution FAPP0).

When analyzing the QoS results, it can be noticed that the results of the generated decision tree are even better than the results obtained with the decisions based on normalized RDCosts. Firstly, the tested sequences are different from the training sequences, which itself is a reasonable reason for different results, since we are analyzing videos with different TI and SI. Secondly, the assumption for using the normalized RDCosts as an arbiter for the control of the level of approximation at FME is an heuristic, which have provided reasonable QoS results but still being a sub-optimize solution. Therefore, other heuristics can provide even better results, like the generated decision tree. Looking the results in Table 10, we can also verify that, for small resolutions videos (like *BasketballDrill* and *BlowingBubbles*), the Qos results are worse, since we are using only 64x64 blocks. For higher resolution videos such an approach impacts less. Furthermore, *SlideEditing* sequence has presented an improvement on QoS. This video is synthetic with a distinct behavior, showing a slide presentation being edited, with a background image not changing for a long time. In this case, the approximation applied did not impact the video QoS.

Table 10 – QoS results in terms of BD-BR for the tested sequences

Sequence	***BD-BR (%)***
PeopleOnStreet	0.585345
Kimono	0.749071
BasketballDrill	1.289618
BlowingBubbles	1.159798
KristenAndSara	0.911402
SlideEditing	-0.018106
Average	0.779521

8 RESULTS AND DISCUSSION

In this thesis, different sort of experiments were performed to enable the whole system modeling and achieve the final results. In this chapter we present the obtained results and a discussion of our main contributions. In Section 8.1 we present the adopted experimental setup. Section 8.2 discusses the obtained results from SNFT algorithm. Next, Section 8.3 discusses the power and energy results of the proposed solutions HSApp-NoC and MLSApp-NoC. In Section 8.4 the obtained QoS results are discussed. Finally, Section 8.5 brings the discussion about the results regarding the performance analysis.

8.1 Experimental Setup

Quality-of-Service results were obtained using the main profile of the HEVC reference software, the HM 16.18 (BOYCE, 2014), according to recommendations of video community (BOSSEN, 2013). Our simulations reproduced the behavior of the approximate hardware accelerators (Chapter 5) being selected by heuristics and by a decision tree (Chapter 7), in order to check the impacts on coding efficiency. It was taken into account the processing of 64x64 block sizes using one reference frame, random access temporal configuration, SAD as similarity criteria, and no bi-prediction to encode 200 frames from each of the 24 recommended video sequences (with exception of *Traffic* and *PeopleOnStreet* sequences, which have a maximum length of 150 frames) and Quantization Parameters (QP) – 22, 27, 32, and 37. The evaluation of the QoS for this different set of cases was made using the BD-BR metric, as aforementioned in this work.

As previously mentioned in Chapter 5, the PEs hardware design were described in VHDL and synthesized using a 40 nm TSMC standard cell library with 0.9V (TSMC, 2020) using the Cadence RTL Compiler tool (CADENCE, 2020). To perform the power estimation of the developed hardware, we have employed the default tool switching activity (20%). Note that the synthesis of each FAPP was made individually, therefore, when analysing the power of the PEs of SApp-NoC we need to consider the number

of instantiated PEs. However, this analysis of the average power dissipated by SApp-NoC (for both proposed solutions) is not trivial. A simple sum of the individual powers of each FAPP would be equivocated since they are not operating at full time. The number of each PE type, their active operating time, their *idle* time, the proportion between the application throughput and the individual FAPP throughput, and the percentage of FAPP selection must be taken into account in order to determine the average power of the NoC. Note that in Chapter 5 we presented FAPP results targeting different frequencies and frame rates, but, in the context of SApp-NoC, we have considered the usage of the smallest throughput FHD@30fps at the processing elements of SApp-NoC (since the inherent parallelism associated to NoCs).

When we design an ordinary hardware accelerator, we make the assumption that the hardware will operate at full time when establishing the operational frequency of the accelerator. When it comes to SApp-NoC it is necessary to check how much time each PE will operate, i.e., we need to determine the level of utilization of each FAPP. We define $util\_rate_i$ as the variable which measures the utilization of each $FAPP_i$, established as follows.

$$util\_rate_i = \frac{sel_{FAPPi} \cdot num_{PEi} \cdot RTAPE}{PE_{Active}} \tag{44}$$

One can notice in (44) that the utilization of each PE will depend on the percentage of selection of each FAPP (sel_{FAPPi}), on *RTAPE*, defined as the relation between the throughput of the application being processed at SApp-NoC ($throughput_{app}$) and the throughput conceived for the design of a single PE ($throughput_{PE}$), on the number of instantiated FAPPs (num_{PEi}), and on the number of active PEs in SApp-NoC (PE_{Active}). Observe that this relation is true only when all PEs are designed with the same frequency and the task allocation performs a uniform task-allocation (i.e., the same number of tasks are mapped on PEs having FAPPs of the same type). The relation between application's and PE's throughput is given by (45).

$$RTAPE = \frac{throughput_{app}}{throughput_{PE}} = \frac{resolution_{app} \cdot fps_{app}}{resolution_{PE} \cdot fps_{PE}} \tag{45}$$

Note that (45) is very straightforward, where RTAPE informs how many times the application is *more* or *less* demanding than a single PE.

Therefore, the power dissipation of SApp-NoC can be estimated for the average scenario by applying what is stated in (46).

$$P_{SAppNoC} = \sum_{i=0}^{3} P_{FAPPi} \cdot util\_rate_i \tag{46}$$

The energy estimation of SApp-NoC considers the power of each PE and the time each one is *active*, or *idle*, although the power gating control was not physically imple-

mented (since the modeling of SApp-NoC is made at a high level of abstraction), we consider the usage of power gating over each *idle* PE, disregarding its power dissipation when not active. Therefore, the energy consumption evaluation take into account the PE's total time of active processing (T_{active_i}), the power of the PEs (P_{FAPPi}), and the percentage of selection of each FAPP (sel_{FAPPi}), as presented in (47). Note that we considered energy consumption evaluation only of the processing elements, without taking into account the energy consumed at the communication across the NoC.

$$Energy_{SAppNoC} = \sum_{i=0}^{3} sel_{FAPPi} \cdot P_{FAPPi} \cdot T_{active_i} \tag{47}$$

Where the time of active processing is given as follows:

$$T_{active_i} = NBPF \cdot NF \cdot NC_{FAPPi} \cdot clock\_period \tag{48}$$

On the one hand, in (47) we consider P_{FAPPi} as the power of each single PE individually, which values were presented in Chapter 5. We made such an assumption taking into account that the amount of samples to be processed by each PE is constant, no matter the number of PEs their evaluation is split on. Thus when the number of PEs increases (bigger NoCs) the time of active processing of each one reduces in the same proportion. Therefore, the consumed energy is still constant for any given scenario (obviously it means that the processing will occur faster, dissipating more power - according to what we have stated in (46)). On the other hand, in (48) we present how we calculate the total time each PE type is active in SApp-NoC. It depends on the number of blocks to be processed regarding a single frame ($NBPF$) (obviously it will depend on the video resolution), the number of frames (NF) to be evaluated, the number of cycles each PE takes to evaluate a block (NC_{FAPPi}) (refer to Chapter 5 for details), and the clock period (defined by the PE operational frequency).

The modeling of HSApp-NoC and MLSApp-NoC architectures as well as instantiation, placement, task mapping, and SNFT algorithms, were made in a java-based environment. The NoC architecture was modeled in high level of abstraction, considering its topology, size and communication frequency. In the same way, the developed FAPPs were also modeled in a high level of abstraction and considering the operational frequency of 14.57 MHz, obtained during the hardware synthesis and discussed in Chapter 6. The schedulability analysis was also performed at this java-based environment, considering the modeling of the whole system which considers priority-preemptive task schedulers, 2D-mesh NoC interconnected with XY dimension routing, distributed memory, and eight virtual channels with priority-preemptive link arbitration.

8.2 NoC Breakdown Frequency Results

Regarding our case study targeting five throughput demands (FHD@30fps, FHD@60fps, 4K@30fps, 4K@60fps, and 4K@120fps), we applied SNFT to all possible scenarios (i.e., different SApp-NoC solutions and configurations - HSApp-NoC and MLSApp-NoC with different sizes), in order to determine the minimum breakdown frequency meeting all scenarios requirements for HSApp-NoC and MLSApp-NoC. The input considers a typical workload scenario, obtained from the reference software of HEVC. We encoded FHD and 4K video sequences with the same configuration presented in the beginning of this chapter, obtaining the average FAPP selection for HSApp-NoC and MLSApp-NoC. Therefore, a single frame employing such a behavior was built as input to test HSApp-NoC and MLSApp-NoC. In Figure 42 and 43 we show the results of SNFT when processing FHD video sequences, for HSApp-NoC and MLSApp-NoC, respectively.

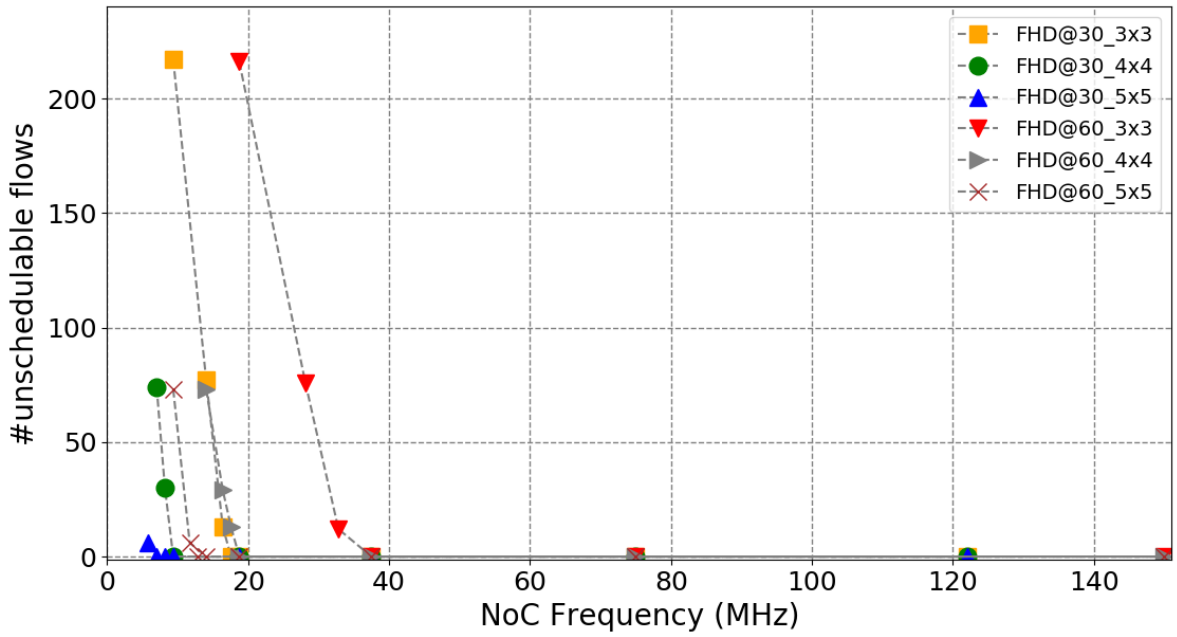

Figure 42 – SNFT results for HSApp-NoC processing FHD video sequences.

One can notice that, when analyzing Figures 42 and 43, the breakdown frequencies are reached at relatively small values, even smaller when the NoC size grows (which makes sense since the tasks are distributed over more PEs and the flows are better spread across the NoC). Regarding this scenario, considering the NoC size as smallest as possible, HSApp-NoC is schedulable at 17.58 MHz and 37.5 MHz, for FHD@30fps and FHD@60fps, respectively, for a HSApp-NoC sizing 3x3; and MLSApp-NoC is schedulable at 18.75 MHz and 37.5 MHz, for FHD@30fps and FHD@60fps, respectively, for a MLSApp-NoC sizing 3x3.

In Figure 44 we show the results of SNFT for HSApp-NoC processing 4K video sequences. This time, the breakdown frequencies are greater, since a more demand-

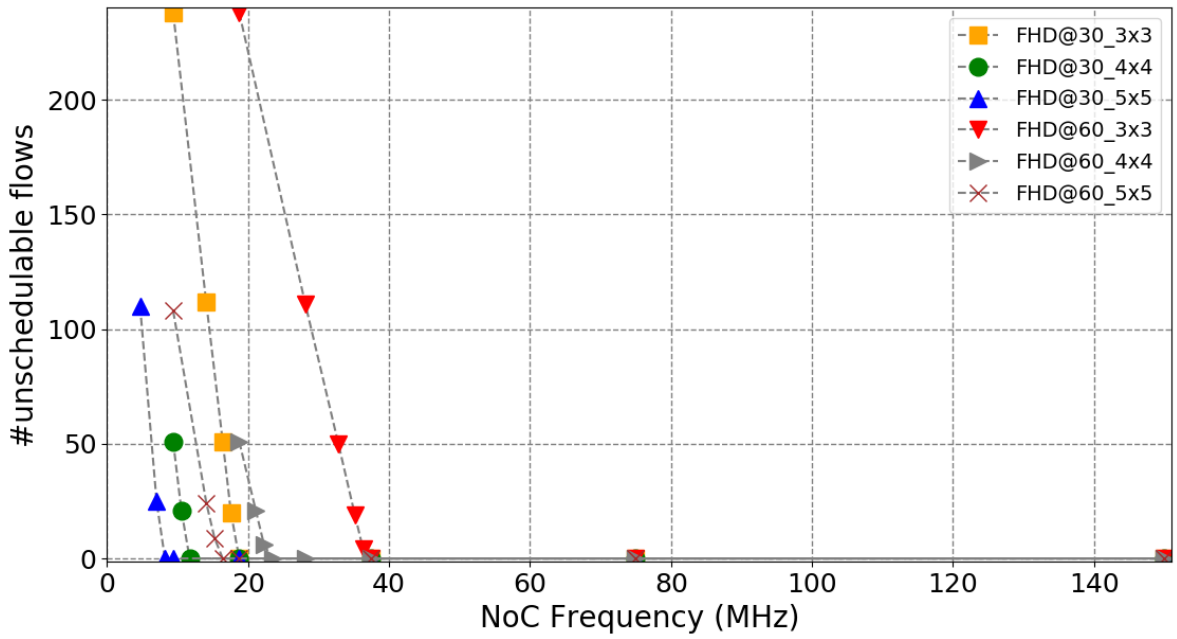

Figure 43 – SNFT results for MLSApp-NoC processing FHD video sequences.

ing scenario is found when processing 4K. The same way, considering the NoC size as small as possible, HSApp-NoC is schedulable at 85.55 MHz for 4K@30fps, for a HSApp-NoC sizing 3x3, at 103.13 MHz for 4K@60fps, for a MLSApp-NoC sizing 4x4, and at 122 MHz for 4K@120fps, for a HSApp-NoC sizing 5x5. Note that Figure 44 does not show results for 4K@120fps at HSApp-NoC 4x4 and 5x5. In these scenarios the tasks were unschedulable. Although 4K@60fps being schedulable at HSApp-NoC 3x3 at a communicating frequency of 174 MHz we have not considered this result due to the huge NoC frequency which is necessary, which would lead to processing waste when performing less demanding scenarios. Therefore, if we analyze the minimum breakdown frequencies which were found, the minimum frequency guaranteeing the system schedulability for any given scenario is 122 MHz, hence this value was selected for HSApp-NoC communication frequency.

In Figure 45 we show the results of SNFT for MLSApp-NoC processing 4K video sequences. This time, the breakdown frequencies are greater, since a more demanding scenario is found when processing 4K. The same way, considering the NoC size as small as possible, MLSApp-NoC is schedulable at 93.75 MHz for 4K@30fps, for a MLSApp-NoC sizing 3x3, at 112.5 MHz for 4K@60fps, for a MLSApp-NoC sizing 4x4, and at 150 MHz for 4K@120fps, for a MLSApp-NoC sizing 5x5. Note that Figure 45 does not show results for 4K@60fps at SApp-NoC 3x3, neither for 4K@120fps at SApp-NoC 4x4 and 5x5. In these scenarios the tasks were unschedulable. When analyzing the minimum breakdown frequencies which were found, the minimum frequency guaranteeing the system schedulability for any given scenario is 150 MHz, hence this value was selected for MLSApp-NoC communication frequency.

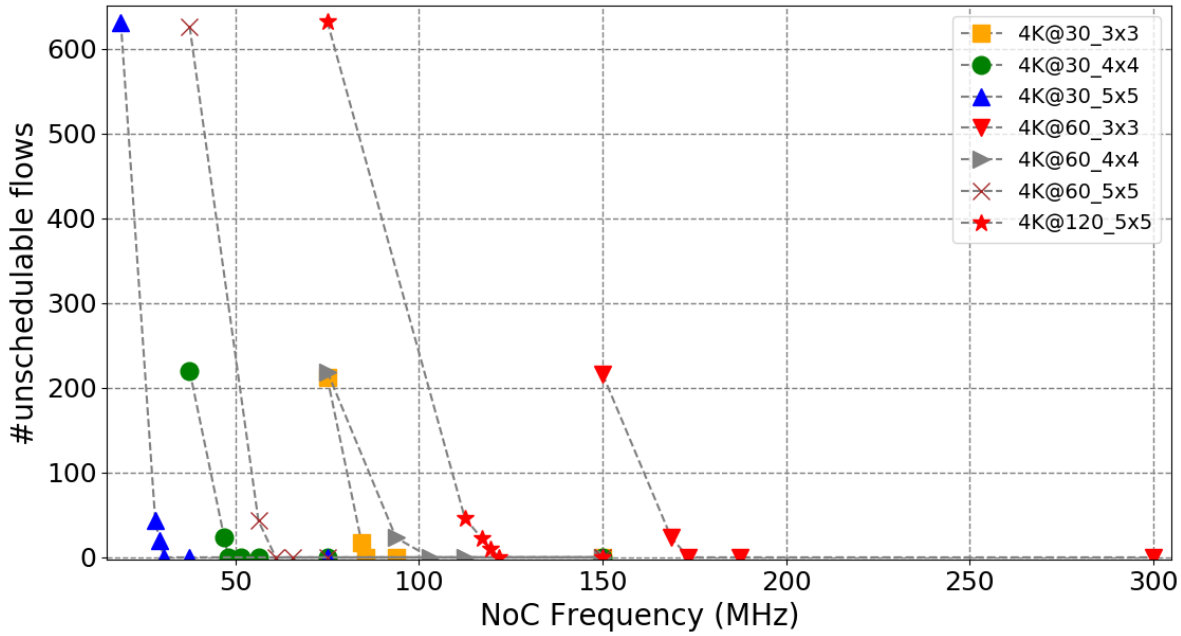

Figure 44 – SNFT results for HSApp-NoC processing 4K video sequences.

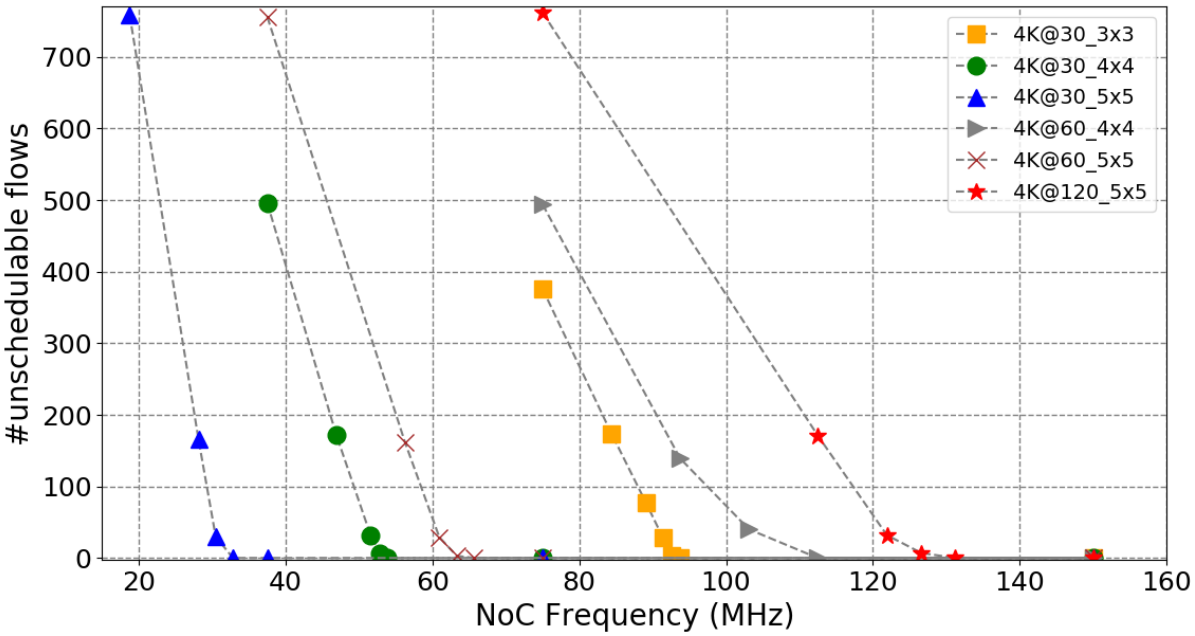

Figure 45 – SNFT results for SApp-NoC processing 4K video sequences.

8.3 Power/Energy Results

Table 11 presents HSApp-NoC and MLSapp-NoC synthesis results, making a comparison with related works that also propose ASIC hardware accelerators designs for the HEVC FME (Interpolation + Search). HE et al. (2015) presented a power-efficient architecture for HEVC FME with power dissipation of 198.6 mW and throughput of 8K@30fps (equivalent to 4K@120fps). In LUNG; SHEN (2019) it is proposed a VLSI architecture and implementation of the HEVC FME. Their design achieves a maximum

Table 11 – SApp-NoC synthesis results and comparison with related works

Parameter	*HE (2015)*	*LUNG (2019)*	*HSApp-NoC*	*MLSApp-NoC*
Power (mW)	198.6	304.8	60.30	47.34
Norm. Power (mW)	91.94	101.6	-	-
Gate Count (K)	1,183.00	525.40	1,994.57	2,731.50
Max. Throughput	4K@120fps	4K@39fps	4K@120fps	4K@120fps
Technology (nm)	65	90	40	40
Voltage (V)	1.2	1.2	0.9	0.9

throughput of 4K@39fps, with a power dissipation of 304.8 mW. Considering the different technologies and voltage supplies employed by these works, in order to make a fair comparison, we have also calculated the normalized power of the related works for 40 nm/0.9V, following the method introduced by HE et al. (2015). When comparing the normalized FME power of the related works with our solutions, it can be noticed that our both solutions presented smaller power dissipation. Moreover, when compared individually, our PEs demand the smallest area among related works (refer to Table 4 in Chapter 5). When considering the whole design (SApp-NoC), the area results must be added according to the number of nodes used for active PEs, resulting, as expected, in the biggest area among related works due to parallelism exploration (note that PEs are turned off when *idle*). It is worth to mention that, in this comparison, only the power dissipation of the FAPPs were considered, i.e., the power dissipation of the FME unit. The power dissipation at GPPs, NoC routers, local memory, and communication links was not considered in the comparison.

When analyzing the energy results, firstly we need to consider the average FAPP selection at HSApp-NoC and MLSApp-NoC, in terms of how many times each FAPP is selected, considering the four recommended QPs, for each video sequence, in order to find the active time of each PE. In Tables 12 and 13 we show the average selection of each FAPP for HSApp-NoC and MSApp-NoC, respectively, when processing 200 frames of all CTC (except *Traffic* and *PeopleOnStreet* which have only 150 frames) and 4K sequences, not included in the CTC but also tested by our work (for simplicity, we will refer 4K sequences as 4K class). In the energy consumption analysis we have used the 4K sequences *Beauty* (MERCAT; VIITANEN; VANNE, 2020), *Foreman* and *Cactus* (AMAZON, 2020). The frame rate and the SApp-Noc size are also presented in Table 13.

In Tables 14 and 15 we present, for HSApp-NoC and MLSApp-NoC, for the same video sequences and recommended QPs, the energy consumption per frame (**Epf**), the energy consumption per second (**Eps**), the total energy consumption (**Et**) considering the encoding of 200 frames, and the energy savings when compared with FAPP0 solution (**Es**). Sequences from same class presented similar energy consumption,

Table 12 – Average selection (%) of each FAPP during video sequences processing at HSApp-NoC

Class	Sequence	fps	Size	FAPP0	FAPP1	FAPP2	FAPP3
4K	*Beauty4K*	120	5x5	25.42	25.39	15.44	33.64
	Foreman4K	60	4x4	16.94	15.25	11.07	56.76
	Cactus4K	60	4x4	17.47	18.53	11.82	52.21
Average Selection 4K				19.95	19.72	12.81	47.54
A	*Traffic*	30	4x4	1.03	3.06	1.96	93.95
	PeopleOnStreet	30	4x4	8.76	18.54	6.01	66.69
	Nebuta	60	4x4	2.21	9.20	8.10	80.49
	SteamLocomotive	60	4x4	9.28	11.65	3.04	80.49
Average Selection Class A				5.32	10.61	4.78	80.41
B	*Kimono*	24	3x3	5.30	14.64	7.21	72.86
	ParkScene	24	3x3	2.13	5.39	3.28	89.20
	Cactus	50	3x3	5.23	7.13	2.35	85.30
	BQTerrace	60	3x3	2.46	5.34	2.52	89.69
	BasketballDrive	50	3x3	11.79	17.42	3.92	66.87
Average Selection Class B				5.38	9.98	3.85	80.78
C	*RaceHorsesC*	30	3x3	12.69	21.64	7.07	58.60
	BQMall	60	3x3	5.17	9.03	2.82	82.99
	PartyScene	50	3x3	3.95	6.20	2.16	87.69
	BasketballDrill	50	3x3	7.17	8.55	3.67	80.61
Average Selection Class C				7.25	11.36	3.93	77.47
D	*RaceHorses*	30	3x3	8.50	20.49	8.98	62.03
	BQSquare	60	3x3	0.16	0.83	1.33	97.70
	BlowingBubbles	50	3x3	2.19	4.47	3.41	89.92
	BasketballPass	50	3x3	5.07	10.89	3.40	80.64
Average Selection Class D				3.98	9.17	4.28	82.57
E	*FourPeople*	60	3x3	0.89	2.97	1.01	95.13
	Johnny	60	3x3	0.49	3.21	1.23	95.08
	KristenAndSara	60	3x3	1.22	5.04	2.28	91.45
Average Selection Class E				0.87	3.74	1.51	93.89
F	*BasketballDrillText*	50	3x3	7.26	8.16	3.27	81.31
	ChinaSpeed	30	3x3	17.46	15.33	2.91	64.29
	SlideEditing	30	3x3	0.99	1.04	0.13	97.85
	SlideShow	20	3x3	7.05	4.43	0.82	87.70
Average Selection Class F				8.19	7.24	1.78	82.79

which depend on the resolution, frame rate and SApp-NoC size. For higher resolutions such as 4K, *Beauty* sequence presented the highest energy consumption, since it also presents the highest frame rate.

In order to analyze the dynamic behavior of SApp-NoC, in Figure 46 we show the energy per frame of Kimono video sequence being encoded with QP37. One can notice a well-behaved consumption until frame 140, alternating the FAPP selection and keeping the energy consumption smaller than FAPP2-only would provide. At this point, where we have made a zoom for better detailing, change scene happens, from a more

Table 13 – Average selection (%) of each FAPP during video sequences processing at MLSApp-NoC

Class	Sequence	fps	Size	FAPP0	FAPP1	FAPP2	FAPP3
4K	*Beauty4K*	120	5x5	8.73	5.86	70.98	14.43
	Foreman4K	60	4x4	15.67	15.16	35.74	33.43
	Cactus4K	60	4x4	13.65	14.02	39.67	32.67
Average Selection 4K				12.68	11.68	48.80	26.84
A	*Traffic*	30	4x4	24.44	20.84	19.46	35.26
	PeopleOnStreet	30	4x4	17.17	11.63	61.41	9.79
	Nebuta	60	4x4	21.94	16.32	48.23	13.52
	SteamLocomotive	60	4x4	16.82	14.55	35.62	33.02
Average Selection Class A				20.09	15.83	41.18	22.90
B	*Kimono*	24	3x3	20.07	13.22	46.24	20.47
	ParkScene	24	3x3	23.66	18.32	26.72	31.31
	Cactus	50	3x3	21.31	19.08	28.25	31.36
	BQTerrace	60	3x3	22.16	19.32	18.55	39.97
	BasketballDrive	50	3x3	16.02	11.12	50.85	22.00
Average Selection Class B				20.64	16.21	34.12	29.02
C	*RaceHorsesC*	30	3x3	13.07	7.24	71.01	8.68
	BQMall	60	3x3	21.42	17.26	38.92	22.42
	PartyScene	50	3x3	22.56	18.99	36.25	22.20
	BasketballDrill	50	3x3	21.15	15.06	40.64	23.16
Average Selection Class C				19.55	14.63	46.70	19.11
D	*RaceHorses*	30	3x3	14.50	7.96	71.52	6.03
	BQSquare	60	3x3	25.14	22.44	18.98	33.44
	BlowingBubbles	50	3x3	23.83	18.86	34.67	22.64
	BasketballPass	50	3x3	19.58	13.93	48.50	17.99
Average Selection Class D				20.76	15.80	43.42	20.02
E	*FourPeople*	60	3x3	23.92	22.64	9.85	43.59
	Johnny	60	3x3	22.47	21.83	11.21	44.49
	KristenAndSara	60	3x3	23.23	20.56	14.78	41.43
Average Selection Class E				23.21	21.68	11.95	43.17
F	*BasketballDrillText*	50	3x3	21.22	15.64	40.69	22.46
	ChinaSpeed	30	3x3	14.91	10.58	58.11	16.40
	SlideEditing	30	3x3	24.45	24.01	3.64	47.91
	SlideShow	20	3x3	19.95	18.04	23.12	38.89
Average Selection Class F				20.13	17.06	31.39	31.41

heterogeneous (look details detached in orange in the image) to a more homogeneous scenario. Since the presence of more homogeneous regions (look the sky background detached in red in the image detail), more aggressive approximation is applied, decreasing the energy consumption. Note that at some points the energy consumption falls to zero, it happens due to the adopted temporal configuration (Random Access), which inserts an All-Intra (AI) frame periodically, which keeps the entire SApp-NoC in *idle* mode).

In order to verify the dynamic behavior of objective video quality (in terms of Y-

Table 14 – Average energy consumption of HSApp-NoC during video sequences processing

Class	Sequence	fps	Size	Epf (mJ)	Eps (mJ)	Et (mJ)	Es (%)
4K	*Beauty4K*	120	5x5	0.818	98.111	157.795	28.34
	Foreman4K	60	4x4	0.698	41.855	134.635	38.86
	Cactus4K	60	4x4	0.720	43.213	139.000	36.87
Average 4K				0.745	61.059	143.810	34.69
A	*Traffic*	30	4x4	0.264	7.921	38.283	55.57
	PeopleOnStreet	30	4x4	0.338	10.151	49.063	43.06
	Nebuta	60	4x4	0.292	17.535	56.404	50.82
	SteamLocomotive	60	4x4	0.329	19.721	63.434	44.69
Average Class A				0.306	13.832	51.796	48.42
B	*Kimono*	24	3x3	0.152	3.652	29.366	46.66
	ParkScene	24	3x3	0.132	3.178	25.555	53.58
	Cactus	50	3x3	0.139	6.966	26.887	51.16
	BQTerrace	60	3x3	0.132	7.946	25.560	53.57
	BasketballDrive	50	3x3	0.165	8.239	31.801	42.23
Average Class B				0.165	8.239	31.801	42.23
C	*RaceHorsesC*	30	3x3	0.033	0.988	6.353	39.12
	BQMall	60	3x3	0.027	1.615	5.196	50.21
	PartyScene	50	3x3	0.026	1.288	4.971	52.36
	BasketballDrill	50	3x3	0.028	1.376	5.311	49.11
Average Class C				0.028	1.317	5.458	47.70
D	*RaceHorses*	30	3x3	0.006	0.187	1.203	41.72
	BQSquare	60	3x3	0.005	0.274	0.883	57.24
	BlowingBubbles	50	3x3	0.005	0.246	0.950	53.96
	BasketballPass	50	3x3	0.005	0.271	1.047	49.28
Average Class D				0.005	0.245	1.021	49.28
E	*FourPeople*	60	3x3	0.058	3.461	11.133	55.88
	Johnny	60	3x3	0.058	3.454	11.112	55.96
	KristenAndSara	60	3x3	0.060	3.571	11.488	54.47
Average Class E				0.058	3.496	11.244	55.43
F	*BasketballDrillText*	50	3x3	0.066	3.313	12.787	49.32
	ChinaSpeed	30	3x3	0.078	2.349	15.109	40.12
	SlideEditing	30	3x3	0.056	1.690	10.875	56.90
	SlideShow	20	3x3	0.063	1.264	12.196	51.66
Average Class F				0.066	2.154	12.742	49.50
Average Energy Saving when Compared to FAPP0							48.19

PSNR), in Figure 47 we show the Y-PSNR behavior for along the processing of *Kimono* sequence with QP37 at SApp-NoC. One can notice that, at the scene change, a decrease on video quality happens, which happens due to the change of the frame being encoded and the reference frame being the same from previous scene. After the AI frame, the PSNR returns to the same average level. It is worth to notice that our content-adaptive solution aims to keep the PSNR as close as possible to the PSNR of the precise solution. Note that there are some peaks of PSNR, which happens when

Table 15 – Average energy consumption of MLSApp-NoC during video sequences processing

Class	Sequence	fps	Size	Epf (mJ)	Eps (mJ)	Et (mJ)	Es (%)
4K	*Beauty4K*	120	5x5	0.726	87.111	140.103	36.37
	Foreman4K	60	4x4	0.743	44.598	143.457	34.85
	Cactus4K	60	4x4	0.732	43.949	141.368	35.80
Average 4K				0.734	58.552	141.643	35.67
A	*Traffic*	30	4x4	0.414	12.429	60.075	30.28
	PeopleOnStreet	30	4x4	0.412	12.363	59.757	30.65
	Nebuta	60	4x4	0.426	25.582	82.290	28.25
	SteamLocomotive	60	4x4	0.389	23.353	75.119	34.50
Average Class A				0.411	18.432	69.310	30.98
B	*Kimono*	24	3x3	0.196	9.811	37.871	31.20
	ParkScene	24	3x3	0.198	4.758	38.261	30.50
	Cactus	50	3x3	0.196	9.811	37.871	31.20
	BQTerrace	60	3x3	0.193	11.554	37.166	32.48
	BasketballDrive	50	3x3	0.189	9.470	36.554	33.60
Average Class B				0.195	8.061	37.549	31.79
C	*RaceHorsesC*	30	3x3	0.036	1.084	6.971	33.22
	BQMall	60	3x3	0.038	2.274	7.313	29.92
	PartyScene	50	3x3	0.038	1.920	7.413	28.97
	BasketballDrill	50	3x3	0.037	1.872	7.225	30.77
Average Class C				0.037	1.787	7.230	30.72
D	*RaceHorses*	30	3x3	0.007	0.218	1.405	31.95
	BQSquare	60	3x3	0.008	0.454	1.461	29.21
	BlowingBubbles	50	3x3	0.008	0.382	1.474	28.60
	BasketballPass	50	3x3	0.007	0.371	1.431	30.68
Average Class D				0.007	0.356	1.443	30.11
E	*FourPeople*	60	3x3	0.089	5.365	17.258	31.60
	Johnny	60	3x3	0.088	5.290	17.016	32.56
	KristenAndSara	60	3x3	0.089	5.332	17.152	32.02
Average Class E				0.089	5.329	17.142	32.06
F	*BasketballDrillText*	50	3x3	0.091	4.546	17.547	30.45
	ChinaSpeed	30	3x3	0.087	2.624	16.883	33.09
	SlideEditing	30	3x3	0.089	2.672	17.190	31.87
	SlideShow	20	3x3	0.087	1.739	16.784	33.48
Average Class F				0.089	2.895	17.101	32.22
Average Energy Saving when Compared to FAPP0							31.81

the AI frames are processed (AI delivers higher PSNRs).

8.4 QoS Results

The Quality-of-Service results for all CTC is presented in Table 16 and considered the experimental setup presented in the first section of this chapter, for the both solutions HSApp-NoC and MLSApp-NoC. The evaluation of the QoS for this different set of cases was made using the metric BD-BR. As previously mentioned in this thesis, the

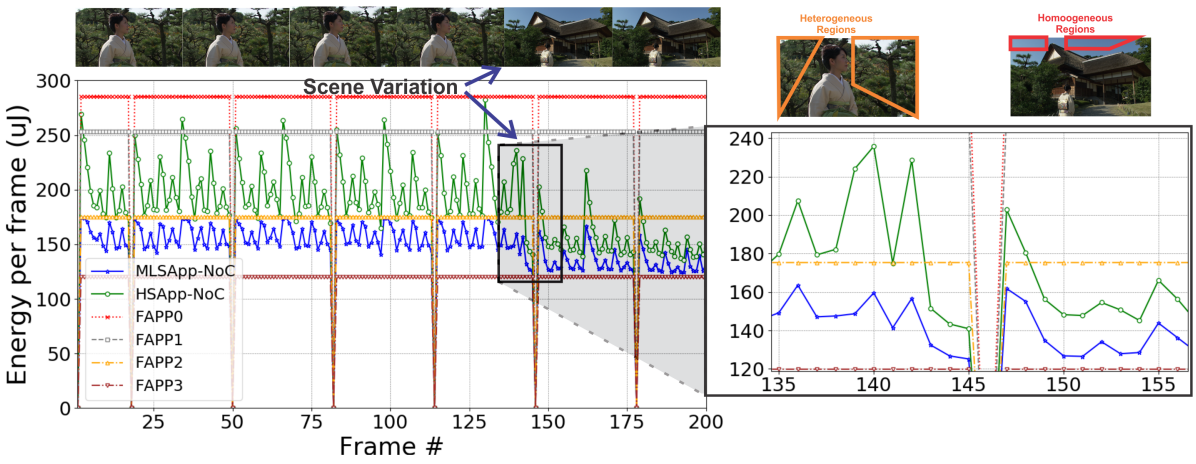

Figure 46 – Energy per frame (uJ) for Kimono sequence with QP37.

BD-BR is widely used by the academic and standardization community to fairly measure the quality losses. Such a metric informs, in percentage (%), the bitrate increase (positive BD-BR - which means a quality loss) or decrease (negative BD-BR - which means a quality gain) for the same objective quality, when compared with a baseline (in our case FAPP0 since it is a direct implementation of HEVC FME). It is worth to notice that when dealing with approximate solutions, the results will always be positive (meaning quality losses) since we are removing functionalities from the original implementation in order to simplify some variable (e.g. energy/power).

When analyzing the results present in Table 16, SApp-NoC balances the energy consumption and QoS degradation reaching the energy reduction at a minimum average cost of 2.74% of BD-BR for HSApp-NoC, and of 1.09% of BD-BR for MLSApp-NoC. Such results are due to our content-based task allocation that smartly selects the approximate PEs for more suitable regions, using heuristics and machine learning strategies. When looking for quality results, our achieved QoS values are considerably good when compared with related works. For instance, work HE et al. (2015) present a QoS

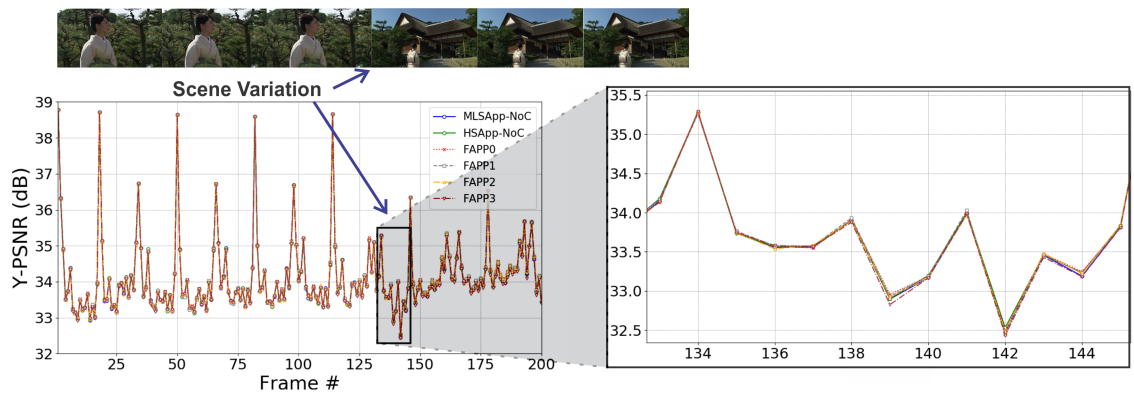

Figure 47 – Y-PSNR (dB) per frame for Kimono sequence with QP37.

of -0.04 dB (they have used another quality metric called BD-PSNR (BJONTEGAARD, 2001) for QoS), our average equivalent QoS degradation in this metric is -0.03564 dB (for MLSApp-NoC) and -0.08999 dB (for HSApp-NoC), whereas work LUNG; SHEN (2019) present 2.08% of quality losses.

Table 16 – QoS results in terms of BD-BR (%) for SApp-NoC

Class / Resolution	***Sequence***	***HSApp-NoC***	***MLSApp-NoC***
Class A / 2560x1600	*Traffic*	7.712167	2.056503
	PeopleOnStreet	1.315470	0.585345
	Nebuta	0.979198	0.756005
	SteamLocomotive	3.427029	1.725584
Class B / 1920x1080	*Kimono*	1.044242	0.749071
	ParkScene	4.052259	1.735514
	Cactus	1.221857	0.893556
	BQTerrace	6.305922	2.669709
	BasketballDrive	2.047288	1.457541
Class C / 832x480	*RaceHorsesC*	2.087318	1.199979
	BQMall	3.267984	1.283681
	PartyScene	1.579224	0.356868
	BasketballDrill	8.638696	1.289618
Class D / 416x240	*RaceHorses*	1.160884	1.256889
	BQSquare	2.448579	0.884385
	BlowingBubbles	3.474428	1.159798
	BasketballPass	1.287565	0.790805
Class E / 1280x720	*FourPeople*	1.118276	0.685059
	Johnny	3.121819	1.738688
	KristenAndSara	0.998400	0.911402
Class F / 1280x720	*BasketballDrillText*	7.975378	1.329306
	ChinaSpeed	0.215654	0.273964
	SlideEditing	0.001073	-0.018106
	SlideShow	1.395129	0.865921
Average		2.736987	1.093655

For a visual comprehension, we have plotted the QoS results in terms of BD-BR for each class. FAPP3 presents the worse results, since it employs a more aggressive approximation. Regarding this scenario, class B presented the worse QoS among all classes being executed by FAPP3. Such a behavior happens due to the nature of some videos. Specifically, in class B there is the *BQTerrace* sequence, considered as high complexity sequence (HCS) (indeed it presents the highest complexity among the benchmarks (CASSA; NACCARI; PEREIRA, 2012)), presenting the highest impacts on QoS when approximation is applied (look at Tables 6 and 48). In fact, when compared to solutions that cannot adapt to the content, HSApp-NoC and MLSApp-NoC drastically reduce QoS degradation in relation to FAPP3 (class B makes the QoS difference clearer), with MLSApp-NoC performing close to FAPP1 and FAPP2 in terms of QoS,

while reducing the energy consumption, since only homogeneous regions of the frame are explored.

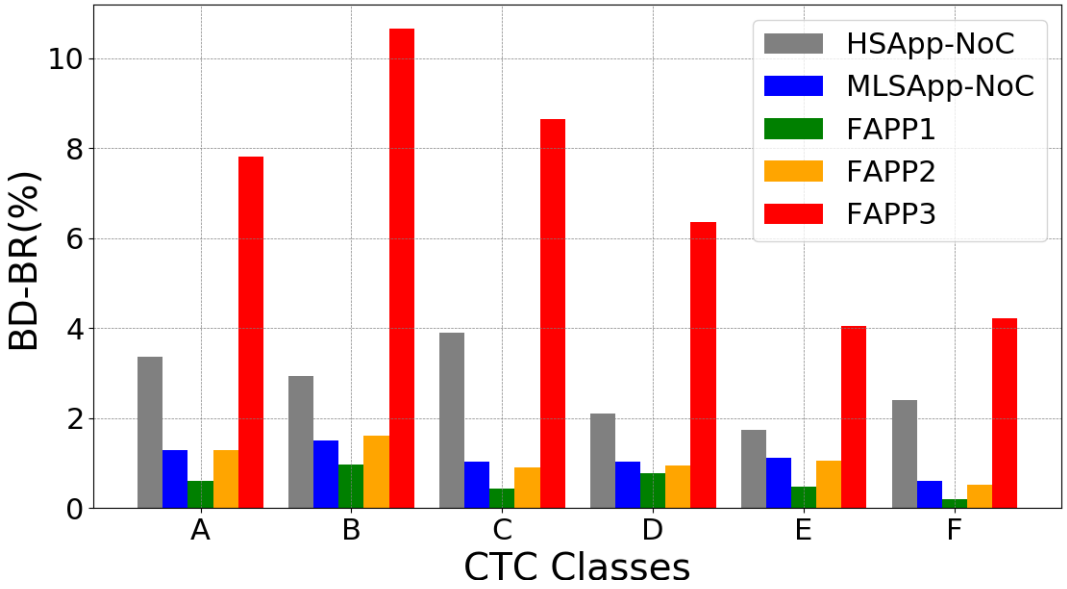

Figure 48 – QoS results in terms of BD-BR for each class.

8.5 Performance Analysis Results

The task-mapping adopted in this work is based on the application behavior. Despite being energy-QoS efficient, such allocation does not guarantee the meeting of real-time constraints. On the one hand, a response time/schedulability analysis, targeting dynamic task-allocation, covering any possible scenario, is not trivial to execute. On the other hand, since a wide range of benchmarks (BOSSEN, 2013) is provided by video community, a set of possible scenarios, describing a typical scenario, is proposed in order to evaluate the end-to-end schedulability of SApp-NoC over different scenarios through the verification of the worst-case response times: from tasks, flows and end-to-end. In order to verify the schedulability, we focus on high resolution workloads (FHD and 4K). We propose the evaluation of the encoding of FHD (class B, at 30 fps and 60 fps) and UHD videos (4K, at 60 fps and 120 fps), for the four recommended QPs, covering different movement and texture situations. The average selection of each accelerator was considered for each set (gathered from Tables 12 and 13, for HSApp-NoC and MLSApp-NoC, respectively) and the performance analysis was applied for each generated mapping, obtaining the worst-case latencies for each scenario.

In Figures 49 and 50 it is presented the response-time/schedulability analysis for HSApp-NoC and MLSApp-NoC, presenting the worst-case response times (WCRT) from tasks (**T**), flows (**F**), and end-to-end (**E**) (tasks plus flows) among the found worst-case latencies for each generated mapping, considering a frame rate of 30 fps, where

each video workload was mapped onto different SApp-NoC sizes, FHD on 3x3 and 4K on 4x4 (as aforementioned in this work). The both scenarios are schedulable, i.e., the end-to-end WCRT is smaller than the deadline – 33.33 ms for 30fps).

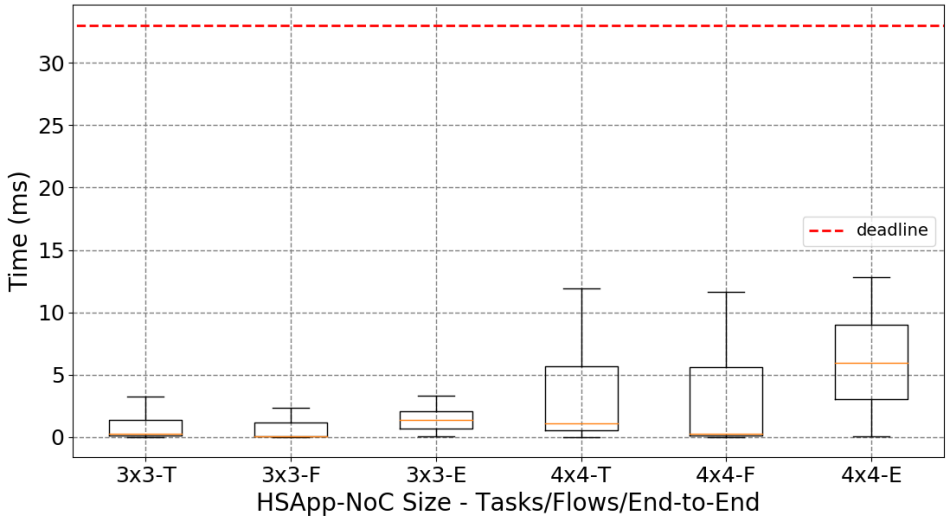

Figure 49 – Response times of HSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for FHD@30fps and 4K@30fps.

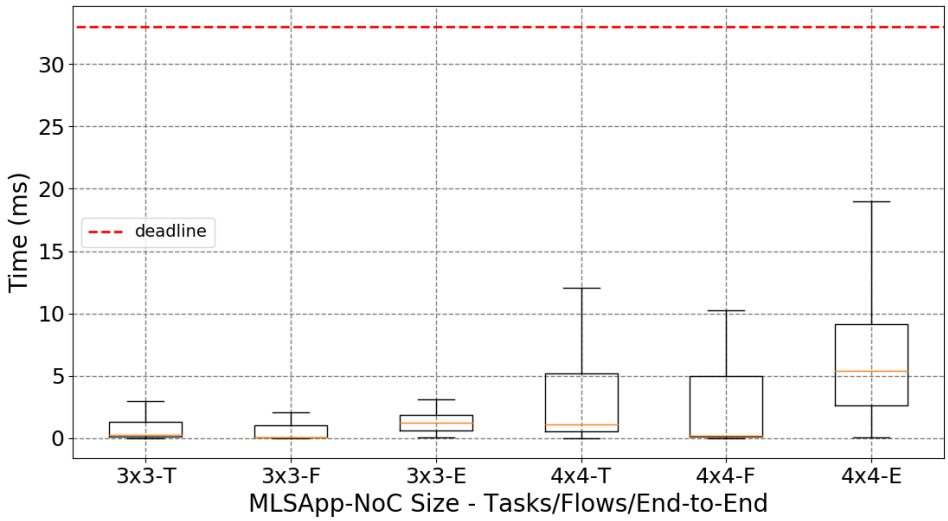

Figure 50 – Response times of MLSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for FHD@30fps and 4K@30fps.

In Figures 51 and 52 it is made the same analysis for HSApp-NoC, and in Figures 53 and 54 it is made the same analysis for MLSApp-NoC, also showing the WCRT from tasks, flows, and end-to-end, for 60 fps and 120 fps, respectively. In Figures 51 and 53 we have two scenarios, FHD videos mapped onto 3x3 SApp-NoC and 4K videos mapped onto 4x4 SApp-NoC. In this case, we also demonstrate that the system is capable of meeting the timing constraints (deadline equals to 16.67 ms for 60 fps) for these scenarios, being also schedulable. In Figure 52 and 54 the most demanding

workload scenario is analyzed, when a 4K video is mapped onto a 5x5 SApp-NoC, but with a harder timing constraint (deadline 8.33 ms for 120 fps). The system is also schedulable at this scenario, but this time with an end-to-end WCRT closer to the deadline.

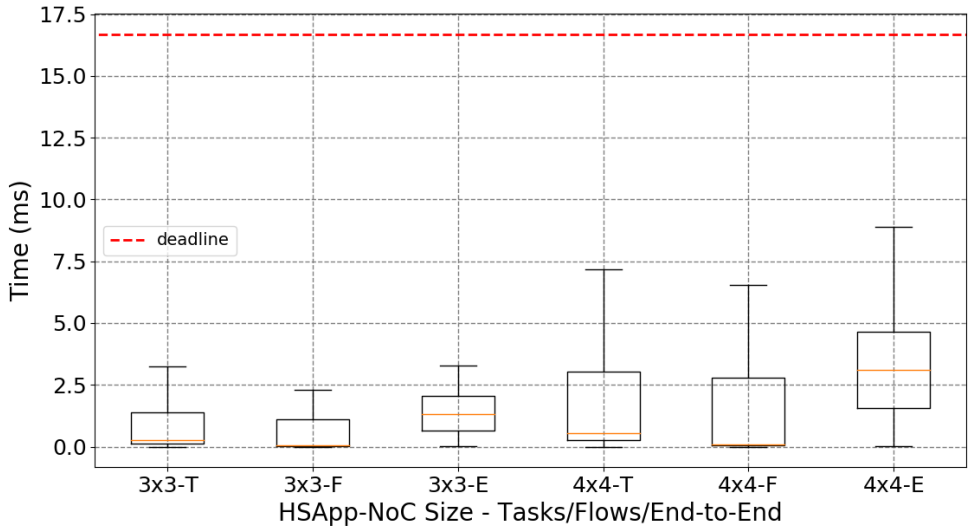

Figure 51 – Response times of HSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for FHD@60fps and 4K@60fps.

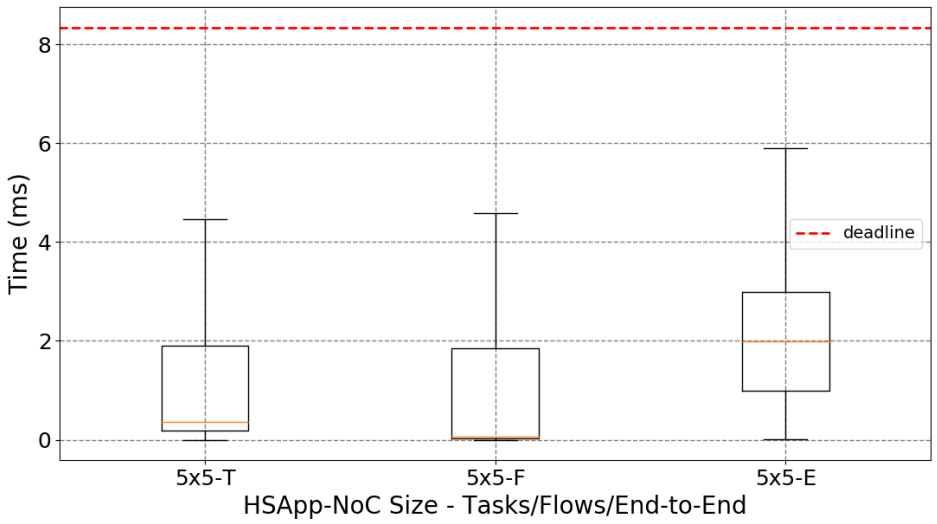

Figure 52 – Response times of HSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for 4K@120fps.

8.6 Results Summary

In this chapter we have demonstrated the effectiveness of our proposed solutions HSApp-NoC and MLSApp-NoC, by discussing the obtained results. First, we have showed the importance of SNFT in the specification of a NoC breakdown communication frequency. Next, the results for both solutions were presented, regarding energy

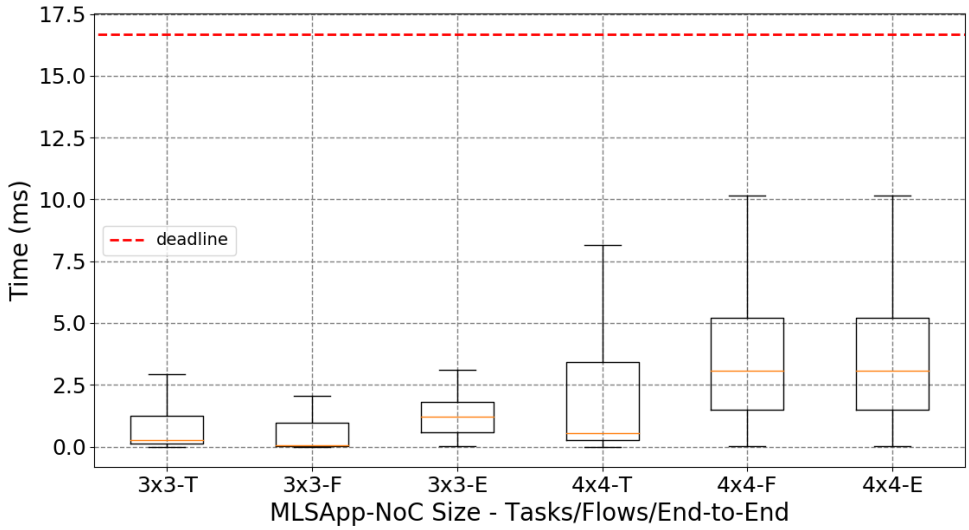

Figure 53 – Response times of MLSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for FHD@60fps and 4K@60fps.

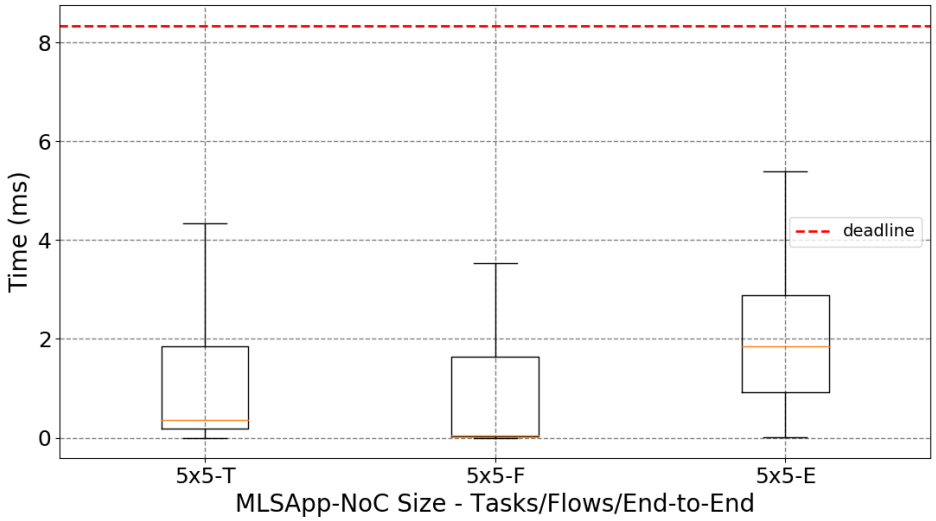

Figure 54 – Response times of MLSApp-NoC regarding tasks (T), flows (F), and end-to-end (E) for 4K@120fps.

consumption and QoS in terms of coding efficiency. On the one hand, HSApp-NoC presented higher energy savings, since it employs the approximation in a deeper level. On the other hand, MLSApp-NoC employs a more precise approximation control, presenting smaller energy savings when compared with HSApp-NoC, but presenting better QoS. For both solutions it was demonstrated with the schedulability analysis that they are able to meet timing constraints, delivering real-time performance.

For the best of author's knowledge, neither of the related works employs all strategies we have proposed in the same solution, delivering a system exploring parallelism by using NoCs, energy efficiency by using approximate hardware acceleration, QoS control by exploring properties of error-resilient applications, and real-time guarantees by performing a schedulability analysis. Related works proposing the exploration of

NoCs for video coding (PENNY et al., 2019a; MA et al., 2015; NOURI; GHAZNAVI-YOUVALARI; NURMI, 2018; PENNY et al., 2019b) have the main focus on exploring the performance to deal with real-time, without exploring approximate hardware or error-resilience. When considering related works proposing hardware acceleration for the HEVC FME (HE et al., 2015; LUNG; SHEN, 2019), our both solutions deliver higher energy savings with better QoS. None of the related works exploring approximate computing for the HEVC FME hardware acceleration presented results for a fair comparison, some of them focuses only on interpolation filters (KALALI; HAMZAOGLU, 2018; SILVA; SIQUEIRA; GRELLERT, 2019) or focuses on the motion compensation step (DIEFY; SHALABY; SAYED, 2015; PALUMBO et al., 2016). Furthermore, some of them do not present ASIC solutions (SAU et al., 2017) or focuses only on FME approximate interpolator design, as our previous work (PENNY et al., 2020).

9 CONCLUSION

This thesis presented the exploration of energy-efficient NoC-based systems for real-time multimedia applications using approximate computing. We defined some research hypotheses in order to answer the main research questions, and we have tried to verify the validity of each of them by addressing different strategies.

We proposed an energy/QoS-aware video processing system featuring scalable NoC topology and multi-level approximate hardware acceleration, called SApp-NoC, proposing a scalable NoC architecture targeting heterogeneous processing elements, designed to accelerate the HEVC FME at multiple levels of approximation. In order to allow scalability when dealing with distinct throughput demands (like frame rate and resolution), our solution employed the strategy of neighbor Tiles to properly size SApp-NoC, being able to deal with 4K UHD videos at 120 fps in a real-time processing. The observed results demonstrate the validity of our research hypothesis RH1, which suggests the use of NoC-based solutions to provide scalability in processing systems.

SApp-NoC processing elements were designed featuring four levels of approximation, implemented as dedicated hardware, and exploring the natural error resiliency of video coding application. These implementations address our research hypothesis RH2, which suggests that energy efficiency with real-time support can be obtained by exploring dedicated hardware acceleration applying approximate computing techniques over the application.

The NoC size, tiling definition as well as the processing elements instantiation and placement within SApp-NoC were conceived at design time, using heuristics based on the statistical behavior of the application, leading to HSApp-NoC solution, and machine learning exploration, leading to MLSApp-NoC solution. Nevertheless, at run-time, the developed application-aware dynamic task-mapping algorithm guarantees a real-time processing exploiting the energy/QoS trade-off. The obtained results show the validity of our research hypothesis RH3, which infers that the reduction of energy consumption maintaining the application QoS at acceptable range can be addressed by exploring application-specific properties/behavior with a run-time management.

A performance evaluation with a set of schedulability analysis was also proposed

for typical workload scenarios in video coding, guaranteeing the meet of timing constraints for different configurations. Therefore, MLSApp-NoC reduces about 31.81% the energy consumption at small quality reduction (1.09% BD-BR), whereas HSApp-NoC is capable of saving up to 48.19% the energy consumption with also small QoS of only 2.74% in BD-BR.

When revisiting the literature, we verified that many challenges are still open regarding digital systems design for multimedia applications when our focus is to provide, simultaneously, scalability to multiples throughput, performance for real-time processing and energy efficiency with low QoS degradation. We have hypothesized that these challenges could be addressed by employing NoCs to provide scalability and performance by parallelism exploration; by using hardware acceleration and approximate computing for energy efficiency, and by leveraging application-specific properties exploring error-resilient applications using heuristics and machine learning. Our solutions have successfully achieved our main goals by proposing different strategies and integrating them into two solutions: HSApp-NoC and MLSApp-NoC. Moreover, our methodology is also suitable to be applied to other error-resilient processing kernels for energy saving.

9.1 Future Works

As future works we would like to introduce the following aspects:

- Perform a more detailed design space exploration of approximate FME hardware accelerators, prioritizing further energy reduction while avoiding quality degradation;
- Introduce more parameters for evaluation during the machine learning analysis;
- Enhance SApp-NoC size in order to allow higher throughput;
- Reproduce the same methodology explored in this thesis on other video codecs like the AV1 (AOMedia Video 1) and VVC (Versatile Video Coding);
- Reproduce the same methodology on other encoding steps, like intra-frame prediction and integer motion estimation;
- Reproduce the same methodology on other multimedia applications, like Internet-of-things (IoT), computer vision, immersive technologies, and so on;
- Perform a study in order to allow the dynamic schedulability analysis of SApp-NoC (instead of the schedulability analysis applied for a typical workload, based on statistical behavior, which was applied in this thesis).

Our work has demonstrated the effectiveness of the proposed solutions by exploiting energy-efficient NoC-based systems for real-time multimedia applications using hardware acceleration and approximate computing. However, an enormous research space is still open regarding other contexts and applications. Thereby, this work has open the opportunity for different researches focusing on the development of approximate dedicate hardware, NoC designs focusing on specific applications, heuristics and algorithms exploiting the energy/QoS trade-off, and schedulability analysis methods for real-time applications.

REFERENCES

AFONSO, V. et al. Hardware implementation for the HEVC fractional motion estimation targeting real-time and low-energy. **J. Integr. Circuits Syst.**, [S.l.], v.11, n.2, p.106–120, 2016.

AFONSO, V.; MAICH, H.; AGOSTINI, L.; FRANCO, D. Low cost and high throughput FME interpolation for the HEVC emerging video coding standard. In: IEEE 4TH LATIN AMERICAN SYMPOSIUM ON CIRCUITS AND SYSTEMS (LASCAS), 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.1–4.

AGARWAL, A. The tile processor: A 64-core multicore for embedded processing. In: HPEC WORKSHOP, 2007. **Proceedings...** [S.l.: s.n.], 2007.

ALI, H. et al. Contention & energy-aware real-time task mapping on noc based heterogeneous mpsocs. **IEEE Access**, [S.l.], v.6, p.75110–75123, 2018.

ALIKHAH-ASL, E.; RESHADI, M. XY-axis and distance based NoC mapping (XY-ADB). In: INTERNATIONAL SYMPOSIUM ON TELECOMMUNICATIONS (IST), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.678–683.

AMAZON. **Recursos de mídia e entretenimento**. Disponível em: <https://aws.amazon.com/pt/media/resources/>. Acesso em: 30 jul. 2020.

AUDSLEY, N. et al. Applying new scheduling theory to static priority pre-emptive scheduling. **Software engineering journal**, [S.l.], v.8, n.5, p.284–292, 1993.

BAI, H. et al. Multiple description video coding based on human visual system characteristics. **IEEE transactions on circuits and systems for video technology**, [S.l.], v.24, n.8, p.1390–1394, 2014.

BARGE, I. J.; ABABEI, C. A network-on-chip based h. 264 video decoder prototype implemented on fpgas. In: IEEE 25TH ANNUAL INTERNATIONAL SYMPOSIUM ON FIELD-PROGRAMMABLE CUSTOM COMPUTING MACHINES (FCCM), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.197–197.

BJERREGAARD, T.; MAHADEVAN, S. A Survey of Research and Practices of Network-on-chip. **ACM Comput. Surv.**, New York, NY, USA, v.38, n.1, June 2006.

BJONTEGAARD, G. **Calculation of average PSNR differences between RD curves (VCEG-M33)**. Austin: [s.n.], 2001.

BOHR, M. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. **IEEE Solid-State Circuits Society Newsletter**, [S.l.], v.12, n.1, p.11–13, 2007.

BOKHARI, H. et al. darkNoC: Designing energy-efficient network-on-chip with multi-Vt cells for dark silicon. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p.1–6.

BOKHARI, H. et al. SuperNet: Multimode Interconnect Architecture for Manycore Chips. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 52., 2015, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2015. (DAC '15).

BOLOTIN, E.; CIDON, I.; GINOSAR, R.; KOLODNY, A. QNoC: QoS architecture and design process for network on chip. **JSA**, [S.l.], v.50, n.2, p.105 – 128, 2004. Special issue on networks on chip.

BORKAR, S. Thousand core chips: a technology perspective. In: OF THE 44TH ANNUAL DESIGN AUTOMATION CONFERENCE, 2007. **Proceedings...** [S.l.: s.n.], 2007. p.746–749.

BOSSEN, F. **Common test conditions and software reference configurations**. [S.l.: s.n.], 2013. JCTVC-L1100.

BOYAPATI, R. et al. Approx-noc: A data approximation framework for network-on-chip architectures. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 44., 2017. **Proceedings...** [S.l.: s.n.], 2017. p.666–677.

BOYCE, J. **HM16**: High Efficiency Video Coding Test Model (HM16) Encoder Description. [S.l.: s.n.], 2014. JCTVC-R1002, Sapporo.

CADENCE. **Encounter RTL Compiler**. Disponível em: <https://www.cadence.com/content/cadence-www/global/en_US/home/training/all-courses/84441.html>. Acesso em: 08 jul. 2020.

CASSA, M. B.; NACCARI, M.; PEREIRA, F. Fast rate distortion optimization for the emerging HEVC standard. In: PICTURE CODING SYMPOSIUM, 2012., 2012. **Anais...** [S.l.: s.n.], 2012. p.493–496.

CHATTERJEE, S.; SARAWADEKAR, K. Approximated Core Transform Architectures for HEVC Using WHT-Based Decomposition Method. **IEEE Transactions on Circuits and Systems I: Regular Papers**, [S.l.], v.66, n.11, p.4296–4308, 2019.

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>. Acesso em: 27 jun. 2020.

CLARK, M.; KODI, A.; BUNESCU, R.; LOURI, A. LEAD: Learning-enabled energy-aware dynamic voltage/frequency scaling in NoCs. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 55., 2018. **Proceedings...** [S.l.: s.n.], 2018. p.1–6.

CORRÊA, G. R. **Computational Complexity Reduction and Scaling for High Efficiency Video Encoders**. 2014. 252p. Tese (Doutorado em Engenharia Eletrotecnica e de Computadores) — University of Coimbra.

DALLY, W. J. et al. Virtual-channel flow control. **IEEE Transactions on Parallel and Distributed systems**, [S.l.], v.3, n.2, p.194–205, 1992.

DENNARD, R. H. et al. Design of ion-implanted MOSFET's with very small physical dimensions. **IEEE Journal of Solid-State Circuits**, [S.l.], v.9, n.5, p.256–268, 1974.

DIEFY, A.; SHALABY, A.; SAYED, M. S. Efficient architectures for HEVC luma interpolation filter. In: INTERNATIONAL CONFERENCE ON MICROELECTRONICS (ICM), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.9–12.

EHRLICH, P.; RADKE, S. Energy-aware software development for embedded systems in HW/SW co-design. In: IEEE DDECS, 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.232–235.

EL-HAROUNI, W. et al. Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), 2017, 2017. **Anais...** [S.l.: s.n.], 2017. p.1384–1389.

ESMAEILZADEH, H. et al. Dark silicon and the end of multicore scaling. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE (ISCA), 2011., 2011. **Anais...** [S.l.: s.n.], 2011. p.365–376.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. et al. Knowledge Discovery and Data Mining: Towards a Unifying Framework. In: KDD, 1996. **Anais...** [S.l.: s.n.], 1996. v.96, p.82–88.

GHANBARI, M. **Standard codecs**: Image compression to advanced video coding. [S.l.]: Iet, 2003. n.49.

GHOSH, A.; RAHA, A.; MUKHERJEE, A. Energy-Efficient IoT-Health Monitoring System using Approximate Computing. **Internet of Things**, [S.l.], v.9, p.100166, 2020.

GOIRI, I.; BIANCHINI, R.; NAGARAKATTE, S.; NGUYEN, T. D. Approxhadoop: Bringing approximations to mapreduce frameworks. In: TWENTIETH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.383–397.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital De Imagens**. 3.ed. São Paulo: Pearson Prentice Hall, 2010.

GRELLERT, M. **Machine Learning Mode Decision for Complexity Reduction and Scaling in Video Applications**. 2018. 196p. Tese (Doutorado em Ciência da Computação) — Universidade Federal do Rio Grande do Sul.

GRELLERT, M.; BAMPI, S.; ZATT, B. Complexity-scalable HEVC encoding. In: PICTURE CODING SYMPOSIUM (PCS), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.1–5.

GROIS, D. et al. Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders. In: PICTURE CODING SYMPOSIUM (PCS), 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.394–397.

GUESMI, A. et al. Defensive Approximation: Enhancing CNNs Security through Approximate Computing. **arXiv preprint arXiv:2006.07700**, [S.l.], 2020.

HALL, M. et al. The WEKA data mining software: an update. **ACM SIGKDD explorations newsletter**, [S.l.], v.11, n.1, p.10–18, 2009.

HE, G. et al. High-throughput power-efficient VLSI architecture of fractional motion estimation for ultra-HD HEVC video encoding. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.23, n.12, p.3138–3142, 2015.

HE, Z. et al. Framework of AVS2-video coding. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.1515–1519.

HENKEL, J.; KHDR, H.; PAGANI, S.; SHAFIQUE, M. New trends in dark silicon. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.1–6.

INDRUSIAK, L. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. **Journal of Systems Architecture**, [S.l.], v.60, n.7, p.553 – 561, 2014.

INDRUSIAK, L. S.; BURNS, A.; NIKOLIĆ, B. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.219–224.

ISO/IEC-JCT1/SC29/WG11. **High Efficiency Video Coding (HEVC) text specification draft 10**. doc JCTVC-L1003.ed. [S.l.: s.n.], 2013.

JANTSCH, A.; TENHUNEN, H. et al. **Networks on chip**. [S.l.]: Springer, 2003. v.396.

JERRAYA, A.; WOLF, W. **Multiprocessor systems-on-chips**. [S.l.]: Elsevier, 2004.

JONES, N. **How to stop data centres from gobbling up the world's electricity**. Disponível em: <https://www.nature.com/articles/d41586-018-06610-y>. Acesso em: 27 jun. 2020.

KALALI, E.; HAMZAOGLU, I. Approximate HEVC fractional interpolation filters and their hardware implementations. **IEEE Transactions on Consumer Electronics**, [S.l.], v.64, n.3, p.285–291, 2018.

KHUN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

KIASARI, A.; JANTSCH, A.; LU, Z. Mathematical Formalisms for Performance Evaluation of Networks-on-chip. **ACM Comput. Surv.**, New York, NY, USA, v.45, n.3, p.38:1–38:41, July 2013.

KIM, B.; KIM, J.; HONG, S.; LEE, S. A real-time communication method for wormhole switching networks. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (CAT. NO.98EX205), 1998., 1998. **Proceedings...** [S.l.: s.n.], 1998. p.527–534.

LAYEK, M. A. et al. Performance analysis of H.264, H.265, VP9 and AV1 video encoders. In: ASIA-PACIFIC NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (APNOMS), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p.322–325.

LIU, W.; LOMBARDI, F.; SHULTE, M. A Retrospective and Prospective View of Approximate Computing [Point of View. **Proceedings of the IEEE**, [S.l.], v.108, n.3, p.394–399, 2020.

LU, Y. et al. Hierarchical Classification for Complexity Reduction in HEVC Inter Coding. **IEEE Access**, [S.l.], v.8, p.41690–41704, 2020.

LUNG, C.-Y.; SHEN, C.-A. Design and implementation of a highly efficient fractional motion estimation for the HEVC encoder. **Journal of Real-Time Image Processing**, [S.l.], v.16, n.5, p.1541–1557, 2019.

MA, N.; ZOU, Z.; LU, Z.; ZHENG, L. Implementing MVC decoding on homogeneous NoCs: Circuit switching or wormhole switching. In: EUROMICRO INTERNATIONAL CONFERENCE ON PARALLEL, DISTRIBUTED, AND NETWORK-BASED PROCESSING, 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.387–391.

MAUNG, H.; ARAMVITH, S.; MIYANAGA, Y. Improved region-of-interest based rate control for error resilient HEVC framework. In: IEEE INTERNATIONAL CONFERENCE ON DIGITAL SIGNAL PROCESSING (DSP), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p.286–290.

MCCNANN, K. et al. **High Efficiency Video Coding (HEVC) Test Model 16 (HM 16) Improved Encoder Description (JCTVC-S1002)**. Strasbourg: [s.n.], 2014.

MENDIS, H. R.; AUDSLEY, N. C.; INDRUSIAK, L. S. Dynamic and static task allocation for hard real-time video stream decoding on NoCs. **Leibniz Transactions on Embedded Systems**, [S.l.], v.4, n.2, p.01–1, 2017.

MENDIS, H. R.; INDRUSIAK, L. S. Low communication overhead dynamic mapping of multiple HEVC video stream decoding on NoCs. In: WORKSHOP ON PARALLEL PROGRAMMING AND RUN-TIME MANAGEMENT TECHNIQUES FOR MANY-CORE ARCHITECTURES AND THE 5TH WORKSHOP ON DESIGN TOOLS AND ARCHITECTURES FOR MULTICORE EMBEDDED COMPUTING PLATFORMS, 7., 2016. **Proceedings...** [S.l.: s.n.], 2016. p.19–24.

MERCAT, A. et al. Probabilistic Approach Versus Machine Learning for One-Shot Quad-Tree Prediction in an Intra HEVC Encoder. **Journal of Signal Processing Systems**, [S.l.], v.91, n.9, p.1021–1037, 2019.

MERCAT, A.; VIITANEN, M.; VANNE, J. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In: ACM MULTIMEDIA SYSTEMS CONFERENCE, 11., 2020. **Proceedings...** [S.l.: s.n.], 2020. p.297–302.

MORAES, F. et al. HERMES: an infrastructure for low area overhead packet-switching networks on chip. **Integration**, [S.l.], v.38, n.1, p.69–93, 2004.

MUKHERJEE, D. et al. The latest open-source video codec VP9 - An overview and preliminary results. In: PICTURE CODING SYMPOSIUM, PCS 2013 - PROCEEDINGS, 2013., 2013. **Anais...** IEEE, 2013. p.390–393.

NI, L. M.; MCKINLEY, P. K. A survey of wormhole routing techniques in direct networks. **Computer**, [S.l.], v.26, n.2, p.62–76, 1993.

NOURI, S.; GHAZNAVI-YOUVALARI, R.; NURMI, J. Design and implementation of multi-purpose DCT/DST-specific accelerator on heterogeneous multicore architecture.

In: IEEE NORDIC CIRCUITS AND SYSTEMS CONFERENCE (NORCAS): NORCHIP AND INTERNATIONAL SYMPOSIUM OF SYSTEM-ON-CHIP (SOC), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–10.

PALUMBO, F. et al. Runtime energy versus quality tuning in motion compensation filters for HEVC. **IFAC-PapersOnLine**, [S.l.], v.49, n.25, p.145–152, 2016.

PASTUSZAK, G.; ABRAMOWSKI, A. Algorithm and architecture design of the H. 265/HEVC intra encoder. **IEEE Transactions on circuits and systems for video technology**, [S.l.], v.26, n.1, p.210–222, 2015.

PENNY, W. et al. Real-time architecture for HEVC motion compensation sample interpolator for UHD videos. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 28., 2015. **Proceedings...** [S.l.: s.n.], 2015. p.1–6.

PENNY, W. et al. Performance evaluation of HEVC RCL applications mapped onto NoC-based embedded platforms. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–6.

PENNY, W. et al. Design Space Exploration of HEVC RCL Mapped onto NoC-Based Embedded Platforms. In: INTERNATIONAL SYMPOSIUM ON RECONFIGURABLE COMMUNICATION-CENTRIC SYSTEMS-ON-CHIP (RECOSOC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p.1–8.

PENNY, W. et al. High-throughput and power-efficient hardware design for a multiple video coding standard sample interpolator. **Journal of Real-Time Image Processing**, [S.l.], v.16, n.1, p.175–192, 2019.

PENNY, W. et al. Low-Power and Memory-Aware Approximate Hardware Architecture for Fractional Motion Estimation Interpolation on HEVC. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS (ISCAS), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p.1–5.

PODDER, P. K.; PAUL, M.; MURSHED, M. A novel motion classification based inter-mode selection strategy for HEVC performance improvement. **Neurocomputing**, [S.l.], v.173, p.1211–1220, 2016.

PORTO, R. et al. Fast and energy-efficient approximate motion estimation architecture for real-time 4 K UHD processing. **Journal of Real-Time Image Processing**, [S.l.], p.1–15, 2020.

POURABED, M. A.; NOURI, S.; NURMI, J. Design and Implementation of 2D IDCT/IDST-Specific Accelerator on Heterogeneous Multicore Architecture. In: IEEE

NORDIC CIRCUITS AND SYSTEMS CONFERENCE (NORCAS): NORCHIP AND INTERNATIONAL SYMPOSIUM OF SYSTEM-ON-CHIP (SOC), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p.1–6.

PRABAKARAN, B. S.; EL-HAROUNI, W.; REHMAN, S.; SHAFIQUE, M. Approximate Multi-Accelerator Tiled Architecture for Energy-Efficient Motion Estimation. In: **Approximate Circuits**. [S.l.]: Springer, 2019. p.249–268.

PRAVEEN, G.; ADIREDDY, R. Analysis and approximation of SAO estimation for CTU-level HEVC encoder. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING (VCIP), 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.1–5.

PURNACHAND, N.; ALVES, L. N.; NAVARRO, A. Fast motion estimation algorithm for HEVC. In: IEEE SECOND INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS-BERLIN (ICCE-BERLIN), 2012., 2012. **Anais...** [S.l.: s.n.], 2012. p.34–37.

QUINLAN, J. **C4. 5**: programs for machine learning. [S.l.]: Elsevier, 2014.

RICHARDSON, I. **Video Codec Design**: Developing Image and Video Compression Systems. Chichester: John Wiley & Sons, Inc., 2002.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression**: Video Coding for Next-generation Multimedia. New York: John Wiley & Sons, Inc., 2003.

RUSSELL, S.; NORVIG, P. **Artificial intelligence**: a modern approach. [S.l.]: Prentice Hall, 2002.

SAU, C. et al. Challenging the best HEVC fractional pixel FPGA interpolators with reconfigurable and multifrequency approximate computing. **IEEE Embedded Systems Letters**, [S.l.], v.9, n.3, p.65–68, 2017.

SAYUTI, M.; INDRUSIAK, L. A Function for Hard Real-Time System Search-Based Task Mapping Optimisation. In: IEEE ISORC, 2015. **Anais...** [S.l.: s.n.], 2015. p.66–73.

SHI, Z.; BURNS, A. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In: IEEE/ACM NOCS, 2008. **Anais...** [S.l.: s.n.], 2008. p.161–170.

SILVA, R. da; SIQUEIRA, Í.; GRELLERT, M. Approximate interpolation filters for the fractional motion estimation in HEVC encoders and their VLSI design. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 32., 2019. **Proceedings...** [S.l.: s.n.], 2019. p.1–6.

SINGH, A. K.; DZIURZANSKI, P.; MENDIS, H. R.; INDRUSIAK, L. S. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. **ACM Computing Surveys (CSUR)**, [S.l.], v.50, n.2, p.1–40, 2017.

SMEI, H.; JEMAI, A.; SMIRI, K. Performance Estimation of HEVC/h.265 Decoder in a Co-Design Flow with SADF-FSM Graphs. **IJCNS**, [S.l.], v.10, p.261 – 281, 2017.

SMITH, S. **The Scientist and Engineer's Guide to Digital Signal Processing**. [S.l.]: California Technical Pub., 1997.

STATISTA. **Semiconductor market size worldwide from 1987 to 2020**. Disponível em: <https://www.statista.com/statistics/266973/global-semiconductor-sales-since-1988/>. Acesso em: 27 jun. 2020.

STATISTA. **Global mobile data traffic 2017-2022**. Disponível em: <https://www.statista.com/statistics/271405/globalmobile-data-traffic-forecast/>. Acesso em: 27 jun. 2020.

STATISTA. **Coronavirus impact on online traffic of selected industries worldwide in week ending April 26, 2020**. Disponível em: <https://www.statista.com/statistics/1105486/coronavirus-traffic-impact-industry/>. Acesso em: 27 jun. 2020.

SULLIVAN, G. J.; OHM, J. R.; HAN, W. J.; WIEGAND, T. Overview of the High Efficiency Video Coding (HEVC) Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.22, n.12, p.1649–1668, 2012.

SZE, V.; BUDAGAVI, M.; SULLIVAN, G. J. High efficiency video coding (HEVC). In: **Integrated circuit and systems, algorithms and architectures**. [S.l.]: Springer, 2014. v.39, p.49–90.

TARIQ, U. U.; WU, H.; ABD ISHAK, S. Energy and memory-aware software pipelining streaming applications on NoC-based MPSoCs. **Future Generation Computer Systems**, [S.l.], 2020.

TSMC. **40 nm Technology**. Disponível em: <https://www.tsmc.com/english/dedicatedFoundry/technology/40nm.htm>. Acesso em: 08 jul. 2020.

VANNE, J.; VIITANEN, M.; HAMALAINEN, T. D.; HALLAPURO, A. Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.22, n.12, p.1885–1898, 2012.

VENKATARAMANI, S.; CHAKRADHAR, S. T.; ROY, K.; RAGHUNATHAN, A. Computing approximately, and efficiently. In: DESIGN, AUTOMATION & TEST IN EUROPE CONFERENCE & EXHIBITION (DATE), 2015., 2015. **Anais...** [S.l.: s.n.], 2015. p.748–751.

VERIZON. **Verizon's Network COVID-19 Reliability Report**. Disponível em: <https://www.verizon.com/about/news/how-americans-are-spending-their-time-temporary-new-normal>. Acesso em: 27 jun. 2020.

VILLA, O. et al. Scaling the power wall: a path to exascale. In: SC'14: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS, 2014. **Anais...** [S.l.: s.n.], 2014. p.830–841.

VIZZOTTO, B. B. et al. A model predictive controller for frame-level rate control in multi-view video coding. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO, 2012., 2012. **Anais...** [S.l.: s.n.], 2012. p.485–490.

WANG, X. et al. Efficient Task Mapping for Manycore Systems. **arXiv preprint arXiv:2004.03462**, [S.l.], 2020.

XIONG, Q.; WU, F.; LU, Z.; XIE, C. Extending Real-Time Analysis for Wormhole NoCs. **IEEE Transactions on Computers**, [S.l.], v.66, n.9, p.1532–1546, 2017.

XU, Q.; MYTKOWICZ, T.; KIM, N. S. Approximate computing: A survey. **IEEE Design & Test**, [S.l.], v.33, n.1, p.8–22, 2015.

YEMLIHA, T. et al. Integrated code and data placement in two-dimensional mesh based chip multiprocessors. In: IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2008., 2008. **Anais...** [S.l.: s.n.], 2008. p.583–588.

YIN, P. et al. Design and Analysis of Energy-Efficient Dynamic Range Approximate Logarithmic Multipliers for Machine Learning. **IEEE Transactions on Sustainable Computing**, [S.l.], 2020.

ZHAN, J. et al. Designing energy-efficient NoC for real-time embedded systems through slack optimization. In: ACM/EDAC/IEEE DESIGN AUTOMATION CONFERENCE (DAC), 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p.1–6.

ZHENG, H.; LOURI, A. An energy-efficient network-on-chip design using reinforcement learning. In: ANNUAL DESIGN AUTOMATION CONFERENCE 2019, 56., 2019. **Proceedings...** [S.l.: s.n.], 2019. p.1–6.

ZHOU, C.; ZHOU, F.; CHEN, Y. Spatio-temporal correlation-based fast coding unit depth decision for high efficiency video coding. **JEI**, [S.l.], v.22, n.4, 2013.

Annexes

ANNEX A – List of Publications during this PhD

PENNY, WAGNER; PALOMINO, DANIEL; PORTO, MARCELO; ZATT, BRUNO. **Power/QoS-Adaptive HEVC FME Hardware using Machine Learning-Based Approximation Control**. In: 2020 IEEE International Conference on Visual Communications and Image Processing (VCIP), 2020.

PENNY, WAGNER; CORREA, GUILHERME; AGOSTINI, LUCIANO; PALOMINO, DANIEL; PORTO, MARCELO; NAZAR, GABRIEL; ZATT, BRUNO. **Low-Power and Memory-Aware Approximate Hardware Architecture for Fractional Motion Estimation Interpolation on HEVC**. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020.

DOMANSKI, R.; GOEBEL, JONES; **PENNY, WAGNER**; PORTO, MARCELO; PALOMINO, DANIEL; ZATT, BRUNO; AGOSTINI, LUCIANO. **High-Throughput Multifilter Interpolation Architecture for AV1 Motion Compensation**. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-EXPRESS BRIEFS, v. 66, p. 883-887, 2019.

PENNY, WAGNER; PALOMINO, DANIEL; PORTO, MARCELO; ZATT, BRUNO; INDRUSIAK, LEANDRO SOARES. **Design Space Exploration of HEVC RCL Mapped onto NoC-Based Embedded Platforms**. In: 2019 14th International Symposium on Reconfigurable Communicationcentric SystemsonChip (ReCoSoC), 2019, York. 2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2019. p. 1.

PENNY, WAGNER; GOEBEL, JONES; CORREA, DOUGLAS; MARTINS, ANDERSON; NAZAR, GABRIEL; AGOSTINI, LUCIANO; PALOMINO, DANIEL; PORTO, MARCELO; ZATT, BRUNO. **Energy-Efficiency Exploration of Memory Hierarchy using NVMs for HEVC Motion Estimation**. In: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019, Genoa. 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019. p. 162.

PENNY, WAGNER; PALOMINO, DANIEL; PORTO, MARCELO; ZATT, BRUNO; INDRUSIAK, LEANDRO. **Performance evaluation of HEVC RCL applications mapped onto NoC-based embedded platforms**. In: Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design, 2019, São Paulo. Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design (SBCCI). New York: ACM Press, 2019. p. 1-6.

PENNY, WAGNER; GOEBEL, JONES; PAIM, G. P.; PORTO, M. S.; AGOSTINI, L. V.; ZATT, B.. **High-throughput and power-efficient hardware design for a multiple video coding standard sample interpolator**. Journal of Real-Time Image Processing, v. 1, p. 1-18, 2018.

MARTINS, ANDERSON; **PENNY, WAGNER**; WEBER, MATHEUS; AGOSTINI, LUCIANO; PORTO, MARCELO; PALOMINO, DANIEL; MATTOS, JULIO; ZATT, BRUNO. **Configurable Cache Memory Architecture for Low-Energy Motion Estimation**. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, Florence. 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018. p. 1.

PENNY, WAGNER; UCKER, MARIANA; MACHADO, ITALO; AGOSTINI, LUCIANO; PALOMINO, DANIEL; PORTO, MARCELO; ZATT, BRUNO. **Power-Efficient and Memory-Aware Approximate Hardware Design for HEVC FME Interpolator**. In: 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018, Bordeaux. 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2018. p. 237.

MACHADO, I. D.; **PENNY, W. I.**; PORTO, M. S.; AGOSTINI, L. V.; ZATT, B.. **Characterizing Energy Consumption in Software HEVC Encoders: HM vs x265**. In: 8th Latin American Symposium on Circuits and Systems (LASCAS), 2017, Bariloche. Proceedings of 8th Latin American Symposium on Circuits and Systems (LASCAS), 2017.

MARTINS, ANDERSON; **PENNY, WAGNER**; WEBER, MATHEUS; PALOMINO, DANIEL; MATTOS, JULIO; PORTO, MARCELO; AGOSTINI, LUCIANO; ZATT, BRUNO. **Cache Memory Energy Efficiency Exploration for the HEVC Motion Estimation**. In: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), 2017, Curitiba. 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), 2017. p. 31. **Obs.: BEST PAPER AWARD of the conference**.

PENNY, WAGNER; MACHADO, ITALO; PORTO, MARCELO; AGOSTINI, LUCIANO; ZATT, BRUNO. **Pareto-based energy control for the HEVC encoder**. In: 2016 IEEE International Conference on Image Processing (ICIP), 2016, Phoenix. 2016 IEEE International Conference on Image Processing (ICIP). p. 814.

PAIM, GUILHERME; GOEBEL, JONES; **PENNY, WAGNER**; ZATT, BRUNO; PORTO, MARCELO; AGOSTINI, LUCIANO. **High-throughput and memory-aware hardware of a sub-pixel interpolator for multiple video coding standards**. In: 2016 IEEE International Conference on Image Processing (ICIP), 2016, Phoenix. 2016 IEEE International Conference on Image Processing (ICIP). p. 2162.

PAIM, GUILHERME; **PENNY, WAGNER**; GOEBEL, JONES; AFONSO, VLADIMIR; SUSIN, ALTAMIRO; PORTO, MARCELO; ZATT, BRUNO; AGOSTINI, LUCIANO. **An efficient sub-sample interpolator hardware for VP9-10 standards**. In: 2016 IEEE International Conference on Image Processing (ICIP), 2016, Phoenix. 2016 IEEE International Conference on Image Processing (ICIP). p. 2167.