FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Abordagem de computação heterogénea para reamostragem e redimensionamento de vídeo de alto desempenho

José Pedro Soares João Pereira

PREPARAÇÃO DA DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Jorge Manuel Gomes Barbosa

9 de Fevereiro de 2018

Abordagem de computação heterogénea para reamostragem e redimensionamento de vídeo de alto desempenho

José Pedro Soares João Pereira

Mestrado Integrado em Engenharia Informática e Computação

9 de Fevereiro de 2018

Resumo

A crescente popularidade dos canais de comunicação de televisão e plataformas de streaming motiva a publicação de novo conteúdo multimédia. De modo a alcançar e a agradar o maior número de consumidores, este conteúdo deve ser criado o mais rapidamente possível com a melhor qualidade disponível. Os conteúdos de multimédia de elevada qualidade são obtidos após a aplicação de várias operações de pós-produção, o que faz surgir um desafio nesta fase pois as imagens de alta qualidade são constituídas por uma quantidade elevada de pixeis que reclamam uma capacidade de recursos computacionais diretamente proporcionais ao seu tamanho para serem processadas visto que as operações de produção são aplicadas ao nível dos pixeis de uma imagem. As operações de processamento de imagem realizadas na fase de pós-produção constituem um fator de importância pois é necessário assegurar que o tempo de execução da sua aplicação é igual ou inferior ao tempo da captura da imagem de modo a garantir que os conteúdos de multimédia estejam preparados a serem exibidos o quanto antes possível. Estas restrições de performance tornam a aplicação das operações da fase de pós-produção extremamente difíceis de serem realizadas utilizando apenas as valências dos processadores. Atualmente, devido ao progresso da tecnologia contida nas placas de processamento gráfico existe uma maior capacidade de processamento em relação à capacidade dos processadores, tipicamente, aliada a uma utilização mais eficiente de energia o que torna a utilização de uma abordagem de computação heterogénea, isto é, uma abordagem que combina as capacidades de processamento de um processador e a placa gráfica de uma máquina, ideal para o desenvolvimento de software de alto desempenho.

O objetivo deste trabalho consiste em analisar e implementar uma abordagem de computação heterogénea de parte das fases de pós-produção recorrendo a técnicas de paralelização e aceleração de hardware para o efeito. As fases de pós-produção de imagem mencionadas são a fase de reamostragem, conversão entre diferentes modelos de cor utilizados na representação dos pixeis, e a fase de redimensionamento, alteração das dimensões de resolução de uma imagem. Este projeto analisa os principais detalhes das soluções atuais para realizar as fases de reamostragem e redimensionamento de imagens, e as diferentes implementações de abordagens de computação paralela e heterogénea.

Espera-se que o resultado obtido seja a implementação de uma ferramenta de aplicação das referidas fases de pós-produção de imagem que tenha em conta as restrições de processamento em tempo real, recorrendo à paralelização do sistema e a aceleração gráfica para o efeito.

Abstract

The increasing popularity of communication channels such as television broadcasters and streaming platforms motivates the issue of new multimedia content. In order to reach and please the most customers, this content should be created as fast as possible with the best quality available. High quality multimedia content is obtained through the application of various post-production operations which arises a challenge during this phase since high quality images are composed by an high number of pixels which require computational resources directly proportional to the their size as these production operations are applied at the image's pixels level. The image processing operations performed during the post-production phase constitutes an important factor since it is needed to ensure that the execution time of their application is equal or less than the time of capturing the image in order to guarantee that the multimedia content are available to be displayed as soon as possible. These performance restrictions render the application of the post-production operations extremely difficulty to be done only using the processors capacities. Nowadays, due to the advance of the technology contained in the graphics processing units there is a greater processing capability in relation to the processing capability of processors, typically, conjugated with a better power usage efficiency which makes the use of a heterogeneous computing approach, the combined use of the processing capabilities of processors and graphics cards, ideal to the development of high performance software.

The goal of this work is to analyze and implement an heterogeneous computing approach of the post-production phases using parallelization techniques and hardware acceleration. The mentioned post-production phases are the resampling phase, the phase of conversion between different color spaces used in the representation of the pixel, and the resizing phase, the phase of alteration of the image resolution dimensions. This project analyze the main details of the current solutions that perform the resampling and resizing phases of image, and the different implementation of parallel and heterogeneous computing approaches.

It is expected that this project's result is an implementation of tool that performs the mentioned image post-production phases which takes into account the restrictions of real time processing, resorting to the parallelization of the system and graphic acceleration in order to do so.

Conteúdo

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

LISTA DE TABELAS

Abreviaturas e Símbolos

ALU	Arithmetic Logic Unit
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPGPU	General Purpose Computing On Graphics Processing Units
GPU	Graphics Processing Unit
MIMD	Multiple Instructions Multiple Data Streams
MISD	Multiple Instructions Single Data Stream
MPI	Message Passing Interface
RAM	Random Access Memory
SIMD	Single Instruction Multiple Data Streams
SISD	Single Instruction Single Data Stream

Capítulo 1

Introdução

A sociedade do século vinte e um tem sofrido uma drástica alteração cultural, económica e social devido à utilização das novas tecnologias dos dispositivos eletrónicos como computadores pessoais, smartphones, e tablets, que aliada à utilização da internet permite que qualquer indivíduo possa aceder, visualizar, modificar, armazenar e partilhar conteúdos de multimédia com um alcance virtualmente global. Por esta razão as indústrias de jornalismo, educação, entretenimento, a indústria editorial e de música têm sofrido significativas transformações para acompanhar a tendência, levando à utilização de meios de comunicação diferentes dos considerados convencionais antigamente de modo a atingir os seus objetivos, como a utilização de plataformas de streaming e canais de televisão. Todo o mercado de criação de conteúdos multimédia, constituído maioritariamente pelas indústrias referidas, está direcionado para a publicação de novo conteúdo multimédia o mais rapidamente possível com a melhor qualidade disponível culminando numa vantagem sobre os outros negócios do mercado.

Desde a invenção dos primeiros computadores, o poder de processamento e as capacidades de armazenamento de informação têm crescido exponencialmente. Com a evolução da tecnologia surgiram diferentes abordagens para realizar os processos de pós-produção de vídeo necessários para gerar conteúdos de multimédia de elevada qualidade. A computação paralela e computação heterogénea são duas das abordagens que podem ser tidas em conta para a aplicação destes processos, pois estas abordagens permitem a execução concorrente dos mecanismos de um sistema utilizando as capacidades de processamento dos processadores e das placas de processamento gráfico de uma máquina de forma a acelerar a aplicação das operações de pós-produção de vídeo permitindo os melhores níveis de qualidade dos resultados em tempo real.

1.1 Contexto

O tema desta dissertação foi proposto pela Mog Technologies, especializada no desenvolvimento de novas plataformas tecnológicas para colocar novos produtos, sistemas e soluções ino-

vadoras que automatizam os processos de trabalho dos operadores de conteúdos de multimédia profissional, nomeadamente, produtoras de vídeo e estações de televisão. Um destes processos de trabalho é o processo de pós-produção, automatizado através do desenvolvimento de soluções de ferramentas de Ingest que manipulam e transportam um vídeo desde um ponto de origem até a um ponto de destino da cadeia de produção.

Figura 1.1: Logótipo da Mog Technologies

A evolução das áreas de computação paralela e computação heterogénea permitem a aplicação destas tecnologias na área das soluções de vídeo de forma a obter um melhor desempenho dos sistemas desenvolvidos e atender às restrições, cada vez mais rigorosas, impostas pela indústria.

1.2 Motivação

Os processos de reamostragem e redimensionamento de imagem são dois dos processos necessários de serem realizados durante a fase de pós-produção de um vídeo. As imagens dos vídeos sem qualquer tipo de compressão, em formato *raw*, são constituídos por um elevado número de pixeis. Uma imagem na resolução *Full HD*, com uma dimensão de 1920 por 1080 pixeis, é constituída, aproximadamente, por dois milhões de pixeis. Atualmente, são cada vez mais utilizadas resoluções superiores ao *Full HD*, como é o caso do *Ultra HD*, designadamente as resoluções *4K* e *8K*, aproximadamente constituídas por, respetivamente, nove milhões e trinta e cinco milhões de pixeis.

Atendendo a que os processos de pós-produção mencionados operam ao nível de cada pixel que constituem uma imagem, a capacidade de processamento necessária para os aplicar é diretamente proporcional à resolução da imagem. De forma a que os conteúdos de multimédia estejam disponíveis o quanto antes possível, idealmente, a aplicação dos processos de pós-produção devem ser realizados em tempo real, isto é, o processamento de uma imagem de um vídeo deve ser realizado antes da captura da imagem seguinte. Por estes motivos, a aplicação dos processos de pós-produção, respeitando as restrições da indústria, torna-se cada vez mais difícil de serem efetuadas sem que haja uma nova análise e implementação das técnicas utilizadas até ao momento para o efeito.

1.3 Objetivos

O objetivo deste projeto é a análise das técnicas utilizadas durante a realização da fase de pós-produção de vídeo e implementação de uma ferramenta para a aplicação dos processos de reamostragem e redimensionamento de imagem recorrendo a uma abordagem de computação heterogénea. Enumera-se os seguintes objetivos:

- Analisar as técnicas utilizadas no processo de reamostragem de imagem;
- Estudar os diferentes algoritmos de redimensionamento de imagem;
- Implementação sequencial dos processos de reamostragem e redimensionamento de imagem;
- Paralelização da implementação sequencial dos processos de reamostragem e redimensionamento de imagem recorrendo a uma abordagem de computação heterogénea, utilizando aceleração gráfica;
- Comparação da solução desenvolvida com outras ferramentas já existentes para o efeito.

1.4 Estrutura do Relatório

Este relatório contém cinco capítulos, incluindo este capítulo introdutório, contextualizando o problema. No capítulo seguinte, capítulo 2 intitulado de "Computação Paralela e Computação Heterogénea", são expostos os principais detalhes das arquiteturas dos processadores utilizados nos sistemas que abordam as áreas de computação paralela e computação heterogénea, uma descrição dos principais conceitos da área de sistemas paralelos e a análise das ferramentas e plataformas de desenvolvimento utilizadas para implementar as abordagens referidas.

No seguinte capítulo, capítulo 3 intitulado de "Processamento de Vídeo", serão analisadas as técnicas e os algoritmos respetivamente utilizados nos processos de reamostragem e redimensionamento de imagem, e uma descrição dos mesmos.

Por último, o capítulo 4 de conclusão aborda as principais conclusões alcançadas no desenvolvimento deste trabalho, a solução propostas e os resultados esperados, e as tarefas a serem realizadas durante a próxima fase deste projeto, assim como um plano de trabalho para as executar.

Capítulo 2

Computação Paralela

Neste capítulo são analisadas as componentes que constituem as unidades de processamento central e as unidades de processamento gráficas dos computadores atuais, descrição de alguns conceitos da área de sistemas paralelos, a comparação entre os dois tipos de processadores frequentemente utilizados em sistemas implementados através de uma abordagem de computação heterogénea, descrição de técnicas e metodologias de paralelização de soluções, e uma análise das ferramentas e plataformas de desenvolvimento utilizadas atualmente na implementação de sistemas de computação paralela e computação heterogénea.

2.1 Unidades de Processamento

Os computadores seguem a arquitetura desenvolvida por John von Neuman, segundo esta arquitetura, uma máquina é dividida em três componentes principais sendo estes: a unidade de processamento central, ou CPU, a memória de acesso aleatório, ou RAM, e a interface de entrada e saída.

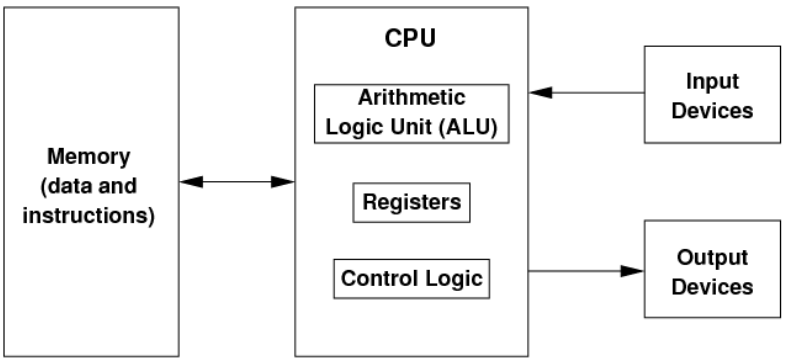

Figura 2.1: Diagrama de arquitetura de von Neuman [1]

A unidade de processamento central é o componente de um computador responsável pela execução de instruções de um determinado programa. O CPU é constituído pela unidade de lógica e aritmética, ou ALU, um circuito responsável pela execução de operações aritméticas e lógicas, pelos registos do processador, encarregado por fornecer operandos à ALU e o armazenamento dos resultados das operações, e a unidade de controlo, responsável pela coordenação entre a recuperação dos dados armazenados a partir da memória e a execução das operações realizadas pela ALU, registos do processador e outras componentes. A RAM armazena as instruções a serem executadas e os dados necessário para as realizar. A interface de entrada e saída faz a ligação entre a RAM e o CPU podendo também criar a ligação com outro tipo de periféricos ou hardware.

2.1.1 Evolução dos CPUs

A lei de Moore, criada por Gordon Moore, co-fundador da Intel, é a observação que o número de transístores de um circuito integrado duplica a cada dois anos. Os transístores permitem criar circuitos complexos cujo objetivo podem ser a execução de operações aritméticas e lógicas. Quanto maior for o número de transístores de um circuito integrado, como por exemplo um processador, maior será o número dos circuitos complexos destinados a realizarem as operações aritméticas e lógicas, e por consequência maior será a capacidade de executar instruções por um processador. Por esta razão, a capacidade de processamento de um CPU está intrinsecamente relacionada com a lei de Moore.

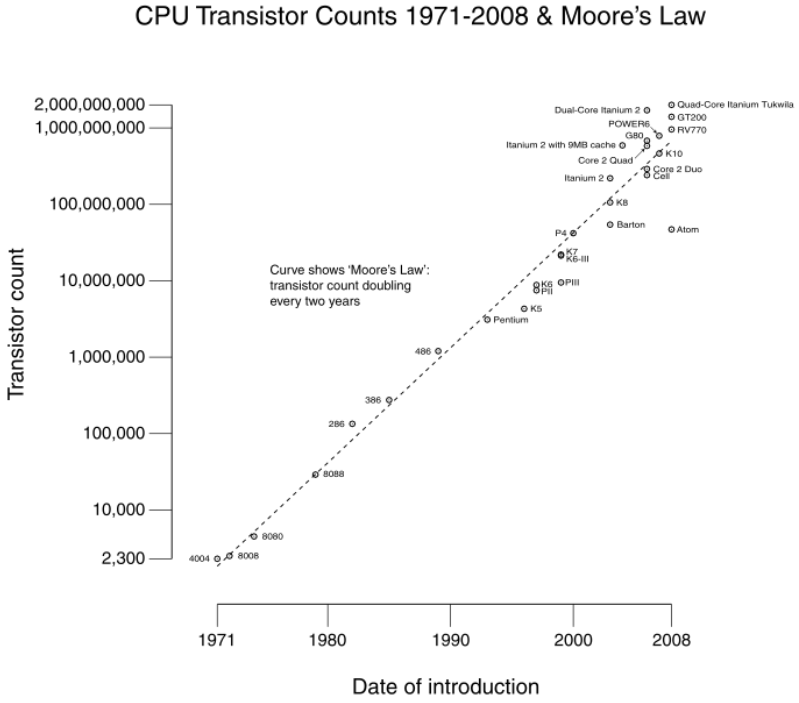

Figura 2.2: Relação entre a evolução dos processadores e a lei de Moore [2]

A utilização de um maior número de transístores implica um maior consumo de energia por parte de um processador. Até 2004, a abordagem utilizada para aumentar a capacidade de processamento de um CPU era a integração de um maior número de transístores no circuito, de modo a aumentar a frequência de relógio que dita a cadência de execução das instruções a serem realizadas. O aumento da frequência de um processador causa um maior consumo de energia devido à dissipação de calor causada. A cada aumento de 400 MHz na frequência de execução de instruções de um CPU existe um incremento de 60% da energia consumida [4]. Esta abordagem tornou-se inviável visto que o custo da energia consumida não compensa o ganho de performance do processador. Por estas circunstâncias outro tipo de abordagem foi tida em conta, a arquitetura *multi-core*.

2.1.2 Arquiteturas Paralelas

As arquiteturas dos computadores definem o modo como as unidades de processamento trabalham e como a memória é organizada. Atualmente, as arquiteturas dos computadores evoluíram para máquinas paralelas devido ao limite prático da frequência de um CPU. As arquiteturas paralelas dos computadores podem ser classificadas relativamente ao seu modo de funcionamento paralelo, podem ser classificadas por aspetos como o tipo de instruções / *data streams*, organização da memória e as comunicações do processador. Esta classificação pode ser realizada através da taxonomia de Flynn [5]:

- SISD, ou *single instruction single data stream*, um único processador processa um único elemento dos dados por unidade de tempo. Este tipo de arquitetura encontra-se presente em micro-controladores e antigos computadores pessoais;
- SIMD, ou *single instruction multiple data streams*, uma mesma instrução é aplicada a múltiplos elementos de dados a cada unidade de tempo. Esta arquitetura é caraterística de processadores vetoriais;
- MISD, ou *multiple instructions single data stream*, um conjunto de unidades de processamento, ligadas sequencialmente, realizam diferentes operações sobre os mesmos dados. *Systolic arrays* são um exemplo de aplicação desta arquitetura;
- MIMD, ou *multiple instructions multiple data streams*, por unidade de tempo, cada processador, contendo uma unidade de controlo independente, pode executar diferentes instruções de um programa. Arquitetura presente na maioria das máquinas multi-processadores e multi-computadores.

2.1.3 Hierarquia de Memória

Na arquitetura de um computador, os componentes destinados ao armazenamento de dados são ordenados hierarquicamente com base no seu tempo de resposta [6]. A hierarquia de memória afeta o desempenho de um computador, de tal modo que a implementação de um programa de alto

desempenho deve considerar as capacidades e limitações de cada um dos componentes de memória utilizados. As componentes de memória presentes num computador atual são apresentadas, por ordem crescente de tempo de resposta e capacidade de armazenamento:

- Registos do processador - Acesso de memória mais rápido da arquitetura de um computador, cerca de um ciclo do CPU, mas de fraca capacidade de armazenamento;
- Memória cache - É a componente de memória mais próxima do processador que armazena fragmentos da memória principal que são utilizados frequentemente. A maioria dos computadores têm diferentes níveis deste tipo de memória que diferem segundo a proximidade ao núcleo do processador, capacidade de armazenamento e tempo de resposta de acesso a memória;
- Memória principal - Componente mais utilizado para armazenamento de dados devido à sua capacidade de armazenamento de dados, contudo é a componente de memória com maior tempo de resposta de acesso a memória dos apresentados;
- Memória externa - Este tipo de memória não está imediatamente disponível a ser utilizado por um computador visto que necessita de interação humana para ser utilizado. Disquetes, discos compactos e memórias flash USB são dispositivos considerados memórias externas.

2.1.4 Arquitetura *Multi-core*

Um processador *multi-core* segue a arquitetura MIMD, ou *multiple instructions multiple data stream*, isto é, segundo esta arquitetura o CPU realiza várias operações sobre diferentes dados e por isso é capaz de executar um programa em paralelo. Um processador multi-core é constituído por várias unidades de processamento contendo, cada uma, unidades de lógica e aritmética, e unidades de controlo independentes.

2.2 Unidades de Processamento Gráfico

Uma unidade de processamento gráfico, também designada de GPU, é um circuito especializado em alterar e manipular a memória de maneira a que o processamento de imagens a serem apresentados num dispositivo de exibição seja acelerado. Um GPU tem uma estrutura altamente paralela em comparação com os processadores, por essa razão as unidades de processamento gráfico são excecionalmente eficientes no processamento de blocos de informação em paralelo.

2.2.1 Arquitetura *Many-core*

Um processador *many-core* é um processador *multi-core* especializado, projetado para realizar processamento paralelo de menor consumo energético à custa de latência de comunicação entre os núcleos do processador, ou *core*, e de fraco desempenho singular dos processadores.

2.2.2 CPU vs GPU

Os processadores modernos seguem a arquitetura MIMD, são capazes de realizar processamento paralelo. Grande parte da área de superfície dos circuitos dos processadores são ocupadas pelas unidades de controlo e a memória cache, deixando apenas uma pequena área para as unidades de lógica e aritmética, por conseguinte, não têm uma elevada capacidade de realizar operações numéricas. O CPU realiza funções muito díspares e ter mecanismos de controlo e memória caches mais avançadas é o único mecanismo que permite o seu bom desempenho.

As unidades de processamento gráfico seguem a arquitetura SIMD, ou *single instruction multiple data streams*, isto é, segundo esta arquitetura o GPU aplica a mesma operação sobre diferentes fragmentos dos dados por unidade de tempo. O objetivo principal da arquitetura do GPU é alcançar um bom desempenho através de paralelismo extremo. Ao contrário do CPU, a área de superfície dos circuitos das unidades de processamento gráfico é maioritariamente constituída por unidades de lógica e aritmética, e apenas uma pequena região é constituída para as unidades de controlo e memória cache [7].

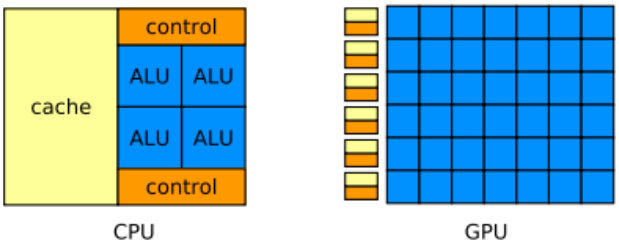

Figura 2.3: Diferenças de arquitetura entre um CPU e um GPU

As diferenças arquiteturais destas unidades de processamento têm um impacto direto na forma como estas unidades desempenham. O GPU é mais restrito em termos de instruções executadas do que o CPU mas tem um maior potencial para realizar operações numéricas e executar programas que foram especificamente desenvolvidos para serem executados neste tipo de unidades. Outra diferença técnica entre este dois tipos de processadores é a alta largura de banda das unidades de processamento gráfico em relação ao CPU. Um GPU tem uma capacidade de comunicação interna, entre os seus núcleos de processamento, superior a 100 gigabytes por segundo em relação aos 10 gigabytes por segundo do CPU. Esta diferença é justificada pela presença de um componente de memória adicional nas unidades de processamento gráfico que se comporta como uma memória de acesso aleatório para os núcleos de processamento do GPU. O valor de largura de banda do GPU permite uma menor latência de comunicação de dados pelos seus núcleos de processamento o que leva a um melhor desempenho de execução.

A relação entre o CPU e o GPU é um balanço entre a flexibilidade de funcionamento e a capacidade computacional. Os processadores modernos têm dificuldades em manter o balanço entre o poder computacional e o propósito geral de execução das diferentes funções que realizam. Por outro lado, as unidades de processamento gráfico proporcionam o máximo de poder computacional sofrendo das restrições que esse poder implica. Algumas destas restrições podem ser ultrapassadas

durante a implementação do programa a ser executado no GPU, uma boa utilização de técnicas de paralelização de problemas é essencial para tirar partido deste tipo de unidades de processamento.

2.3 Computação Paralela

Computação paralela é a designação atribuída ao mecanismo segundo o qual vários diferentes processadores executam ou processam uma determinada computação em simultâneo. A utilização de computação paralela permite realizar computações complexas dividindo a carga de trabalho por mais de uma unidade de processamento, todas as unidades de processamento realizam as operações ao mesmo tempo.

De modo a ser possível dividir um problema em diferentes partes, é essencial identificar o tipo de problema antes da formulação da solução paralela [7]. Se P_D é um problema de domínio D, e P_D é paralelizável então D pode ser decomposto em k sub-problemas:

$$D = d_1 + d_2 + ... + d_k = \sum_{i=1}^{k} d_i \tag{2.1}$$

P_D é um problema de paralelização de dados se D é composto por elementos de dados e o problema é solucionado através da aplicação de uma determinada função $f()$ a todo o domínio:

$$f(D) = f(d_1) + f(d_2) + ... + f(d_k) = \sum_{i=1}^{k} f(d_i) \tag{2.2}$$

P_D é um problema de paralelização funcional se D é composto por diferentes operações e o problema é solucionado através da aplicação de cada operação a um mesmo conjunto de dados S:

$$D(S) = d_1(S) + d_2(S) + ... + d_k(S) = \sum_{i=1}^{k} d_i(S) \tag{2.3}$$

Problemas de paralelização de dados são ideais para serem solucionados recorrendo às potencialidades do GPU, pois a sua arquitetura SIMD é ideal para a resolução de problemas cuja solução é obtida através da aplicação da mesma instrução por parte de todos os núcleos de processamento em fragmentos de dados diferentes. Por outro lado, problemas de paralelização funcional são ideias para serem solucionados recorrendo às potencialidades do CPU, pois a arquitetura do CPU permite a execução de diferentes tarefas em cada *core*.

2.3.1 Modelos de Programação paralela

Existem dois modelos de programação paralela que indicam a interação entre a memória de uma máquina e as suas unidades de processamento. Segundo o modelo de memória partilhada, todos os *threads*, isto é, os processos que estão a ser executados nos núcleos de processamento partilham o mesmo espaço de memória interagindo entre si por variáveis partilhadas [8]. Com este modelo, os *threads* podem executar assincronamente não havendo necessidade de sincronizar os dados utilizados, pois estes dados são transversais a todos os *threads*, contudo poderá ser

necessária a implementação de uma secção crítica com mecanismos de controlo caso algum dos *threads* necessite de acesso exclusivo à memória. No modelo de memória partilhada cada *thread* é constituído pelo seu próprio estado interno e as variáveis globais partilhadas definidas pelo *thread* principal.

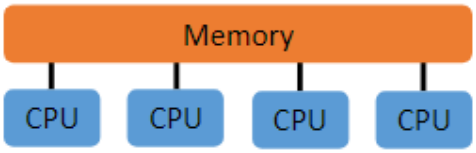

Figura 2.4: Modelo de memória partilhada

Quando um dos núcleos acede um espaço da memória, provoca que o espaço de memória circundante a esse local, designado de linha de cache, seja copiado para a memória cache do núcleo. Referências e acessos, subsequentes, ao mesmo espaço de memória ou a dados da mesma linha de cache podem ser realizados sem recorrer à memória principal do sistema. A linha de cache é preservada na memória cache até que o sistema determine que é necessário restituir a coerência entre esta memória e a memória principal. As alterações individuais de elementos da mesma linha de cache, realizadas por diferentes *threads*, invalida toda essa linha. A cada alteração a linha de cache é marcada como inválida e todos os outros processos que contêm a mesma linha de cache recebem o mesmo estatuto. O sistema obriga que estes processos resgatem a versão mais recente da linha de cache inválida a partir da memória principal, isto porque a coerência de memória é baseada na alteração da linha de cache e não de elementos individuais de memória [9]. Esta situação é denominada de *false sharing* e provoca um aumento do tráfico de comunicações dos *threads* e um maior *overhead*, o que diminui consideravelmente o desempenho de um sistema.

O segundo modelo de programação paralela é o modelo de memória distribuída em que cada núcleo de processamento tem uma componente de memória independente dos restantes interagindo entre si por de troca de mensagens utilizando a rede interna do processador, fazendo surgir a necessidade de realizar operações de sincronização dos dados utilizados através de mensagens entre cada *thread*.

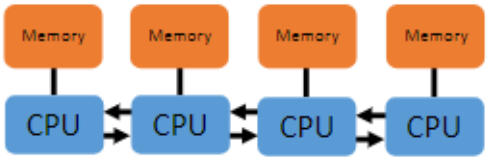

Figura 2.5: Modelo de memória distribuída

2.3.2 Extração de Paralelismo

Ian Foster sugeriu uma metodologia utilizada para extrair paralelismo de um determinado problema e planear uma solução para o mesmo, maximizando o número de opções viáveis para o fazer e diminuindo o custo de retroceder na fase de desenvolvimento para resolver possíveis problemas que possam surgir [10]. Esta metodologia permite ao programador focar-se nos problemas não dependentes da máquina onde será executado o sistema na fase inicial do processo de modelação, problemas como a concorrência da solução. Esta metodologia divide o processo de planeamento da solução em quatro fases distintas: divisão em partes ou *partitioning*, comunicação, aglomeração e mapeamento.

2.3.2.1 Divisão em Partes ou Partitioning

A fase de divisão em partes refere-se a decomposição das atividades computacionais a serem realizadas e dos dados que serão operados em tarefas de dimensão inferior. A decomposição dos dados associados com o problema é designada de *domain/data decomposition* ou decomposição do domínio/dados, e a decomposição das atividades computacionais a serem realizadas em tarefas disjuntas é designada de *functional decomposition* ou decomposição funcional.

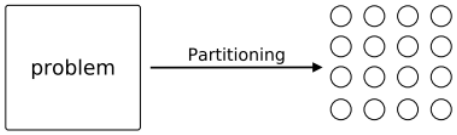

Figura 2.6: Fase de divisão do problema em partes

2.3.2.2 Comunicação

A fase de comunicação concentra-se na criação do fluxo de informação e coordenação das tarefas criadas na fase de divisão do problema em partes ou *partitioning* através da instauração de canais de comunicação entre as diferentes divisões. A especificidade do problema e o método de decomposição realizado determina o padrão de comunicação entre as tarefas do programa paralelo.

Figura 2.7: Fase de criação dos canais de comunicação

2.3.2.3 Aglomeração

A fase de aglomeração refere-se a fase do planeamento do sistema paralelo em que as tarefas idealizadas durante a fase da *partitioning* são agrupadas de modo a que a carga de trabalho necessária para as executar compensa a ocupação de um *thread*. Este aspeto é usualmente qualificado de granularidade do problema. Um problema de granularidade fina pode ser dividido num elevado número de pequenas tarefas a serem realizadas, tendo este tipo de problemas um alto potencial de serem paralelizáveis e um maior *overhead* para realizar as comunicações. Um problema de granularidade grossa pode ser dividido num número reduzido de tarefas de grande dimensão, tendo este tipo de problemas uma solução fracamente paralelizável mas de menor *overhead* durante as comunicações.

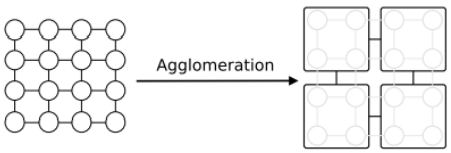

Figura 2.8: Fase de aglomeração de tarefas por *threads*

2.3.2.4 Mapeamento

A fase de mapeamento refere-se à atribuição dos grupos de tarefas, correspondentes às tarefas que serão realizadas por cada *thread*, realizada durante a fase de aglomeração, aos núcleos de processamento da máquina onde será executada a solução. O mapeamento é a última fase da metodologia de Foster e pode ser conseguida através de várias estratégias de atribuição dos grupos resultado da fase de aglomeração aos núcleos de processamento disponíveis. Estas estratégias podem ser um simples mapeamento de um para um, isto é, cada grupo de tarefas é atribuído a um só núcleo de processamento, ou um mapeamento mais complexo que pode provocar maior *overhead* das comunicações e um desequilíbrio da carga de trabalho de cada núcleo. O objetivo desta fase é encontrar o melhor equilíbrio da complexidade do mapeamento realizado para o problema em análise.

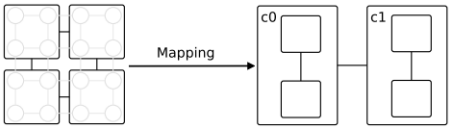

Figura 2.9: Fase de mapeamento de grupos de tarefas por núcleos de processamento

2.3.3 Computação Heterogénea

Computação heterogénea refere-se a sistemas que utilizam mais que um tipo de processadores. Este tipo de sistemas têm alto nível de desempenho e eficiência de consumo de energia adicionando processadores especializados para diferentes tipos de objetivos. A arquitetura de sistemas que implementam uma abordagem de computação heterogénea é, habitualmente, constituída por CPUs e unidades de processamento gráfico. Este tipo de sistemas encarrega a execução de tarefas de computação paralela intensiva ao GPU enquanto o resto do programa é atendido e executado pelo CPU.

2.4 Ferramentas e Plataformas de Desenvolvimento

Nesta secção serão apresentadas, descritas e comparadas as ferramentas e plataformas de desenvolvimento frequentemente utilizadas atualmente na implementação de sistemas que abordagem as arquiteturas de computação paralela e computação heterogénea.

2.4.1 Para Computação Paralela

2.4.1.1 OpenMP

A ferramenta OpenMP é uma API, ou *application programming* interface, multi-plataforma para as linguagens de programação C, C++ e Fortran, cuja utilização permite paralelizar através de um mecanismo de *multi-threads* uma determinada solução. *Multi-thread* é o mecanismo que permite a utilização das capacidades de processamento de todos os cores de um CPU. Com *multi-threads* é possível realizar diversos processos independentes concorrentemente com total suporte do sistema operativo [11]. O OpenMP é uma ferramenta destinada ao modelo de programação paralela de memória partilhada implementando o paralelismo do sistema através de diretivas do compilador *pragma* recorrendo a um modelo de execução paralela nomeado de *fork-join*.

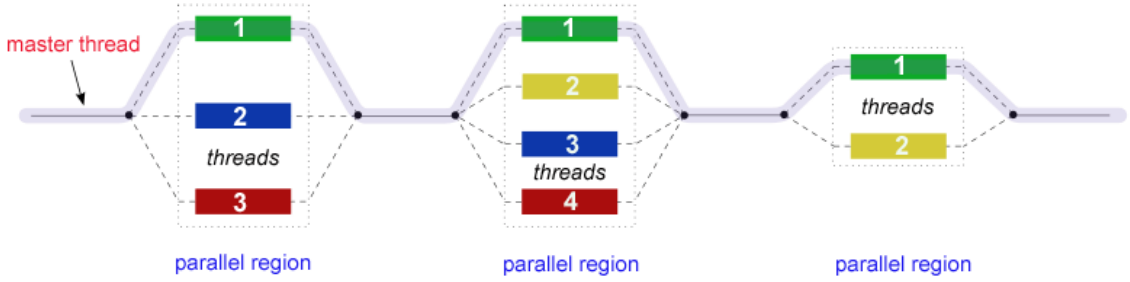

Figura 2.10: Modelo de programação paralela *fork-join* utilizado pelo OpenMP

A execução de um sistema paralelo implementado com OpenMP é realizada a partir de um *thread* principal, designado de *master thread*. As diretivas de compilador presentes nas instruções a serem executadas criam uma região paralela que será executada por diversos *threads*. Contudo,

no começo do programa apenas está ativo um *thread*, considerado o *thread* principal do sistema, também designado de *master thread*. O *master thread* ativa *threads* assim que as instruções a serem executadas apresentem as diretivas de compilador que o indiquem.

2.4.1.2 OpenMPI

O OpenMPI é uma interface de programação para sistemas operativos baseados em Unix para as linguagens de programação C, C++ e Fortran, cuja utilização permite paralelizar um sistema através do modelo de programação de memória distribuída. A ferramenta OpenMPI utiliza o standard MPI [12] que especifica os métodos necessários para implementar um sistema paralelo de memória distribuída. Esta interface implementa os mecanismos de identificação e aglomeração por grupos de processos, e os mecanismos de difusão e centralização de dados.

2.4.2 Para Computação Heterogénea

2.4.2.1 OpenCL

O OpenCL é uma plataforma de desenvolvimento de software para o desenvolvimento de sistemas de computação heterogénea, constituídos por várias unidades de processamento gráfico e CPUs. A plataforma OpenCL fornece uma interface de métodos necessários para a implementação de sistemas paralelos cujo problema é a paralelização de dados e a paralelização funcional da solução [13]. O OpenCL pode ser executado em sistemas operativos baseados em Unix e Windows, e integrado com as linguagens de programação C e C++. Esta plataforma de desenvolvimento permite a criação de métodos que se baseiam em tarefas, nomeados de *kernels*, e a execução dos mesmos sobre diferentes dados por cada núcleo de processamento. Cada novo *kernel* é colocado numa fila de espera, sendo executado logo que a unidade de processamento tenha disponibilidade para o fazer.

2.4.2.2 CUDA

O CUDA, ou *Compute Unified Device Architecture*, é uma plataforma de computação paralela e uma API criada pela Nvidia para o desenvolvimento de sistemas paralelos recorrendo a aceleração gráfica. O CUDA permite tirar partido das capacidades de processamento das placas gráficas para a execução de programas de propósito geral, do tipo GPGPU, ou *General Purpose computing on Graphics Processing Units*. A plataforma CUDA pode ser executado em sistemas operativos baseados em Unix e Windows através das linguagens de programação C, C++ e Fortran, logo que estes sistemas contenham uma unidade de processamento gráfico da Nvidia. O CUDA é ideal para o desenvolvimento de software para problemas de paralelização de dados [14]. Em relação ao modo de funcionamento desta plataforma, o processo é iniciado sendo realizada a cópia dos dados a serem processados da memória principal para a componente de memória da unidade de

processamento gráfico, de seguida o GPU efetiva a operação ou tarefa a ser realizada pela totalidade dos dados, também designada de *kernel*, concluindo com a transferência dos resultados da componente de memória do GPU para a memória principal.

2.4.3 Comparação de Ferramentas e Plataformas de Desenvolvimento

Comparando as ferramentas e plataformas de desenvolvimento utilizadas na implementação de sistemas de computação paralela e computação heterogénea com o GPU podemos verificar que o OpenMP é a ferramenta que implica menores alterações de código para a sua integração visto que esta integração é realizada através da inserção de diretivas de compilador *pragma*. De seguida é apresentado um excerto de código sem recorrer a integração da ferramenta OpenMP:

```
int main(){
    for(int i = 0; i < N_ITERATIONS; i++)
        func(...);
    return 0;
}
```

Figura 2.11: Excerto de código sem paralelização

Podemos verificar na figura seguinte que o excerto de código anterior pode ser paralelizável facilmente, sem alterar drasticamente o código original, através da implementação do OpenMP. A inserção da diretiva de compilador permite que o bloco de instruções incluído no ciclo for seja dividido e executado pelos vários *threads* da máquina:

```
int main(){
    #pragma omp parallel for
    for(int i = 0; i < N_ITERATIONS; i++)
        func(...);
    return 0;
}
```

Figura 2.12: Excerto de código paralelizado com OpenMP

O OpenMPI é uma ferramenta versátil para o desenvolvimento de soluções paralelas de memória distribuída, especialmente quando esta solução recorre às capacidades de processamento de um grupo de diferentes máquinas, pois permite um mecanismo simples de transmissão de dados por sistemas com caraterísticas diferentes.

O OpenCL e o CUDA são plataformas bastante semelhantes em termos de implementação de uma solução paralela, isto porque ambas necessitam da implementação de um *kernel*, utilizado para o processamento de um conjunto de dados. Ambas plataformas apresentam funcionalidades idênticas o que torna o processo de portabilidade de programas entre estas plataformas simples de ser realizado. Embora a plataforma CUDA seja destinada para implementação de soluções que utilizem unidades de processamento gráfico da Nvidia, esta plataforma apresenta um melhor desempenho de execução em todos aspetos em relação à plataforma OpenCL [15]. O período

de transferência de dados entre a memória principal e a componente de memória da unidade de processamento gráfico, o tempo de aplicação de um *kernel* sobre um conjunto de dados e à complexidade da solução são parte dos aspetos que apresentam melhores resultados com a plataforma CUDA do que a plataforma OpenCL.

2.5 Conclusão

A partir do conteúdo apresentado neste capítulo podemos aferir que as diferenças das arquiteturas entre um CPU e uma unidade de processamento gráfico, têm implicações na sua performance. A arquitetura de um processador, devido aos níveis de memória que contém, apresenta um valor baixo de latência de acesso a memória principal por possuir vários níveis de memória cache que armazenam espaços de memória frequentemente acedidos. Os valores de tempo de acesso a memória só é afetado caso existam ocorrências de *false sharing*.

O GPU, por conter um número reduzido de unidades de controlo e de memória, permite a incorporação de um maior número de unidades de lógica e aritmética, o que possibilita trocar toda a flexibilidade funcional de processamento de um CPU por uma maior capacidade computacional, caso haja a utilização das unidades de processamento gráfico em substituição dos processadores.

Tendo em conta a abordagem de computação heterogénea utilizada na solução deste trabalho, as ferramentas que apresentam mais vantagens de utilização são a ferramenta OpenMP e a plataforma de desenvolvimento CUDA. A ferramenta OpenMP apresenta uma boa performance de execução para sistemas paralelos e é ideal para a paralelização funcional das instruções executadas no processador sem introduzir um nível de complexidade adicional ao código do programa. A plataforma de desenvolvimento CUDA destaca-se por apresentar um melhor desempenho em termos gerais em relação à plataforma OpenCL [15]. Também, a solução resultado deste trabalho será destinada a ser executada em máquinas com unidades de processamento gráfico da Nvidia, o que torna a utilização do CUDA ideal nestas circunstâncias.

Capítulo 3

Processamento de Vídeo

De seguida é apresentado um estudo e compreensão das fases de processamento de vídeo que constituem o problema de processamento de vídeo, especificamente, as fases de reamostragem e redimensionamento. Esta análise foi dividida nos seguintes componentes:

- Estudo da representação digital de vídeos e imagens;
- Descrição dos diferentes formatos e representação interna de pixéis de diferentes espaços de cor;
- Operação de conversão de imagens entre diferentes modelos de cor;
- Verificação de abordagens utilizadas no processo de redimensionamento de imagem;
- Análise dos resultados obtidos pela aplicação do processo de redimensionamento;
- Bibliotecas e ferramentas disponíveis para a realização dos processos de vídeo.

3.1 Estrutura de um Vídeo

Um vídeo é uma sequência de imagens, usualmente designadas de frames. Cada frame é uma imagem estática que visualizada em sequência, em conjunto com outras frames, recriam a uma imagem animada do vídeo. O número de frames que constituem um vídeo depende do número de imagens capturadas na gravação do vídeo, este valor refere-se à métrica frame rate. Esta medida é a frequência a que cada frame é capturada por segundo. A frame *rate* também pode ser designada de frame *frequency* e ser expressa em hertz ao invés de frames por segundo.

O processamento de um vídeo como a aplicação de processos de reamostragem e redimensionamento, que serão abordados nas secções seguintes, pode ser tomada como a aplicação dos mesmos processos à sequência de imagens que constituem o vídeo, por esta razão a resolução do problema de processamento de um vídeo será considerado como a resolução do mesmo problema numa só imagem.

3.1.1 Representação de uma Imagem

As imagens podem ser distinguidas segundo o mecanismo de armazenamento gráfico utilizado na sua representação digital por imagens vetorizadas e imagens rasterizadas.

As imagens vetorizadas são formadas a partir de formulações matemáticas que definem primitivas geométricas, as suas posições relativas em relação a outras primitivas e as suas respetivas cores. As primitivas geométricas que constituem as imagens vetorizadas são representadas por pontos, linhas, curvas e formas geométricas básicas. A disposição e a escala destas primitivas são calculadas em função da resolução em que a imagem será apresentada, por esta razão é possível apresentar este tipo de imagens com diferentes dimensões sem que haja alteração da qualidade das caraterísticas da imagem.

As imagens rasterizadas são constituídas por pixeis, o elemento mais básico de representação gráfica. Mais especificamente, as imagens rasterizadas são representadas digitalmente a partir de uma matriz de duas dimensões que armazena os valores de intensidade de cor para cada um dos pontos da imagem.

Como as imagens vetorizadas são constituídas a partir de primitivas geométricas, este tipo de imagens tornam-se inviáveis para a representação de imagens capturadas a partir de um cenário real por não apresentarem detalhe suficiente das formas e cores das características do cenário. Tendo em conta o contexto do problema desta dissertação serão apenas tidas em conta imagens rasterizadas.

3.2 Modelos de cor

Os pixeis de uma imagem representam o valor de intensidade de cor de um ponto de uma imagem segundo um modelo de cor específico. Um modelo de cor descreve de forma normalizada como o valor de intensidade de cor é representado utilizando um sistema de coordenadas de três dimensões segundo o qual cada valor de intensidade de cor é representado por um único ponto deste sistema.

Existem dois diferentes tipos de modelos de cor: os modelos de cor de imagem que se focam na representação de imagens e da sua qualidade como os modelos RGB, CMY, HSI, entre outros; e os modelos de cor de vídeo que são direcionados para a representação de imagens de um vídeo focando-se na redução do tamanho necessário para representação dos pixeis que as constituem ao invés da qualidade das próprias imagens, YIQ, YUV, Y'CbCr são exemplos deste tipo de modelos.

No contexto do trabalho desta dissertação são apenas considerados os modelos RGB, quanto aos modelos de cor de imagem, e Y'CbCr, quanto aos modelos de cor de vídeo, pois estes modelos são, respetivamente, considerados standard da tecnologia de captura de imagem de câmaras digitais e de dispositivos de playback digital de vídeo, e daí serem de maior relevância para o problema em análise.

3.2.1 Modelo RGB

O modelo de cor RGB é um modelo aditivo de cor em que o valor de intensidade de cor representado por um pixel é constituído por três diferentes componentes que correspondem às cores primárias vermelho, verde e azul (*Red, Green and Blue*).

Segundo este modelo, a cor de um ponto de uma imagem é igual ao resultado da junção dos valores de cor de cada uma das três componentes de cores primárias. Pode ser visualizado na figura seguinte a decomposição de uma imagem nas diferentes componentes de cor primária, sendo a primeira imagem o resultado da junção das três seguintes componentes:

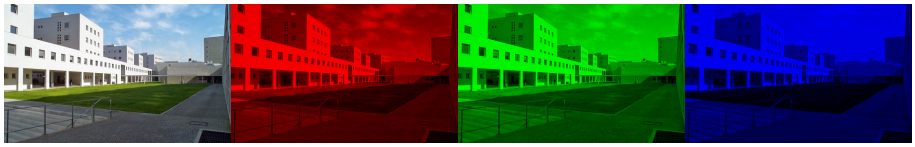

Figura 3.1: Decomposição de uma imagem nas componentes do modelo de cor RGB

A estrutura interna de um pixel, de acordo com este modelo, é dividida pelas diferentes componentes de cores primárias, sendo que em algumas representações deste modelo existe a adição de uma nova componente correspondente ao valor de transparência de cor de um pixel.

3.2.1.1 Profundidade de cor

A profundidade de cor é o número de bits utilizados na representação do valor de cada componente de cor de um pixel. Quanto maior for o valor de profundidade de cor, maior será o intervalo de valores que cada componente pode representar e, por consequência, maior será a variedade de tonalidades de cor que um pixel pode tomar [16].

A figura seguinte demonstra as diferenças de variedade de cor presentes numa imagem com diferentes valores de profundidade de cor, respetivamente: 8 bits, 4 bits, 2 bits e 1 bit.

Figura 3.2: Influência do valor de profundidade de cor na variedade de cores de uma imagem

3.2.2 Modelo Y'CbCr

O modelo de cor Y'CbCr é um modelo que tira proveito da fraca perceção da visão humana às variações de cor em relação às variações de luminosidade de uma imagem, por essa razão o valor de cor de um pixel é constituído por três diferentes componentes que correspondem à componente luma, o brilho de uma imagem, e duas outras componentes de crominância, segundo uma projeção de cor azul e vermelha [17].

Segundo este modelo, a cor de um ponto de uma imagem é igual ao resultado da junção dos valores de intensidade de cada uma das três componentes referidas. Pode ser visualizado na figura seguinte a decomposição de uma imagem nas diferentes componentes de luma e crominância, sendo a primeira imagem o resultado da junção da seguinte componente de luma e das restantes imagens correspondentes às componentes de crominância:

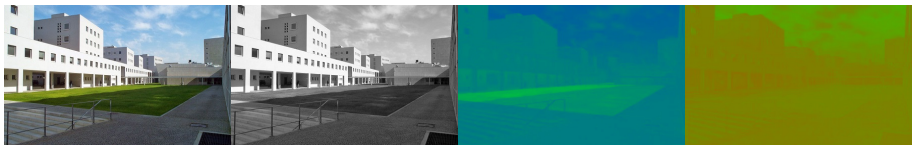

Figura 3.3: Decomposição de uma imagem nas componentes do modelo de cor Y'CbCr

A estrutura interna de um pixel, de acordo com este modelo, é dividida pelas diferentes componentes de luma e crominâncias, sendo que em algumas representações deste modelo existe a partilha dos valores de crominância por diferentes pixeis de um determinado bloco de pixeis.

3.2.2.1 Subamostragem de Crominâncias

A subamostragem de crominâncias é a prática que consiste em reduzir o número de amostras das componentes de crominâncias de uma imagem segundo o modelo Y'CbCr, esta prática permite reduzir o tamanho utilizado na representação de uma imagem.

Segundo esta prática, um bloco de pixeis partilha os mesmos valores das componentes de crominância ao invés de cada pixel ter o seu próprio valor. A partilha dos valores de crominância por blocos de pixeis é definida através de diferentes tipos de amostragem que podem ser observados na seguinte figura:

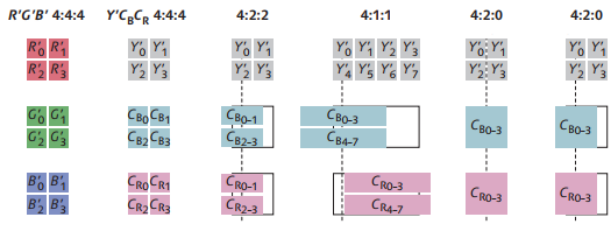

Figura 3.4: Diferentes tipos de subamostragem de crominâncias [3]

Para cada tipo de subamostragem de crominância, cada pixel mantém um valor próprio da componente de luma, havendo apenas partilha das componentes de crominância que difere com os diferentes tipos de subamostragem.

Para uma subamostragem do tipo 4:4:4 as componentes de crominância não são partilhadas por diferentes pixeis, enquanto numa subamostragem do tipo 4:2:2 cada dois pixeis consecutivos partilham os mesmos valores de crominância. Na subamostragem do tipo 4:1:1 existe a partilha das componentes de crominância por cada quatro pixeis consecutivos. Por último, numa subamostragem do tipo 4:2:0 existe a partilha das componentes de crominância por cada bloco de quatro pixeis [18].

3.3 Reamostragem de imagem

O processo de reamostragem designa o processo de conversão do espaço de cores de uma imagem, isto é, é o processo de alteração do modelo de cor utilizado para a representação dos pixeis de uma imagem. Para perceber a operação de conversão de uma imagem entre os modelos de cor considerados, o modelo RGB e Y'CbCr, é primeiro necessário entender as características dos formatos utilizados por cada um dos modelos na representação das respetivas componentes de cor na estrutura interna de um pixel.

3.3.1 Formatos do modelo RGB

De seguida são apresentados os formatos mais frequentemente utilizados na representação das componentes de cor de um pixel segundo o modelo RGB, a respetiva esquematização e uma descrição do formato:

- RGB888

O formato RGB888 considera uma profundidade de cor de 8 bits por cada componente de cor na representação dos pixeis de uma imagem. Cada pixel é representado pela sequência das componentes de cor primária vermelho, verde e azul, utilizando para esta representação 24 bits.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B

Figura 3.5: Estrutura interna de um pixel segundo o formato RGB888

- RGBA8888 e ARGB8888

Estes formatos são uma representação do modelo RGB considerando uma profundidade de cor de 8 bits por componente de cor com a adição de uma nova componente referente a transparência de cor do pixel representado. Esta componente de transparência é, também, representada utilizando 8 bits.

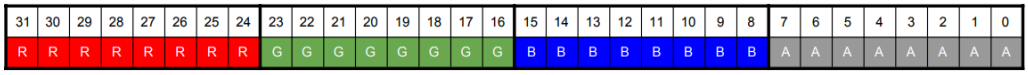

Figura 3.6: Estrutura interna de um pixel segundo o formato RGBA8888

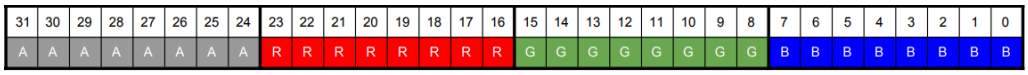

Figura 3.7: Estrutura interna de um pixel segundo o formato ARGB8888

Os dois modelos apresentados diferem segundo a posição da representação das componentes de transparência de cor do pixel. No formato RGBA8888, a componente de transparência é representada após a representação da sequência das componentes de cores primárias do modelo, enquanto, segundo o formato ARGB8888, a componente de transparência é representada antes das restantes componentes.

3.3.2 Formatos do modelo Y'CbCr

De seguida são apresentados os formatos de representação de um pixel segundo o modelo de cor Y'CbCr, a respetiva esquematização e uma descrição do formato:

- YCbCr444

O formato YCbCr444 tem em conta uma subamostragem de crominâncias do tipo 4:4:4 para o modelo de cor Y'CbCr com uma profundidade de cor de 8 bits. Segundo este formato, cada pixel é representado pela componente de crominância Cb, seguida da componente de luma e a componente de crominância Cr, utilizando 24 bits para esta representação.

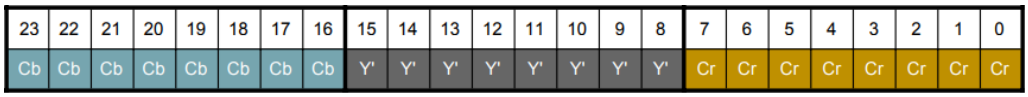

Figura 3.8: Estrutura interna de um pixel segundo o formato YCbCr444

- YCbCr422

A representação dos pixéis de uma imagem segundo o formato YCbCr422, considera uma subamostragem de crominâncias do tipo 4:2:2, isto é, cada par de pixéis consecutivos partilha os mesmos valores de crominâncias e por essa razão, a representação deste tipo de formatos tem em conta a representação do par de pixéis ao invés da representação singular de cada pixel.

A representação corresponde, então, à representação do primeiro pixel do par à semelhança do formato YCbCr444, com a particularidade de ser apresentado uma componente adicional correspondente ao valor de luma do segundo pixel do par a ser representado. Segundo este tipo de formato é possível representar dois pixéis diferentes utilizando 32 bits.

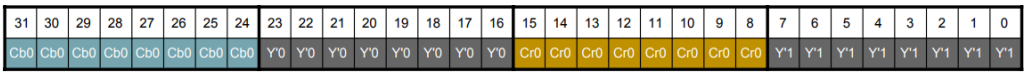

Figura 3.9: Estrutura interna de um bloco de pixéis segundo o formato YCbCr422

- YCbCr411

O formato YCbCr411 tem em conta uma subamostragem de crominâncias do tipo 4:1:1 para o modelo de cor Y'CbCr, em que cada bloco de 2x2 pixéis de dimensão partilha os mesmos valores de crominâncias. Segundo este formato, cada bloco de pixéis é representado pela componente de crominância Cb, seguida das componentes de luma de dois dos pixéis do bloco, a componente de crominância Cr e, por último, as componentes de luma dos dois restantes pixéis do bloco, utilizando 48 bits para a representação dos 4 pixéis do bloco.

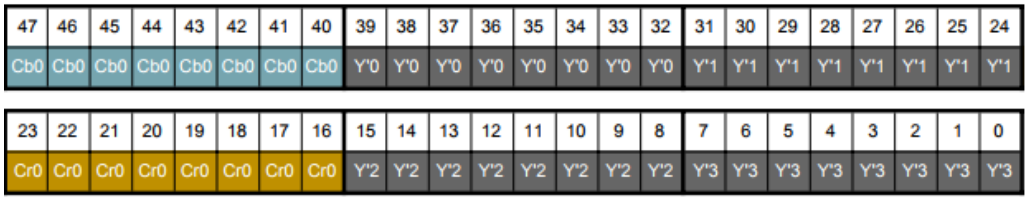

Figura 3.10: Estrutura interna de um bloco de pixéis segundo o formato YCbCr411

- YCbCr420p

O formato YCbCr420p difere dos restantes formatos do modelo de cor Y'CbCr por apresentar uma característica na representação das componentes de um pixel, nomeada de formato planar. O formato planar designa uma representação dos pixeis de uma imagem agrupando os valores das componentes de cada pixel, ao invés do que acontece com os anteriores formatos, segundo os quais os pixeis de uma imagem são representados entrelaçando as respetivas componentes. Segundo o formato YCbCr420p, uma imagem com N pixeis é representada internamente por todas as componentes de luma de todos os pixeis, seguido das componentes de crominância para cada bloco de 2x2 pixeis.

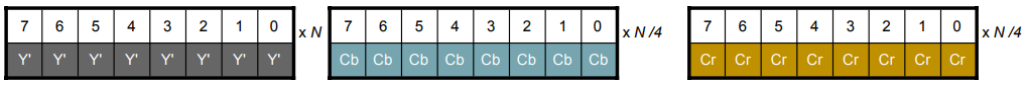

Figura 3.11: Estrutura interna de uma imagem segundo o formato YCbCr420p

Tendo em conta os diferentes formatos de representação de um pixel, o processo de conversão do espaço de cores de uma imagem entre os modelos de cor mencionados é conseguido a partir do cálculo de cada componente de cor e organizando a sua disposição na representação interna da imagem [19].

Sendo então o processo de reamostragem de uma imagem, fundamentalmente, um processo de transformação de valores, este pode ser realizado através das seguintes equações de conversão entre as diferentes componentes de cor de cada modelo [20]:

$$R = clamp(Y + 1.402 * (Cr - 128)) \tag{3.1}$$
$$G = clamp(Y - 0.344 * (Cb - 128) - 0.714 * (Cr - 128)) \tag{3.2}$$
$$B = clamp(Y + 1.722) * (Cb - 128) \tag{3.3}$$

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \tag{3.4}$$
$$Cb = -0.169 * R - 0.331 * G + 0.499 * B + 128 \tag{3.5}$$
$$Cr = 0.499 * R - 0.418 * G - 0.0813 * B + 128 \tag{3.6}$$

Considerando um valor de profundidade cor de 8 bits, o valor decimal que uma componente pode tomar pode tomar os valores entre 0 e 255 no caso do modelo de cor RGB, ou a componente de luma pode tomar os valores entre 0 e 255, e as componentes de crominância podem tomar valores entre -127 e 128, no caso do modelo de cor Y'CbCr. Para restringir os valores que uma componente de cor pode tomar foi utilizada a operação *clamp* cuja função é atribuir o valor do limite mais próximo do intervalo de cores caso o valor do seu parâmetro os ultrapasse.

3.4 Redimensionamento de Imagem

O redimensionamento é um processo que aplicado a uma imagem produz uma nova versão com diferentes dimensões, é o aumento ou a diminuição do número de pixeis que constituem uma imagem. A dimensão com que as imagens são criadas têm em conta a aparência das suas características ao serem representadas numa determinada resolução, a alteração da dimensão de uma imagem provoca um desalinhamento entre a disposição original dos seus pixeis e as novas posições dos pixeis da imagem redimensionada.

De seguida é apresentado uma esquematização do desalinhamento consequente do aumento de dimensão de uma imagem e a disposição original dos seus pixeis:

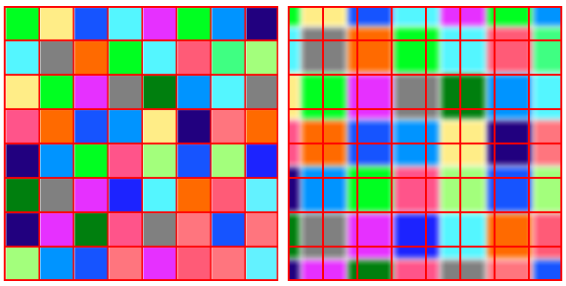

Figura 3.12: Desalinhamento entre os pixeis e a sua disposição original

O aumento da dimensão da imagem, ou *upsampling* implica a introdução de novos pixeis na imagem escalada que não existiam na imagem original enquanto que com *downsampling*, ou a redução da dimensão da imagem, existe a remoção de parte dos pixeis originais. Para realizar o redimensionamento de uma imagem é necessário calcular a cor dos novos pixeis inseridos, no caso da operação de *upsampling*, e a cor dos pixeis que restaram após a remoção de parte dos pixeis originais, no caso da operação de *downsampling*, tendo em conta a conservação das características das estruturas apresentadas pela imagem original.

O cálculo de cor referido constitui o problema de redimensionamento que pode ser realizado através de diferentes abordagens que balanceiam a qualidade dos resultados e o tempo de processamento. De seguida são apresentadas as diferentes abordagens para este problema.

3.4.1 Nearest Neighbor

Segundo este algoritmo, o valor de intensidade de cor de um pixel de uma imagem escalada será igual ao valor de intensidade de cor do pixel mais próximo da imagem original.

Assim, como exemplo, ao aumentar a dimensão de uma imagem por um fator de 2, um pixel da imagem original é representado por uma área de 2 por 2 pixeis na imagem resultante com o mesmo valor de intensidade de cor.

No caso do processo de *upsampling* a imagem resultante apresenta os mesmos valores de intensidade de cor que a imagem original, como pode ser visualizado no exemplo seguinte em que uma imagem é escalada por um fator de 3 na horizontal e um fator de 2 na vertical.

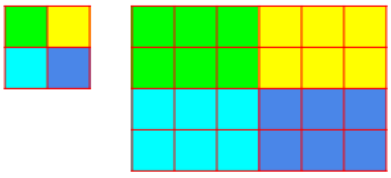

Figura 3.13: *Upsampling* de 3x2 utilizando o algoritmo Nearest Neighbor

O processo de *downsampling*, segundo este algoritmo, apresenta um problema de perda de informação, visto que esta transformação implica a remoção de parte dos pixeis da imagem original de modo a obter uma imagem com as dimensões desejadas.

Neste tipo de casos é necessário indicar uma condição de preferência de escolha de pixeis quando o pixel resultante está à mesma distância de dois ou mais pixeis originais.

De seguida é apresentado um exemplo *downsampling* de uma imagem por um fator de 2 na horizontal e 1,25 na vertical em que ocorre o problema mencionado:

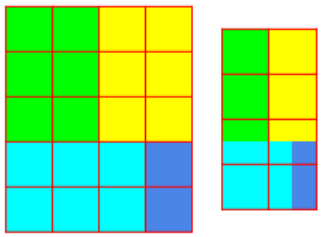

Figura 3.14: *Downsampling* incompleto de 2x1,25 utilizando o algoritmo Nearest Neighbor

Na figura a imagem da direita, os pixeis da terceira linha e o segundo pixel da quarta linha encontram-se em conflito por estarem à mesma distância de diferentes pixeis originais, a sua cor não pode ser calculada a não ser que seja fornecido uma condição de decisão. Esta condição de decisão indica qual cor persistirá perante um conflito.

Usualmente a condição de decisão tem como referência a proximidade ao canto superior esquerdo da imagem original, isto é, na ocorrência de um conflito, a cor do pixel original mais próximo do canto superior esquerdo será atribuída ao pixel em conflito. Na seguinte imagem é apresentado o resultado do processo de *downsampling* nas mesmas condições especificando o canto superior esquerdo da imagem original como referência na resolução de conflitos:

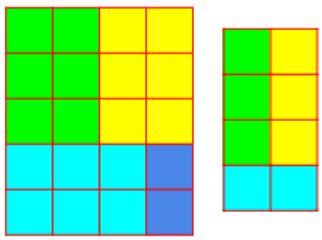

Figura 3.15: *Downsampling* de 2x1,25 segundo o canto superior esquerdo utilizando o algoritmo Nearest Neighbor

3.4.2 Interpolação Bilinear

Segundo este algoritmo, o valor de intensidade de cor de um pixel da imagem escalada será igual à média ponderada dos quatro pixeis originais mais próximos. Este algoritmo é um processo mais sofisticado em comparação ao algoritmo Nearest Neighbor pois são utilizadas operações de interpolação linear para calcular a intensidade de cor de um pixel da imagem resultante[21]. Quanto maior for a proximidade de um pixel original ao pixel da imagem escalada, maior será a sua influência no valor de intensidade de cor a calcular.

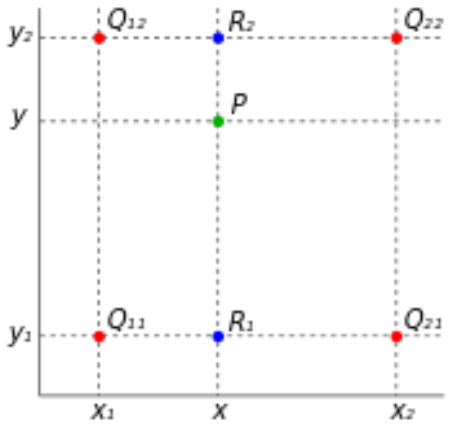

Figura 3.16: Interpolação bilinear no cálculo de intensidade de cores

Na figura anterior é possível visualizar um exemplo da aplicação deste algoritmo. O valor de intensidade de cor do pixel no ponto P é igual ao valor do resultado da interpolação linear entre os pontos R_1 e R_2, que são resultado, respetivamente, de uma operação de interpolação linear entre os pontos Q_11 e Q_21, e os pontos Q_12 e Q_22.

Este algoritmo introduz novos valores de intensidade de cor que poderiam não existir na imagem original por consequência das interpolações realizadas, por esta razão as imagens escaladas recorrendo a este algoritmo apresentam características desfocadas especialmente em locais da imagem em que existem grandes discrepâncias de intensidade de cor.

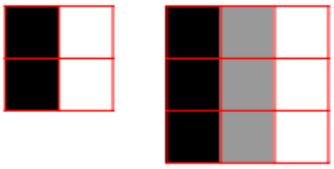

Figura 3.17: *Upsampling* de 1,5x1,5 utilizando o algoritmo de interpolação bilinear

No exemplo anterior é possível verificar a introdução de novos valores de intensidade de cor na imagem resultante que não existiam na imagem original. Estes novos pixeis provocam uma aparência desfocada nas delimitações das características da imagem original.

3.4.3 Interpolação Bicúbica

À semelhança da interpolação bilinear, o algoritmo de interpolação bicúbica utiliza os pixeis originais mais próximos do pixel da imagem escalada para calcular o seu valor de intensidade de cor. Contudo, a interpolação bicúbica utiliza uma função polinomial de terceiro grau para o fazer recorrendo aos valores de intensidade de cor dos dezasseis pixeis mais próximos. Por esta razão,

o valor calculado será mais aproximado às variações de gradiente de cor da imagem original do que o resultado de uma interpolação bilinear [22].

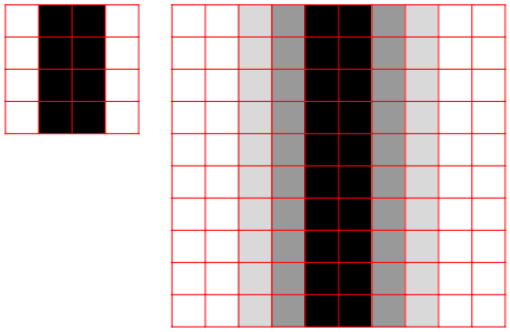

Figura 3.18: *Upsampling* de 2,5x2,5 utilizando o algoritmo de interpolação bicúbica

No exemplo anterior podemos ver o resultado do processo de *upsampling* por um fator de 2,5 segundo o algoritmo de interpolação bicúbica. Os resultados deste algoritmo apresentam uma maior homogeneidade nas diferenças de gradiente da imagem em comparação com o algoritmo de interpolação bilinear o que aparenta resultados de melhor qualidade embora com características mais desfocadas.

Capítulo 4

Conclusão

Neste capítulo serão referidas as conclusões deste projeto no âmbito da unidade curricular de PDIS, ou Preparação de Dissertação, mencionando o fecho do trabalho, a precisão dos resultados a obter, as tarefas a serem concluídas e o seu plano de execução.

4.1 Conclusão

Com o estudo realizado no decorrer deste trabalho, foi possível observar as diferentes arquiteturas e mecanismos de implementação de sistemas paralelos, quer estes abordem mecanismos de computação paralela ou de computação heterogénea. As diferenças entre as unidades de processamento e a sua constituição, como os componentes de memória, permitiram uma melhor compreensão sobre os aspetos a ter em conta no planeamento da solução do problema. A investigação e a comparação entre as ferramentas e plataformas de desenvolvimento de implementação de sistemas paralelos possibilitou uma perceção mais precisa das capacidades de cada uma delas e que ambientes estas melhor desempenham.

O planeamento da implementação da solução a ser realizada foi apurada devido à análise dos métodos e algoritmos utilizados durante o processo de reamostragem e redimensionamento de vídeo. Este estudo auxiliou a aplicação do método de extração de paralelização de problemas de Foster, isto é, a melhor compreensão das operações a serem realizadas ajudou na preparação da divisão de dados e tarefas que serão divididas pelas diferentes unidades de processamento.

Em suma, todo o trabalho realizado permitirá que o trabalho de desenvolvimento a ser realizado nesta próxima fase da dissertação seja acelerado, pois os problemas e peculiaridades dos sistemas paralelos já foram devidamente ponderados.

4.2 Solução e Resultados Esperados

Como já enunciado no capítulo introdutório, o objetivo deste trabalho é a implementação de uma ferramenta para a aplicação dos processos de reamostragem e redimensionamento de imagem recorrendo a uma abordagem de computação heterogénea, aliando as capacidades de processamento de um processador e de uma unidade de processamento gráfico Nvidia. A ferramenta desenvolvida receberá um vídeo como parâmetro de entrada e as configurações do resultado a ser obtido, como o modelo de cor utilizado e a resolução de vídeo. O processamento do vídeo será realizado em três fases: fase de divisão de um vídeo por frames utilizando a ferramenta OpenMP, recorrendo às capacidades de processamento do CPU para o efeito, e as restantes duas fases serão a operação de redimensionamento e de reamostragem de imagem aplicadas a cada frame de um vídeo, recorrendo a aceleração gráfica com a plataforma de desenvolvimento CUDA da Nvidia.

O resultado esperado é uma ferramenta capaz de realizar os processos de reamostragem e redimensionamento em tempo real, com um melhoramento de tempo de execução face às ferramentas sequenciais já existentes para o efeito como é o caso da ferramenta FFmpeg.

4.3 Tarefas a Serem Realizadas e Plano de Trabalho

O trabalho a ser efetuado durante o próximo semestre será realizado segundo o seguinte agendamento. O primeiro mês, mês de fevereiro, será destinado à familiarização das ferramentas e plataformas de desenvolvimento de sistemas paralelos, especificamente, a ferramenta OpenMP e a plataforma CUDA. Durante este período de tempo será também preparado o ambiente físico onde será desenvolvida a dissertação, procedendo-se a instalação de ferramentas necessárias e elaboração de testes com as mesmas. A solução será desenhada em termos de arquitetura de sistema e, o problema será analisado e planeado segundo a metodologia de Foster.

Os meses seguintes, de março e abril, serão destinados à implementação do sistema que dá solução ao problema desta dissertação, dividindo a mesma em duas fases distintas, a fase de implementação sequencial da solução e a fase de paralelização da mesma. Ao longo deste período, a dissertação será escrita gradualmente sofrendo um processo de refinamento até à data de submissão.

Em maio e junho, os meses finais de trabalho, serão dedicados à resolução de problemas que podem ter ocorrido durante a fase de implementação, como o comportamento inesperado de alguma funcionalidade, e a análise e comparação de resultados da ferramenta desenvolvida com as ferramentas de realização dos processos de reamostragem e redimensionamento atuais.

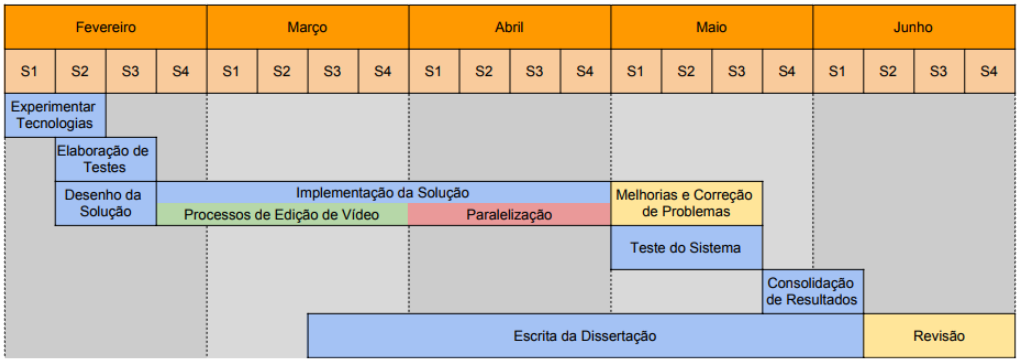

Figura 4.1: Plano de trabalho da dissertação

Conclusão

Referências

[1] Warren Toomey. Introduction to systems architecture. Disponível em http://minnie.tuhs.org/CompArch/Lectures/week01.html, Maio 2011.

[2] Wgsimon. Transistor count and moore's law. Disponível em https://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2008.svg, Novembro 2008.

[3] Charles Poynton. Chroma subsampling notation. *Retrieved June*, 19:2004, 2002.

[4] William Knight. Two heads are better than one [dual-core processors]. *IEE Review*, 51(9):32–35, 2005.

[5] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.

[6] Nikola Zlatanov. Computer memory, applications and management, 02 2016.

[7] Cristobal A Navarro, Nancy Hitschfeld-Kahler, e Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, 15(2):285–329, 2014.

[8] Kai Hwang e Zhiwei Xu. *Scalable Parallel Computing: Technology,Architecture,Programming*. McGraw-Hill, Inc., New York, NY, USA, 1998.

[9] Josep Torrellas, HS Lam, e John L. Hennessy. False sharing and spatial locality in multiprocessor caches. *IEEE Transactions on Computers*, 43(6):651–663, 1994.

[10] Ian Foster. *Designing and building parallel programs*, volume 78. Addison Wesley Publishing Company Boston, 1995.

[11] Blaise Barney. Openmp. *Website: https://computing. llnl. gov/tutorials/openMP*, 2008.

[12] MPI Open. Open source high performance computing, 2012.

[13] Khronos Group. The opencl specification, Maio 2017.

[14] Fedy Abi-Chahla. Nvidia's cuda: The end of the cpu?', 2008.

[15] Kamran Karimi, Neil G Dickson, e Firas Hamze. A performance comparison of cuda and opencl. *arXiv preprint arXiv:1005.2581*, 2010.

[16] Ben Waggoner. *Compression for great digital video: power tips, techniques, and common sense*. Taylor & Francis, 2002.

[17] George H Joblove e Donald Greenberg. Color spaces for computer graphics. Em *ACM siggraph computer graphics*, volume 12, páginas 20–25. ACM, 1978.

[18] Douglas A Kerr. Chrominance subsampling in digital images. *The Pumpkin,(1), November*, 2005.

[19] David A Rumball. Method and apparatus for converting digital yuv video signals to rgb video signals, Junho 23 1992. US Patent 5,124,688.

[20] Yang Yang, Peng Yuhua, e Liu Zhaoguang. A fast algorithm for ycbcr to rgb conversion. *IEEE Transactions on Consumer Electronics*, 53(4), 2007.

[21] Sen Wang e KJ Yang. An image scaling algorithm based on bilinear interpolation with vc++. *Techniques of Automation and Applications*, 27(7):44–45, 2008.

[22] Marco Aurelio Nuno-Maganda e Miguel O Arias-Estrada. Real-time fpga-based architecture for bicubic interpolation: an application for digital image scaling. Em *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*, páginas 8–pp. IEEE, 2005.