Universidade de Aveiro
Mestrado em Engenharia Computacional
Computação Paralela
40779

FAST CORNER DETECTION WITH CUDA AND OPENMP

Vasco Costa - 97746

22 de Maio 2023

Introdução

Acontecimentos em paralelo são algo do quotidiano e alguns dos mesmos princípios podem ser aplicados à programação e resolução de problemas.

A utilização de sistemas de computação paralela em sistemas distribuídos apresenta vantagens e desvantagens devido às considerações que devem ser tomadas, como o tempo de resposta, sendo possível distinguir modos síncronos de assíncronos [5]. Desta forma é sabido que o desempenho das unidades é sempre inferior à possível nominal individual. A componente não produtiva de processos paralelos depende do número de unidades em trabalho apresentando uma maior importância quanto maior for a escala. Em grandes aplicações, como em super computadores, este fenómeno já tem sido evidente e é a explicação para alguns dos resultados obtidos [6].

Na área do processamento de imagens, a utilização de métodos de segmentação é bastante comum para a atribuição a *threads* independentes. Existem métodos que utilizam outras abordagens para tarefas mais complexas, a fim de executá-las de forma independente [2]. Da mesma forma, é uma abordagem vantajosa para a encriptação de imagens, que requer permutações e apresenta um melhor desempenho ao ser feita dessa maneira [7].

Considerando que existem diversas vantagens ao aplicar programação paralela em diversos casos de aplicação pretende-se com este trabalho partir de um algoritmo para o processamento de imagens para a deteção de cantos e paralelizá-lo com diferentes ferramentas. Para a deteção de um canto numa imagem em escala de cinzento aplica-se o algoritmo de *FAST Detector* [4] que identifica um canto consoante a luminosidade do *pixel* e dos seus vizinhos.

OpenMP

Uma das plataformas de trabalho pedida é através da utilização das diretivas de *OpenMP*. Este modelo de programação em memória partilhada nasceu da cooperação de diversos fabricantes de software e de hardware.

Nesta implementação utilizou-se o programa fornecido, `fastDetectorOpenMP.cpp`, e modificou-se a função `fastDetectorOpenMP`. Esta é baseada na função `fastDetectorHost`que já se encontrava disponibilizada e efetuava a deteção de cantos de forma sequencial.

Processo

Analisando a sequência de passos que o algoritmo toma verifica-se que inicialmente ocorre a alocação de memória para a imagem de saída e de seguida existe a deteção de cantos, `fastDetectCorners`, esta pode ser paralelizada, `fastDetectCorners_OpenMP`, de forma a que cada *thread* analise uma porção da imagem em vez de esta ser toda analisada por uma só. Como este processo é independente a sua realização é mais fácil e apenas é necessário ter aten-

ção para que as variáveis de iteração sejam privadas quando definidas anteriormente ao ciclo. Podendo-se assim incluir a instrução `#pragma omp parallel for num_threads-(NUM_THREADS)` para quando se efetuam ciclos. Desta forma o ciclo é subdividido pelas `NUM_THREADS` em que cada *thread* faz um ciclo de menor tamanho, $\frac{1}{\texttt{NUM\_THREADS}}$. A inclusão de `reduction` para a variável de `count` é apenas relevante para processos de desenvolvimento para obter a soma total final, uma vez que não apresenta efeito na figura final esta parte foi comentada.

Um dos pontos do algoritmo é que este leva a que ocorra a classificação de mais cantos do que os que realmente existem e por isso existe uma forma de atribuir um *score* a cada um dos classificados e apenas se contabiliza o que tiver um maior valor, para isto ao executar o programa tem que se incluir a opção `-m`. Este processo é desempenhado pela função `non-MaximumSupression` que pode ser paralelizada, `nonMaximumSupression_OpenMP`, dado que o processo de determinação do *score* de cada canto pode ser feito independentemente. Da mesma forma é possível depois verificar quais os cantos que apresentam maior *score* face aos seus vizinhos.

Também o processo final para o preenchimento da figura com os cantos com a original mas a mais escuro para se evidenciarem os cantos pode ser paralelizado com o mesmo intuito de a cada trabalhador atribuir uma porção da figura.

Contudo, a abordagem aqui tomada de paralelizar quando possível não é a mais adequada em todas as situações, para imagens mais pequenas, de menor resolução, este método não é o mais benéfico. Isto devido ao pré-processamento necessário para realizar as tarefas em paralelo, sendo que ainda seria pior em casos de limitação de largura de banda de memória.

Resultados

A obtenção destes resultados foi feita numa máquina virtual com a distribuição de Ubuntu 22.04.2 LTS com 6,6Gib de memória num processador AMD Ryzen 7 3700x 8-core com 6 processadores alocados. A compilação foi feita através da *Makefile* disponibilizada com `g++ -fopenmp -O3`.

Para verificar os resultados obtidos e medir se estes eram melhores fez-se uso da equação 1 para calcular o benefício do método. O cálculo do *speedup* será assim a razão entre o tempo de execução entre o código sequencial e o que se encontra paralelizado. Esta acaba por ser a maneira mais simples e básica de verificar o aumento de desempenho.

$$Speedup = \frac{\text{Tempo de execução sequencial}(s)}{\text{Tempo de execução paralelizado}(s)} \tag{1}$$

Assim foram calculados os tempos e os *speedups* obtidos para cada imagem com o método paralelizado consoante o número de *threads* utilizados, sendo todos os resultados apresentados na Tabela A e os valores médios aí presentes na Tabela 1. Estes foram obtidos mais facilmente através de *scripts* da *bash*, `run_open_mp.sh`, que também fez a comparação de cada um dos

resultados obtidos para verificar que não tinha existido qualquer erro de programação. Para a seleção do número de *threads* a utilizar alterou-se o valor da variável indicativa e voltou-se a compilar o código para testar.

Tabela 1 – Valores dos *speedup* médios obtidos com o código desenvolvido em *OpenMP* repetindo 10 vezes para cada imagem consoante o número de *threads* utilizado.

Número de *Threads*	Ficheiro (`.pgm`)									
	building		chessBig		chessRotate1		house		squares	
	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
2	1,87661	0,041595	1,86014	0,018049	1,72785	0,081518	1,81909	0,07098	1,69427	0,103244
4	3,28974	0,108358	3,36368	0,057014	2,04654	0,421171	2,87515	0,104962	2,41478	0,383376
8	3,06057	0,129783	3,36967	0,053015	1,62892	0,169157	2,48393	0,133386	2,13092	0,137024
16	3,53950	0,256117	4,00820	0,113067	1,42783	0,169184	2,47173	0,17301	1,66912	0,279263

Dado que a inclusão da opção `-m` apresenta mais uns passos ao algoritmo que também foi paralelizado os mesmos testes foram realizados[1] com a inclusão desta opção. Os valores de cada teste são apresentados na Tabela B com o seu resumo a ser apresentado na Tabela 2.

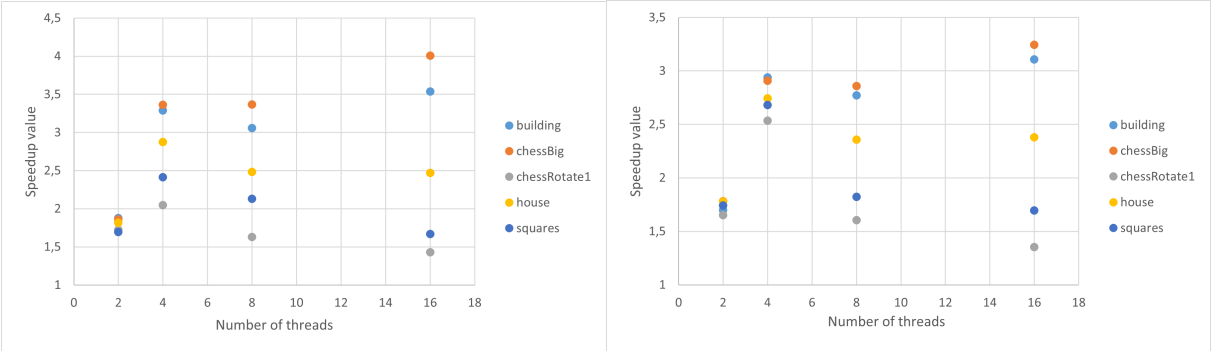
Tabela 2 – Valores dos *speedup* médios obtidos com o código desenvolvido em *OpenMP* com a opção `-m` repetindo 10 vezes para cada imagem consoante o número de *threads* utilizado.

Número de *Threads*	Ficheiro (`.pgm`)									
	building		chessBig		chessRotate1		house		squares	
	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
2	1,70016	0,05698	1,78237	0,03041	1,65215	0,069166	1,78088	0,064092	1,74350	0,084897
4	2,94089	0,142041	2,90836	0,105188	2,53658	0,120669	2,74507	0,161065	2,68114	0,160049
8	2,77088	0,217682	2,85829	0,142476	1,60700	0,124388	2,35803	0,122632	1,82292	0,089605
16	3,10896	0,256067	3,24466	0,230776	1,35311	0,119297	2,37913	0,15405	1,69478	0,058127

Para perceber melhor os resultados em função do número de *threads* podem-se visualizar os dados, Figura 1. Daqui percebe-se logo claramente que ao dividir-se o problema em duas *threads* o tempo de execução é logo melhor, *speedup* maior que 1 e até que 1,5. Contudo, este não consegue ser $2\times$ mais rápido devido aos requisitos que são necessários para lançar processos e aguardar que todos terminem. O mesmo acontece para as restantes situações inclusive aquando 8 ou mais processadores, dado que só existem 6 processadores o escalonador do sistema operativo é de grande influência nos resultados obtidos. Os valores de *speedup* obtidos encontram-se assim no intervalo de 1,42 até 4,00 com a inclusão da opção `-m` a apresentar um valor inferior face à sua contrapartida.

Analisando os valores de *speedup* obtidos com o tamanho de cada um dos ficheiros em análise, Tabela 3, consegue-se verificar que existe uma grande dependência entre estes. Para ficheiros de menor tamanho a adição de mais trabalhadores, *threads*, não apresenta qualquer benefício levando até a resultados piores como para `chessRotate1.pgm`, `house.pgm` e

[1] Utilizando o código da *bash* `run_open_mp_m.sh`



(a) Sem a opção `-m`.

(b) Com a opção `-m`.

Figura 1 – Comparativo dos *speedup* médios para cada imagem para cada ambas as situações de inclusão ou não da opção `-m`.

`squares.pgm`. Contudo, para ficheiros maiores verifica-se uma melhoria no *speedup*, `building.pgm` e `chessBig.pgm`. Acontecendo a mesma tendência quando se opta ou não pela utilização de `-m`, 1a e 1b.

Tabela 3 – Tamanho em disco ocupado por cada uma das imagens analisadas em `KB`.

Nome ficheiro (`.pgm`)	building	chessBig	chessRotate1	house	squares
Tamanho (`KB`)	319	2 023	40	77	76

Desta forma concluí-se que a implementação em *OpenMP* obteve resultados satisfatórios tendo conseguido de uma forma simples melhorado o desempenho do programa. Ainda assim, mais otimizações poderão ser feitas e estudadas mas estas dependerão da situação de aplicação, o tamanho da imagem, por exemplo, e até do próprio *hardware* e como este e o sistema operativo lidam com diversos processos.

CUDA

A outra implementação pedida para a realização deste projeto é o desenvolvimento do código em *CUDA*. Esta ferramenta foi criado pela empresa `NVIDIA` para ser uma extensão à linguagem `C`. Tendo como intuito guiar à abstração para utilizar sistemas massivamente multi paralelizados [1, 3].

Assim sendo, partiu-se do ficheiro `fastDetectorCuda.cu` em que se altera a função `fastDetectorDevice`. Esta parte do algoritmo em `fastDetectorHost` e fazem-se as modificações necessárias para paralelizar utilizando *CUDA*.

Processo

O desenvolvimento deste algoritmo partiu da mesma base que *OpenMP* de paralelizar através da análise de determinadas porções mais pequenas da imagem, neste caso, utilizou-se a

memória global do *device* e assim os *kernels* foram desenvolvidos como se estando a analisar cada *pixel* localmente.

Em `fastDetectorDevice` procede-se primeiramente à alocação de memória no *device*, e efetua-se a cópia da imagem de entrada para este, assim como de um espaço de memória para a imagem de saída. De seguida são definidos os parâmetros do tamanho do bloco e da grelha de forma a se conseguir maximizar a utilização do dispositivo atendendo que os *warp* são feitos em conjuntos de 32. Contudo, dado que as dimensões da grelha são dependentes do tamanho da imagem irão ocorrer situações em que *threads* não irão realizar trabalho útil, sendo lançadas mais que as exclusivamente necessárias.

Invoca-se então o *kernel*, `fastDetectCorners_CUDA`, para efetuar a deteção de cantos. Este calcula o índice da *thread* que está a utilizar, atendendo a se ainda é uma posição válida na imagem, e aplica o código da função `fastCorner`. A esta teve que ser adicionada as instruções de `__host__ __device__` para que seja compilada para ser usada tanto no *device* como no *host*. Procedendo-se da mesma forma para a função `fastScore` que é utilizada por ambos mas numa fase posterior. A `int offset[16]` foi também necessário adicionar `__managed__` para que pudesse ser acedido tanto pelo *host* como pelo *device* sem ser necessário efetuar as migrações de memória explicitamente e ficando a cargo do modelo de *Unified Memory*.

Após se efetuar a identificação de cantos verifica-se se foi escolhida a opção `-m`. Caso tenha sido então são alocadas no *device* duas regiões de memória cada uma com o tamanho da imagem. Aqui escolheu-se efetuar a divisão da função `nonMaximumSupression` que era fornecida em dois, para se determinar o *score* de cada um dos cantos, `cornerScore_Maximum_supression_CUDA`, e depois se fazer a supressão destes, `nonMaximumSupression_CUDA`. Optou-se por fazer esta separação uma vez que era mais fácil ter a certeza que o *score* de cada *pixel* já tinha sido obtido antes de se passar à supressão dos mesmos. Estas foram modificadas das originais para efetuar a identificação da *thread* na imagem e verificar se era uma das posições que se deseja analisar, dentro dos limites, e a cumprir as condições do algoritmo.

Para finalizar, é adicionada a imagem original escurecida aos resultados obtidos da deteção de cantos. Esta poderia ser feita já no *host* ou no *device* como é realizado com `faded_divider_CUDA`. Sendo por fim copiado para o *host* a imagem final com as devidas libertações de memória a serem realizadas no *device*.

Apresentam-se assim dois códigos, `fastDetectorCuda.cu` e `fastDetectorCuda_better.cu`, em que são utilizados os *device* 1 e 0 respetivamente, não se verificando mais qualquer distinção. Isto é efetuado porque se obtiveram resultados de desempenho distintos consoante o *device* que era utilizado, obtendo ambos o resultado correto em todas as repetições.[2]

[2] Ficheiro da *bash* `cuda.sh` que por si chama outros códigos, `run_cuda.sh`, `run_cuda_better.sh`, `run_cuda_m.sh` e `run_cuda_m_better.sh`.

Resultados

A obtenção destes resultados foi efetuada através do acesso remoto ao computador *banana* disponibilizado. Aplicando o comando `lshw` é possível ter uma ideia de qual o *hardware* aqui presente e verifica-se que o processador é um AMD Ryzen 9 3900x 12-core com 31GiB de memória. No sistema também se encontram duas placas gráficas, comando `nvidia-smi`, uma NVIDIA GeForce GT 710 e uma NVIDIA GeForce GTX 1660Ti. Dado que esta segunda é mais rápida que a primeira o seu `iD` é atribuído automaticamente pelo sistema com o valor 0 e a primeira com o valor 1. A compilação foi efetuada através da *Makefile* fornecida que utiliza `nvcc -arch=sm_30 -O3` como instrução.

Para avaliar o desempenho da paralelização foi utilizado o *speedup* como referência, equação 1. Na Tabela C são apresentados os tempos de cada uma das repetições efetuadas para a execução normal do processo em ambos os *devices* com a Tabela 4 a apresentar os valores sintetizados.

Contudo, estes resultados não podem ser interpretados desta maneira exclusiva sem primeiro se considerar que se estão a medir tempos de execução e que estes são obtidos em equipamentos diferentes. O *host* é um processador mais moderno, de 2019, assim como o *device* 0, face ao *device* 1 que é de 2014 apresentando um desempenho inferior. Tendo em conta a Tabela 3 percebe-se que para as imagens mais pequenas, `chessRotate1.pgm`, a vantagem é quase nula quando utilizando o *device* 1 ao contrário de quando o 0 que tem sempre um desempenho superior. Em ficheiros de dimensão superior, `chessBig.pgm`, verifica-se ainda mais a sua superioridade de desempenho em comparação ao *host* onde mesmo o *device* 1 é vantajoso.

Tabela 4 – Valores dos *speedup* médios obtidos com o código desenvolvido em *CUDA* repetindo 10 vezes para cada imagem consoante o *Device* utilizado.

Device	Ficheiro (`.pgm`)									
	building		chessBig		chessRotate1		house		squares	
	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
1	2,19568	0,02878	2,20946	0,04169	1,13271	0,01890	1,89881	0,02747	1,39228	0,02429
0	12,61395	0,79424	14,76909	0,20348	3,98554	0,37074	8,37726	0,76038	5,61190	0,53537

Como a inclusão da opção `-m` apresenta uma alteração significativa ao algoritmo esta é testada em separado sendo os resultados por extenso apresentados na Tabela D e o resumo na Tabela 5. Daqui é possível verificar que o *speedup* varia consoante o dispositivo que está a ser utilizado. O efeito que a dimensão da imagem apresenta é reforçado face ao visto na Tabela 4. Apenas para os maiores ficheiros existe uma vantagem em utilizar o *device* 1 uma vez que este consegue ser pior que o *host* em algumas situações, `building.pgm`, `chessRotate1.pgm` e `house.pgm`. Ainda assim o *device* 0 apresenta-se sempre como sendo melhor em todas as ocasiões.

Assim sendo, a implementação em *CUDA* obteve resultados satisfatórios conseguindo-se em algumas situações obter um desempenho quase 15$\times$ superior ao *host*. A escolha do *device*

Tabela 5 – Valores dos *speedup* médios obtidos com o código desenvolvido em *CUDA* com a opção `-m` repetindo 10 vezes para cada imagem consoante o *Device* utilizado.

Device	Ficheiro (`.pgm`)									
	building		chessBig		chessRotate1		house		squares	
	Média	σ	Média	σ	Média	σ	Média	σ	Média	σ
1	0,95188	0,01566	1,54614	0,02914	0,58592	0,00800	0,85900	0,01330	1,03989	0,01322
0	9,69944	0,70614	14,77941	0,49579	3,82113	0,06737	9,20838	1,68923	5,58310	0,93517

leva a uma diferença de desempenho significativa como se poderia já esperar devido a serem de gerações diferentes. Isto leva a que seja necessário efetuar escolhas consoante o *hardware* em uso pois o *host* ou o *device* poderão ser benéficos numa situação ou noutra. A opção de se ter escolhido o maior tamanho do bloco, 32 por 32, vem por se raciocinar que ao permitir um maior número de *threads* por bloco é possível aumentar o paralelismo e sendo que neste caso estas operações podem todas ser executadas em concorrência existiria vantagem. Este aspeto pode ser algo a estudar futuramente assim como a sua dependência pelo *device* utilizado devido à largura de banda do mesmo.

Ainda assim apenas se fez uso de memória global e existem outras maneiras que podem apresentar vantagens no tempo de acesso, *texture memory*. Para outras situações poder-se-ia ver como necessário colocar as instruções de `cudaDeviceSynchronize` mas nesta situação o mesmo não foi visto como sendo necessário até devido às cópias de memória serem síncronas.

Conclusões

Comparando os resultados obtidos consegue-se perceber que existe por vezes uma grande vantagem em aplicar métodos de execução em concorrência que levam a melhorias de desempenho bastante significativas. A aplicação em *OpenMP* foi de desenvolvimento mais simples e conseguiu obter resultados positivos. O desenvolvimento em *CUDA* já apresenta uma complexidade acrescida sendo penalizado com o *overhead* das transferências dos elementos em memória que afeta o desempenho possível de alcançar. Ainda assim, este consegue obter resultados bastante favoráveis sendo no geral melhor e mais vantajoso para situações de maior escala.

Outras formas de medir os *Speedup* torna-se também indispensável para trabalho futuro que poderia ter em conta o consumo energético. Este fica dependente de cada caso em específico e do *hardware* disponível mas em grande escala como em *server farms* pode ser visto como de bastante interesse.

Um aspeto a ter em conta é o facto que *CUDA* é exclusivo para placas gráficas da `NVIDIA` enquanto *OpenMP* é uma abordagem direta a muitos processadores multi-core estando disponível em diversos compiladores e sistemas operativos sem a necessidade de *drivers* específicos. Contudo, isto permite que seja possível tirar maior partido do sistema em uso e da enorme

quantidade de *threads* disponíveis para situações que possam ser altamente paralelizados.

Para outras aplicações pode até ser relevante combinar as duas técnicas para conseguir melhores resultados. Caso o *device* 0 não estivesse disponível poderia ser relevante utilizar uma ou outra abordagem consoante as dimensões da imagem a utilizar. Contudo, é necessário ter especial atenção no desenvolvimento de algoritmos que possam tomar partido destas ferramentas. Atendendo aos acessos à memória de forma independente minimizando a leitura em regiões de escrita para não se terem que efetuar momentos de espera.

Assim com este trabalho conseguiram-se estudar estes dois métodos de programação paralela e verificar que cada um apresenta os seus benefícios consoante a situação em que é utilizado.

Anexo

Tabela A – Tabela com os valores do tempo em (ms) para os ensaios de *OpenMP* consoante o número de *threads* e a imagem a analisar assim como os valores de *Speedup* obtidos e os valores médios.

Threads	Ficheiro (`.pgm`)	Executor	Repetição 1	2	3	4	5	6	7	8	9	10	Média	Min	Max	σ
2	building	host	10,99600	11,00300	11,36400	11,12800	10,88300	10,93500	10,96900	10,94000	11,05700	11,01900	11,02940	10,88300	11,36400	0,09216
		openmp	6,09100	5,76300	5,78300	5,85800	5,86000	6,15500	5,88600	5,77900	5,81700	5,78100	5,87730	5,76300	6,15500	0,10002
		speedup	1,80529	1,90925	1,96507	1,89962	1,85717	1,77660	1,86357	1,89306	1,90081	1,90607	1,87661	1,77660	1,96507	0,041595
	chessBig	host	55,80800	55,36700	54,91100	55,17200	57,18500	55,22500	55,20700	55,27400	55,47400	54,88800	55,45110	54,88800	57,18500	0,42274
		openmp	30,04400	29,84600	29,87400	29,50700	29,58300	29,92600	29,81700	30,25500	29,87000	29,37900	29,81010	29,37900	30,25500	0,19226
		speedup	1,85754	1,85509	1,83809	1,86979	1,93304	1,84539	1,85153	1,82694	1,85718	1,86827	1,86014	1,82694	1,93304	0,018049
	chessRotate1	host	1,13800	1,21400	1,13500	1,12700	1,15000	1,12400	1,18200	1,23300	1,18400	1,23300	1,17200	1,12400	1,23300	0,0372
		openmp	0,75000	0,65000	0,70800	0,65700	0,66000	0,65300	0,65900	0,66800	0,65100	0,72700	0,67830	0,65000	0,75000	0,03002
		speedup	1,51733	1,86769	1,60311	1,71537	1,74242	1,72129	1,79363	1,84581	1,81874	1,69601	1,72785	1,51733	1,86769	0,081518
	house	host	3,15800	3,20800	3,14000	3,14300	3,13800	3,10700	3,22500	3,28200	3,12900	3,19500	3,17250	3,10700	3,28200	0,044
		openmp	1,87200	1,68300	1,92800	1,71800	1,67700	1,72700	1,67000	1,76300	1,69400	1,70800	1,74400	1,67000	1,92800	0,0662
		speedup	1,68697	1,90612	1,62863	1,82945	1,87120	1,79907	1,93114	1,86160	1,84711	1,87061	1,81909	1,62863	1,93114	0,07098
	squares	host	1,88400	1,93600	1,80100	1,87000	1,89300	1,83000	1,87500	1,96500	1,85600	1,81000	1,87200	1,80100	1,96500	0,0386
		openmp	1,05600	1,07100	1,06800	1,02400	1,04000	1,22700	1,04900	1,25900	1,09900	1,15600	1,10490	1,02400	1,25900	0,06546
		speedup	1,78409	1,80766	1,68633	1,82617	1,82019	1,49144	1,78742	1,56076	1,68881	1,56574	1,69427	1,49144	1,82617	0,103244
4	building	host	10,87300	11,06600	11,21800	11,23600	11,14300	11,25100	11,11500	11,04300	11,23400	10,97800	11,11570	10,87300	11,25100	0,1007
		openmp	3,30400	3,32100	3,86500	3,24700	3,34900	3,39000	3,28500	3,29900	3,25200	3,47700	3,37890	3,24700	3,86500	0,11906
		speedup	3,29086	3,33213	2,90246	3,46043	3,32726	3,31888	3,38356	3,34738	3,45449	3,15732	3,28974	2,90246	3,46043	0,108358
	chessBig	host	55,74600	55,37100	55,32200	54,92300	55,65400	56,61100	56,71700	55,30400	55,03500	56,34300	55,70260	54,92300	56,71700	0,52132
		openmp	16,57200	16,91100	16,43500	16,65400	16,54800	16,24200	16,66900	16,57000	16,85200	16,14700	16,56000	16,14700	16,91100	0,1736
		speedup	3,36387	3,27426	3,36611	3,29789	3,36319	3,48547	3,40254	3,33760	3,26578	3,48938	3,36368	3,26578	3,48938	0,057014
	chessRotate1	host	1,18800	1,22600	1,15100	1,13200	1,30300	1,16900	1,20500	1,19500	1,14100	1,11900	1,18290	1,11900	1,30300	0,0405
		openmp	0,49800	0,52800	0,45000	1,08100	0,72000	0,69700	0,45600	0,48700	0,43300	0,43000	0,57800	0,43000	1,08100	0,1528
		speedup	2,38554	2,32197	2,55778	1,04718	1,80972	1,67719	2,64254	2,45380	2,63510	2,60233	2,04654	1,04718	2,64254	0,421171
	house	host	3,13400	3,38300	3,18700	3,19900	3,19400	3,16200	3,24300	3,16300	3,19300	3,17400	3,20320	3,13400	3,38300	0,04392
		openmp	1,26500	1,10700	1,08200	1,11800	1,08000	1,08600	1,09700	1,13700	1,09000	1,07900	1,11410	1,07900	1,26500	0,03554
		speedup	2,47747	3,05601	2,94547	2,86136	2,95741	2,91160	2,95624	2,78188	2,92936	2,94161	2,87515	2,47747	3,05601	0,104962
	squares	host	1,86700	1,86100	2,01500	1,86200	1,97200	1,88500	1,85700	1,81400	1,82500	1,82900	1,87870	1,81400	2,01500	0,04718
		openmp	0,64800	0,67400	0,66400	0,68200	1,14100	0,63700	1,20900	0,72200	0,65800	0,74500	0,77800	0,63700	1,20900	0,1588
		speedup	2,88117	2,76113	3,03464	2,73021	1,72831	2,95918	1,53598	2,51247	2,77356	2,45503	2,41478	1,53598	3,03464	0,383376
8	building	host	11,02000	10,80300	10,84300	10,90500	11,03900	11,01800	10,77100	10,92000	11,03500	11,03700	10,93910	10,77100	11,03900	0,0907
		openmp	3,47000	3,51500	4,06300	3,50100	3,44200	3,48400	3,47400	3,86800	3,38200	3,54300	3,57420	3,38200	4,06300	0,15652
		speedup	3,17579	3,07340	2,66872	3,11482	3,20715	3,16246	3,10046	2,82316	3,26286	3,11516	3,06057	2,66872	3,26286	0,129783
	chessBig	host	55,56300	55,05900	55,03500	55,28300	55,21800	55,20700	55,14100	55,83100	55,03200	54,68000	55,20490	54,68000	55,83100	0,2155
		openmp	16,48100	16,39700	16,00500	16,09800	16,10900	16,01200	16,67600	16,66100	16,39200	16,99800	16,38290	16,00500	16,99800	0,26152
		speedup	3,37134	3,35787	3,43861	3,43415	3,42777	3,44785	3,30661	3,35100	3,35725	3,21685	3,36967	3,21685	3,44785	0,053015
	chessRotate1	host	1,13700	1,16800	1,23000	1,18500	1,19300	1,12900	1,13900	1,12600	1,13500	1,15100	1,15930	1,12600	1,23000	0,02776
		openmp	1,00400	0,63400	0,68500	0,73100	0,65500	0,61100	0,61600	0,78700	0,68600	0,70800	0,71170	0,61100	1,00400	0,07738
		speedup	1,13247	1,84227	1,79562	1,62107	1,82137	1,84779	1,84903	1,43075	1,65452	1,62571	1,62892	1,13247	1,84903	0,169157
	house	host	3,18100	3,15300	3,12300	3,21700	3,13600	3,12200	3,13500	3,14500	3,16000	3,16400	3,15360	3,12200	3,21700	0,02152
		openmp	1,41000	1,33500	1,33000	1,27400	1,34000	1,16200	1,24900	1,20900	1,22900	1,15800	1,26960	1,15800	1,41000	0,0682
		speedup	2,25603	2,36180	2,34812	2,52512	2,34030	2,68675	2,51001	2,60132	2,57120	2,73230	2,48393	2,25603	2,73230	0,133386
	squares	host	1,81500	1,88600	1,83300	1,93800	1,85200	1,82600	1,84200	1,91800	1,84500	1,83300	1,85880	1,81500	1,93800	0,03312
		openmp	0,80900	0,81900	0,82200	1,09900	0,90600	0,80500	0,90700	0,82200	0,89500	0,83900	0,87230	0,80500	1,09900	0,06356
		speedup	2,24351	2,30281	2,22993	1,76342	2,04415	2,26832	2,03087	2,33333	2,06145	2,18474	2,13092	1,76342	2,33333	0,137024
16	building	host	10,87100	11,06600	11,16200	10,86500	11,20500	10,98600	11,24200	11,45800	11,05000	11,10800	11,10130	10,86500	11,45800	0,1337
		openmp	3,17300	2,90900	3,62900	3,61300	2,93800	3,02400	3,02500	3,30100	2,88000	2,87200	3,13640	2,87200	3,62900	0,23408
		speedup	3,42610	3,80406	3,07578	3,00720	3,81382	3,63294	3,71636	3,47107	3,83681	3,86769	3,53950	3,00720	3,86769	0,256117
	chessBig	host	55,69600	55,20700	55,37600	55,56500	55,87700	55,77600	54,82000	54,94900	55,42700	55,32500	55,40180	54,82000	55,87700	0,2664
		openmp	13,14300	13,21700	13,63500	13,91400	13,64500	14,29400	13,51700	14,10100	14,50200	14,25300	13,82210	13,14300	14,50200	0,3907
		speedup	4,23769	4,17697	4,06131	3,99346	4,09505	3,90206	4,05563	3,89682	3,82202	3,88164	4,00820	3,82202	4,23769	0,113067
	chessRotate1	host	1,29600	1,17400	1,13500	1,15000	1,14100	1,13200	1,14400	1,25100	1,17000	1,16800	1,17610	1,13200	1,29600	0,03896
		openmp	1,07100	0,89900	0,74300	0,86700	0,79600	0,71200	0,71800	0,65400	0,77500	1,00200	0,82370	0,65400	1,07100	0,10884
		speedup	1,21008	1,30590	1,52759	1,32641	1,43342	1,58989	1,59331	1,91284	1,50968	1,16567	1,42783	1,16567	1,91284	0,169184
	house	host	3,23000	3,15800	3,16000	3,14400	3,21900	3,24200	3,20700	3,15800	3,20300	3,15200	3,18730	3,14400	3,24200	0,0329
		openmp	1,23400	1,25800	1,36900	1,24000	1,23900	1,25900	1,46900	1,13700	1,51500	1,17500	1,28950	1,13700	1,51500	0,0969
		speedup	2,61750	2,51033	2,30825	2,53548	2,59806	2,57506	2,18312	2,77748	2,11419	2,68255	2,47173	2,11419	2,77748	0,17301
	squares	host	1,86800	1,87000	1,82600	1,92300	1,88300	1,85300	1,87400	1,84700	1,82500	1,86500	1,86340	1,82500	1,92300	0,02052
		openmp	0,86200	1,05800	0,95100	0,87700	1,11200	0,93100	1,04800	0,86800	0,88800	2,56900	1,11640	0,86200	2,56900	0,29052
		speedup	2,16705	1,76749	1,92008	2,19270	1,69335	1,99033	1,78817	2,12788	2,05518	0,72596	1,66912	0,72596	2,19270	0,279263

Tabela B – Tabela com os valores do tempo em (ms) para os ensaios de *OpenMP* com a opção `-m` consoante o número de *threads* e a imagem a analisar assim como os valores de *Speedup* obtidos e os valores médios.

Threads	Ficheiro (`.pgm`)	Executor	Repetição 1	2	3	4	5	6	7	8	9	10	Média	Min	Max	σ
2	building	host	13,86200	14,37500	13,45100	13,47500	14,12400	13,51800	13,45100	13,64000	13,27500	13,49500	13,66660	13,275	14,375	0,27224
		openmp	7,55800	8,41000	8,27000	7,51500	9,02200	8,13400	7,89800	7,84500	7,68200	8,05000	8,03840	7,515	9,022	0,3388
		speedup	1,83408	1,70927	1,62648	1,79308	1,56551	1,66191	1,70309	1,73869	1,72807	1,67640	1,70016	1,565507	1,834083	0,05698
	chessBig	host	63,85800	63,29900	70,25500	63,42000	64,70000	63,47300	62,70000	63,45700	62,71500	63,44400	64,13210	62,7	70,255	1,33816
		openmp	36,31000	34,89400	38,61600	35,51400	35,33200	36,06600	37,09900	35,34200	35,43500	35,20600	35,98140	34,894	38,616	0,83308
		speedup	1,75869	1,81404	1,81932	1,78577	1,83120	1,75991	1,69007	1,79551	1,76986	1,80208	1,78237	1,690073	1,831201	0,03041
	chessRotate1	host	1,52100	1,51000	1,59400	1,58100	1,49000	1,50800	1,55900	1,53100	1,55300	1,55600	1,54030	1,49	1,594	0,0283
		openmp	1,02200	0,89100	0,89400	0,94400	0,97200	0,94300	0,90500	0,92100	0,92000	0,91100	0,93230	0,891	1,022	0,03036
		speedup	1,48826	1,69473	1,78300	1,67479	1,53292	1,59915	1,72265	1,66232	1,68804	1,70801	1,65215	1,488258	1,782998	0,069166
	house	host	4,29100	4,36200	4,31600	4,28300	4,41100	4,28000	4,33500	4,36900	4,26600	4,23600	4,31490	4,236	4,411	0,0437
		openmp	2,32800	2,39600	2,55100	2,49300	2,51300	2,37000	2,34200	2,32800	2,58700	2,32100	2,42290	2,321	2,587	0,09048
		speedup	1,84321	1,82053	1,69189	1,71801	1,75527	1,80591	1,85098	1,87672	1,64901	1,82508	1,78088	1,649014	1,876718	0,064092
	squares	host	2,14400	2,09300	2,25500	2,10800	2,20700	2,10200	2,09900	2,09800	2,24600	2,16800	2,15200	2,093	2,255	0,0536
		openmp	1,16700	1,16500	1,20000	1,27300	1,22700	1,21000	1,36400	1,25300	1,31400	1,17000	1,23430	1,165	1,364	0,05336
		speedup	1,83719	1,79657	1,87917	1,65593	1,79870	1,73719	1,53886	1,67438	1,70928	1,85299	1,74350	1,538856	1,879167	0,084897
4	building	host	13,72400	13,51200	13,61600	14,07700	14,83100	13,86100	13,94100	13,74200	13,49200	13,40800	13,82040	13,408	14,831	0,28568
		openmp	4,37700	4,41200	4,50800	4,71600	6,28900	4,70500	4,59200	4,39500	4,34800	4,65200	4,69940	4,348	6,289	0,32236
		speedup	3,13548	3,06256	3,02041	2,98494	2,35824	2,94601	3,03593	3,12673	3,10304	2,88220	2,94089	2,358245	3,135481	0,142041
	chessBig	host	63,68600	63,59100	63,44400	62,95300	63,03800	63,53700	63,85000	64,01600	63,74300	64,17800	63,60360	62,953	64,178	0,291
		openmp	20,99900	21,55400	20,90800	21,32600	21,11000	26,50500	21,63500	21,04600	21,62600	21,98300	21,86920	20,908	26,505	0,94992
		speedup	3,03281	2,95031	3,03444	2,95194	2,98617	2,39717	2,95124	3,04172	2,94752	2,91944	2,90836	2,39717	3,041718	0,105188
	chessRotate1	host	1,50900	1,53800	1,60100	1,58300	1,52200	1,50900	1,51000	1,74300	1,57600	1,58000	1,56710	1,509	1,743	0,0495
		openmp	0,66000	0,68300	0,60300	0,61300	0,56600	0,57100	0,57400	0,69500	0,58900	0,62400	0,61780	0,566	0,695	0,03816
		speedup	2,28636	2,25183	2,65506	2,58238	2,68905	2,64273	2,63066	2,50791	2,67572	2,53205	2,53658	2,25183	2,689046	0,120669
	house	host	4,33700	4,58800	4,29900	4,30300	4,24300	4,30500	4,26800	4,32100	4,27800	4,34500	4,32870	4,243	4,588	0,05678
		openmp	1,55900	1,56000	1,56100	1,69800	1,93100	1,46200	1,49900	1,55000	1,47100	1,47800	1,57690	1,462	1,931	0,09504
		speedup	2,78191	2,94103	2,75400	2,53416	2,19731	2,94460	2,84723	2,78774	2,90823	2,93978	2,74507	2,197307	2,944596	0,161065
	squares	host	2,11000	2,74200	2,11700	2,11800	2,11100	2,15400	2,11400	2,11000	2,14100	2,07800	2,17950	2,078	2,742	0,1125
		openmp	0,78700	0,83200	0,83500	0,78800	0,76600	0,84000	0,82700	0,83000	0,76600	0,85800	0,81290	0,766	0,858	0,02892
		speedup	2,68107	3,29567	2,53533	2,68782	2,75587	2,56429	2,55623	2,54217	2,79504	2,42191	2,68114	2,421911	3,295673	0,160049
8	building	host	14,24500	13,46100	13,18900	13,77500	13,62300	13,66800	13,79200	13,85600	13,40600	13,45900	13,64740	13,189	14,245	0,2198
		openmp	5,26100	4,56900	6,88900	4,98900	4,48000	4,89200	4,54600	4,54000	4,60100	4,48600	4,92530	4,48	6,889	0,47262
		speedup	2,70766	2,94616	1,91450	2,76107	3,04085	2,79395	3,03388	3,05198	2,91371	3,00022	2,77088	1,914501	3,051982	0,217682
	chessBig	host	63,56100	63,18800	62,72600	62,95600	63,93500	63,77500	63,79300	63,76400	63,56400	63,74200	63,50040	62,726	63,935	0,32624
		openmp	20,47300	21,32600	22,76700	25,32400	22,95200	20,97600	22,00800	20,90200	23,09800	22,33600	22,21620	20,473	25,324	1,0792
		speedup	3,10463	2,96296	2,75513	2,48602	2,78560	3,04038	2,89863	3,05062	2,75193	2,85378	2,85829	2,486021	3,104626	0,142476
	chessRotate1	host	1,50700	1,57500	1,67700	1,55500	1,57100	1,54500	1,63600	1,55900	1,55100	1,53400	1,57100	1,507	1,677	0,035
		openmp	1,26700	0,95200	0,90900	1,03100	0,95800	0,99400	0,90900	0,91200	0,93000	0,91400	0,97760	0,909	1,267	0,07184
		speedup	1,18942	1,65441	1,84488	1,50824	1,63987	1,55433	1,79978	1,70943	1,66774	1,67834	1,60700	1,189424	1,844884	0,124388
	house	host	4,32100	4,39200	4,32300	4,37400	4,32800	4,36800	4,31000	4,32300	4,29800	4,30600	4,33430	4,298	4,392	0,02622
		openmp	1,85300	1,63800	1,86800	1,77900	1,78000	2,24700	1,88400	1,82300	1,75200	1,75700	1,83810	1,638	2,247	0,09992
		speedup	2,33189	2,68132	2,31424	2,45868	2,43146	1,94393	2,28769	2,37137	2,45320	2,45077	2,35803	1,943925	2,681319	0,122632
	squares	host	2,12300	2,11000	2,08700	2,20700	2,14200	2,09100	2,26700	2,13300	2,10000	2,13200	2,13920	2,087	2,267	0,03968
		openmp	1,16500	1,14000	1,13800	1,08300	1,20500	1,14900	1,16600	1,10900	1,12800	1,45200	1,17350	1,083	1,452	0,062
		speedup	1,82232	1,85088	1,83392	2,03786	1,77759	1,81984	1,94425	1,92335	1,86170	1,46832	1,82292	1,46832	2,037858	0,089605
16	building	host	13,59300	13,90000	13,81200	13,67600	14,59600	13,14100	13,38200	13,35900	13,28800	13,81100	13,65580	13,141	14,596	0,3032
		openmp	4,10500	5,06400	3,92200	3,85300	4,63300	5,04200	4,57300	4,31600	4,38800	4,02800	4,39240	3,853	5,064	0,34848
		speedup	3,31133	2,74487	3,52167	3,54944	3,15044	2,60631	2,92631	3,09523	3,02826	3,42875	3,10896	2,606307	3,549442	0,256067
	chessBig	host	64,00400	63,26500	64,05500	63,35700	63,57500	62,89500	63,10800	63,95500	62,87200	62,81100	63,38970	62,811	64,055	0,40604
		openmp	17,69000	21,49800	20,83000	18,56200	17,94000	18,80100	21,06000	18,41600	21,78100	18,78800	19,53660	17,69	21,781	1,40452
		speedup	3,61809	2,94283	3,07513	3,41326	3,54376	3,34530	2,99658	3,47280	2,88655	3,34314	3,24466	2,886552	3,618089	0,230776
	chessRotate1	host	1,52900	1,56900	1,50700	1,55700	1,52600	1,57000	1,56300	1,55000	1,54500	1,59200	1,55080	1,507	1,592	0,0194
		openmp	1,41200	1,28600	1,09100	1,00700	1,04200	1,03400	1,23000	1,18900	1,12500	1,04500	1,14610	1,007	1,412	0,10652
		speedup	1,08286	1,22006	1,38130	1,54618	1,46449	1,51838	1,27073	1,30362	1,37333	1,52344	1,35311	1,082861	1,546177	0,119297
	house	host	4,33100	4,29100	4,28000	4,43300	4,34200	4,29300	4,33500	4,31800	4,45400	4,37300	4,34500	4,28	4,454	0,045
		openmp	1,69100	1,93900	1,84200	1,86800	1,76800	1,64900	1,69300	2,16100	1,74200	1,91000	1,82630	1,649	2,161	0,1177
		speedup	2,56121	2,21300	2,32356	2,37313	2,45588	2,60340	2,56054	1,99815	2,55683	2,28953	2,37913	1,998149	2,603396	0,15405
	squares	host	2,20100	2,17400	2,11400	2,14200	2,23000	2,12000	2,10900	2,10000	2,10800	2,18000	2,14780	2,1	2,23	0,03876
		openmp	1,35500	1,25100	1,25900	1,21100	1,20500	1,25600	1,27400	1,33200	1,22400	1,30600	1,26730	1,205	1,355	0,03956
		speedup	1,62435	1,73781	1,67911	1,76879	1,85062	1,68790	1,65542	1,57658	1,72222	1,66922	1,69478	1,576577	1,850622	0,058127

Tabela C – Tabela com os valores do tempo em (ms) para os ensaios de *CUDA* consoante o *Device* utilizado e a imagem a analisar assim como os valores de *Speedup* obtidos e os valores médios.

Device	Ficheiro (`.pgm`)	Executor	Repetição										Média	Min	Max	σ
			1	2	3	4	5	6	7	8	9	10				
1	building	host	13,65712	13,65606	14,26947	13,7687	13,73798	13,84346	13,7728	13,78512	14,3217	13,79126	13,86037	13,65606	14,32170	0,17409
		cuda	6,29328	6,433792	6,317664	6,30352	6,2912	6,28912	6,289792	6,283104	6,305984	6,321824	6,31293	6,28310	6,43379	0,02690
		speedup	2,170112	2,122553	2,258663	2,184288	2,183683	2,201175	2,189707	2,193998	2,271128	2,181532	2,19568	2,12255	2,27113	0,02878
	chessBig	host	75,60397	76,34944	75,57529	76,15897	75,01718	75,19744	72,03738	72,09574	75,33158	72,25037	74,56174	72,03738	76,34944	1,46035
		cuda	33,76173	33,76819	33,72969	33,76586	33,76326	33,7817	33,67981	33,73469	33,77511	33,70064	33,74607	33,67981	33,78170	0,02789
		speedup	2,239339	2,260987	2,240616	2,255502	2,221858	2,225982	2,138889	2,13714	2,230388	2,143887	2,20946	2,13714	2,26099	0,04169
	chessRotate1	host	1,487872	1,542144	1,501184	1,50224	1,569792	1,552352	1,555488	1,57392	1,506336	1,560576	1,53519	1,48787	1,57392	0,02863
		cuda	1,355904	1,35152	1,350976	1,348896	1,35664	1,364032	1,351488	1,35904	1,352	1,362368	1,35529	1,34890	1,36403	0,00431
		speedup	1,097328	1,141044	1,111185	1,113681	1,157118	1,138061	1,150945	1,158112	1,114154	1,145488	1,13271	1,09733	1,15811	0,01890
	house	host	3,91168	3,828736	3,91168	3,772448	3,944448	3,74784	3,913728	3,867648	3,913728	3,7888	3,86007	3,74784	3,94445	0,06049
		cuda	2,040512	2,030688	2,037184	2,038624	2,035008	2,02224	2,043712	2,027776	2,025312	2,02752	2,03286	2,02224	2,04371	0,00615
		speedup	1,917009	1,885438	1,920141	1,850487	1,938296	1,853311	1,91501	1,907335	1,932407	1,868687	1,89881	1,85049	1,93830	0,02747
	squares	host	2,57744	2,564096	2,505728	2,483168	2,558976	2,563072	2,509824	2,473952	2,557952	2,590752	2,53850	2,47395	2,59075	0,03626
		cuda	1,815744	1,811584	1,786912	1,808512	1,881056	1,907616	1,802432	1,80496	1,806816	1,812032	1,82377	1,78691	1,90762	0,02823
		speedup	1,419495	1,415389	1,402267	1,373045	1,360393	1,3436	1,392465	1,370641	1,415724	1,42975	1,39228	1,34360	1,42975	0,02429
0	building	host	13,61133	20,96323	13,56803	14,20061	13,71674	13,65824	13,54259	14,11501	14,13171	13,60467	14,51122	13,54259	20,96323	1,29040
		cuda	1,216768	1,265504	1,120544	1,124	1,12704	1,115136	1,119648	1,14272	1,120608	1,118016	1,14700	1,11514	1,26550	0,03766
		speedup	11,18646	16,56513	12,10843	12,63399	12,17058	12,24805	12,0954	12,35211	12,61075	12,16858	12,61395	11,18646	16,56513	0,79424
	chessBig	host	75,59895	75,70023	75,11722	72,15075	75,264	72,61203	72,16445	72,60646	74,64765	73,00141	73,88631	72,15075	75,70023	1,37929
		cuda	5,046336	4,985792	5,053472	4,97616	4,995904	4,937312	4,96288	4,95952	5,069088	5,040032	5,00265	4,93731	5,06909	0,03967
		speedup	14,98096	15,18319	14,86448	14,49928	15,06514	14,70679	14,54084	14,63982	14,72605	14,48431	14,76909	14,48431	15,18319	0,20348
	chessRotate1	host	3,145664	1,504352	1,476256	1,756224	1,466336	1,553568	1,49296	1,481344	1,53328	1,541632	1,69516	1,46634	3,14566	0,30231
		cuda	0,54144	0,39008	0,38208	0,555808	0,390112	0,401536	0,396672	0,381888	0,395168	0,383968	0,42188	0,38189	0,55581	0,05070
		speedup	5,809811	3,856522	3,863735	3,159767	3,758756	3,869063	3,763714	3,879001	3,880071	4,015001	3,98554	3,15977	5,80981	0,37074
	house	host	3,8912	3,78784	3,883968	3,76624	3,739648	3,753248	8,024224	3,896128	3,905664	3,910784	4,25589	3,73965	8,02422	0,75367
		cuda	0,478464	0,470752	0,475456	0,546912	0,47008	0,484544	0,658848	0,470112	0,481568	0,473472	0,50102	0,47008	0,65885	0,04074
		speedup	8,132691	8,04636	8,168933	6,886373	7,955344	7,745938	12,17917	8,287659	8,110306	8,2598	8,37726	6,88637	12,17917	0,76038
	squares	host	2,47776	2,5192	2,764928	2,543648	5,173248	2,432	2,546944	2,522432	2,526432	2,541568	2,80482	2,43200	5,17325	0,47369
		cuda	0,456384	0,461184	0,637696	0,465088	0,624128	0,46832	0,467264	0,46528	0,456928	0,458816	0,49611	0,45638	0,63770	0,05392
		speedup	5,429112	5,462462	4,335809	5,469176	8,288761	5,19303	5,45076	5,42132	5,529169	5,539406	5,61190	4,33581	8,28876	0,53537

Tabela D – Tabela com os valores do tempo em (ms) para os ensaios de *CUDA* com a opção `-m` consoante o *Device* utilizado e a imagem a analisar assim como os valores de *Speedup* obtidos e os valores médios.

Device	Ficheiro (`.pgm`)	Executor	Repetição										Média	Min	Max	σ
			1	2	3	4	5	6	7	8	9	10				
1	building	host	16,45466	16,54067	17,20422	16,98301	16,39014	17,24621	16,35021	16,53146	16,51507	16,38298	16,65986	16,35021	17,24621	0,29077
		cuda	17,4848	17,43111	17,56832	17,51427	17,48858	17,50883	17,51674	17,56128	17,4527	17,49136	17,50180	17,43111	17,56832	0,03209
		speedup	0,941084	0,948917	0,979275	0,969667	0,937191	0,985	0,933405	0,941358	0,946276	0,936633	0,95188	0,93340	0,98500	0,01566
	chessBig	host	81,21446	80,02765	84,47283	84,02227	83,64954	82,9952	84,09904	79,95699	80,32768	83,29318	82,40588	79,95699	84,47283	1,61935
		cuda	53,21114	53,2368	53,3048	53,2735	53,56934	53,25197	53,30598	53,30067	53,22483	53,29264	53,29717	53,21114	53,56934	0,05843
		speedup	1,526268	1,503239	1,584713	1,577187	1,561519	1,558538	1,577666	1,500112	1,509214	1,56294	1,54614	1,50011	1,58471	0,02914
	chessRotate1	host	1,988608	1,897472	1,95888	1,942496	1,965056	1,974272	1,9712	1,98144	1,928192	1,897472	1,95051	1,89747	1,98861	0,02728
		cuda	3,44784	3,329984	3,297216	3,282944	3,328224	3,33136	3,332096	3,339296	3,28128	3,321888	3,32921	3,28128	3,44784	0,02690
		speedup	0,576769	0,569814	0,594101	0,591693	0,590422	0,592632	0,59158	0,593371	0,587634	0,571203	0,58592	0,56981	0,59410	0,00800
	house	host	5,211136	5,143552	4,984832	5,141504	5,02784	5,151744	4,97152	5,181408	5,147616	5,206016	5,11672	4,97152	5,21114	0,07319
		cuda	6,159104	5,898752	5,901056	6,138688	5,882976	5,932672	5,880032	5,91216	5,964064	5,904672	5,95742	5,88003	6,15910	0,07792
		speedup	0,846087	0,871973	0,844736	0,837557	0,854642	0,868368	0,845492	0,876398	0,863105	0,881677	0,85900	0,83756	0,88168	0,01330
	squares	host	2,792416	2,707488	2,751488	2,785216	2,783232	2,719744	2,814976	2,807808	2,832384	2,804736	2,77995	2,70749	2,83238	0,03223
		cuda	2,678176	2,649824	2,666368	2,657088	2,732832	2,662976	2,66704	2,670656	2,671808	2,677152	2,67339	2,64982	2,73283	0,01360
		speedup	1,042656	1,021761	1,031924	1,048221	1,018442	1,021318	1,055468	1,051355	1,0601	1,047657	1,03989	1,01844	1,06010	0,01322
0	building	host	16,13533	16,28262	23,78118	16,05546	16,75728	16,12566	16,88518	16,88615	16,05898	16,77926	17,17471	16,05546	23,78118	1,32129
		cuda	1,759936	1,745024	1,797504	1,761376	1,783712	1,760064	1,779104	1,77616	1,75312	1,778368	1,76944	1,74502	1,79750	0,01353
		speedup	9,168133	9,330888	13,23011	9,115292	9,394611	9,161976	9,490835	9,507108	9,160226	9,435204	9,69944	9,11529	13,23011	0,70614
	chessBig	host	86,48676	80,00035	81,84237	90,95248	85,82557	79,956	79,3912	82,03725	86,90624	82,67485	83,60731	79,39120	90,95248	3,14836
		cuda	5,837056	5,5992	5,64048	5,641024	5,61072	5,622944	5,621632	5,673056	5,65808	5,66384	5,65680	5,59920	5,83706	0,04096
		speedup	14,81685	14,28782	14,50982	16,1234	15,29671	14,2196	14,12245	14,46086	15,35967	14,59696	14,77941	14,12245	16,12340	0,49579
	chessRotate1	host	1,919712	1,926944	1,892896	1,899552	1,844768	1,931648	1,91664	1,859552	1,921536	1,929184	1,90424	1,84477	1,93165	0,02404
		cuda	0,496	0,51264	0,50928	0,49184	0,49648	0,494496	0,501792	0,475744	0,493344	0,513376	0,49850	0,47574	0,51338	0,00862
		speedup	3,870387	3,758864	3,716808	3,862134	3,715694	3,906297	3,819591	3,908724	3,894921	3,757838	3,82113	3,71569	3,90872	0,06737
	house	host	10,5504	4,83152	4,82672	9,550176	4,958208	10,50214	4,84352	5,06048	5,054464	5,066752	6,52444	4,82672	10,55040	2,20588
		cuda	0,843776	0,618496	0,62336	0,865568	0,605184	0,837888	0,612224	0,63008	0,625984	0,614336	0,68769	0,60518	0,86557	0,09683
		speedup	12,50379	7,811724	7,74307	11,03342	8,192893	12,53407	7,911353	8,031488	8,07443	8,247526	9,20838	7,74307	12,53407	1,68923
	squares	host	2,65488	2,66032	5,708992	2,66032	2,717984	2,629568	2,7096	5,722112	2,72544	2,717184	3,29064	2,62957	5,72211	0,96996
		cuda	0,53888	0,51536	0,72208	0,543616	0,538816	0,53968	0,532384	0,721056	0,540576	0,548	0,57404	0,51536	0,72208	0,05901
		speedup	4,926663	5,162061	7,906315	4,893749	5,044364	4,872458	5,089559	7,935739	5,041733	4,958365	5,58310	4,87246	7,93574	0,93517

Bibliografia

[1] Michael Garland et al. “Parallel Computing Experiences with CUDA”. Em: *IEEE Micro* 28.4 (2008), pp. 13–27. DOI: 10.1109/MM.2008.57.

[2] K. I. Kiy, D. A. Anokhin e A. V. Podoprosvetov. “A Software System for Processing Images with Parallel Computing”. Em: *Programming and Computer Software* 46.6 (nov. de 2020), pp. 406–417. ISSN: 1608-3261. DOI: 10.1134/S0361768820060043. URL: https://doi.org/10.1134/S0361768820060043.

[3] NVIDIA. *CUDA toolkit - free tools and training*. Abr. de 2023. URL: https://developer.nvidia.com/cuda-toolkit.

[4] Edward Rosten e Tom Drummond. “Machine Learning for High-Speed Corner Detection”. Em: *Computer Vision – ECCV 2006*. Ed. por Aleš Leonardis, Horst Bischof e Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443. ISBN: 978-3-540-33833-8.

[5] Hanan Shukur et al. “A state of art survey for concurrent computation and clustering of parallel computing for distributed systems”. Em: *Journal of Applied Science and Technology Trends* 1.4 (2020), pp. 148–154.

[6] János Végh, József Vásárhelyi e Dániel Drótos. “The Performance Wall of Large Parallel Computing Systems”. Em: *Reliability and Statistics in Transportation and Communication*. Ed. por Igor Kabashkin, Irina Yatskiv (Jackiva) e Olegas Prentkovskis. Cham: Springer International Publishing, 2019, pp. 224–237. ISBN: 978-3-030-12450-2.

[7] Xingyuan Wang, Le Feng e Hongyu Zhao. “Fast image encryption algorithm based on parallel computing system”. Em: *Information Sciences* 486 (2019), pp. 340–358. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.02.049. URL: https://www.sciencedirect.com/science/article/pii/S0020025519301641.