Complex Morphological Filtering For Serial, Parallel, GPU, SoC, PetaLinux And FPGA Execution

T. B. Almeida, E. C. Pedrino and M. M. Fernandes

***Abstract*— Image processing is a vast field of research and extremely important for a large number of applications such as security systems, geoprocessing, medical technologies, etc. There are some applications that require a higher level of processing, requiring higher computing power. As an example of this requirement in image processing, morphological filtering with filter chains involving erosion and dilation may be used. This study aims to discuss the advantages and disadvantages between running these chains in a software context, using a personal computer and in a hardware context, using the ZedBoard board running these filters in baremetal, FPGA and embedded linux modes. In addition, a discussion on the possibilities of parallel hardware processing, inspired by the multi-core environment and its power, will be discussed.**

***Index Terms*— Image processing, timing evaluation, morphological filtering, FPGA, embedded linux, many-core**

I. INTRODUÇÃO E CONTEXTO

O processamento digital de imagens é uma área de pesquisa cujos resultados são úteis em diversas aplicações. Como exemplos, temos o uso em sistemas de segurança e processamento de imagens de satélites. Muitas técnicas de processamento de imagens são extremamente custosas, entre elas estando a filtragem morfológica de imagens digitais[1]. Neste trabalho busca-se explorar a execução de filtros morfológicos grandes sobre imagens de alta e baixa resolução, além de alternativas de paralelização para a execução de cadeias de filtros deste tipo, operações que demandam muito do processamento. A proposta é que sejam discutidas alternativas para este tipo de processamento de imagens de forma que ele consiga ser usado em situações onde a economia de tempo de processamento é um requisito.

Este trabalho foi submetido na data 02 de dezembro de 2019. Ele é apoiado pela Universidade Federa l de São Carlos através do CoPICT – Coordenadoria dos Programas de Iniciação Científica e Tecnológica – e do CNPq – Centro Nacional de Pesquisa e Desenvolvimento - financiando o projeto de pesquisa que gerou este trabalho através do programa de bolsas PIBIC – Programa Institucional de Bolsas de Iniciação Científica.

T. B. de Almeida é aluno de graduação na Universidade Federal de São Carlos. Estuda no Departamento de Computação, São Carlos, São Paulo, Brasil, CEP 13565-905 (e-mail: tiago_almeida0297@hotmail.com)

E. C. Pedrino é professor na Universidade Federal de São Carlos. Trabalha no Departamento de Computação, São Carlos, São Paulo, Brasil, CEP 13565-905 (e-mail: ecpdrino@gmail.com).

M. M. Ferndandes é professor na Universidade Federal de São Carlos. Trabalha no Departamento de Computação, São Carlos, São Paulo, Brasil, CEP 13565-905 (e-mail: marcio@dc.ufscar.br).

O contexto no qual este trabalho se insere é o que diz respeito à execução dos chamados filtros morfológicos, especificamente os filtros de erosão e dilatação. Estes filtros, se aplicados individualmente, geralmente não exigem muito dos sistemas computacionais onde estão inseridos, mas quando aplicados em cadeias, sobre imagens de grande resolução e utilizando-se filtros também volumosos, podem exigir bastante do sistema de processamento. Algumas aplicações de filtros morfológicos podem ser encontradas nos trabalhos [2], [3] e [4].

Existem diversos casos onde é necessária a aplicação de cadeias de filtros morfológicos. Um exemplo clássico deste tipo de necessidade é na identificação de componentes em esquemáticos de circuitos impressos, comumente chamados de PCBs[5]. Com combinações de erosões e dilatações, variando ainda o tamanho do elemento estruturante que será suporte da filtragem, consegue-se, por exemplo, isolar os capacitores de um circuito. Estas operações são utilizadas também na eliminação de diversos tipos de ruídos, no reconhecimento de digitais, reconhecimento de texto, segmentação de formas em uma imagem, entre outras aplicações além das já elencadas aqui [6][7][8].

Sobre o uso de cadeias de filtros morfológicos, de fato, é difícil que um programador consiga identificar quais combinações de erosões e dilatações geram quais resultados em uma dada imagem, ainda mais se for levado em conta que este tipo de cadeia de filtros morfológicos pode contar com centenas de filtros. Os trabalhos [9] e [10] dão base para a geração de cadeias de filtros morfológicos com o uso de algoritmos genéticos. Em [9] a filtragem foi aplicada para a detecção de artefatos em imagens de partituras musicais, como notas, marcações de tempo, linhas do compasso, etc. Vale ressaltar, que a execução de cadeias de filtros deste tipo, embora muito úteis, são extremamente custosas, principalmente quando aplicados a sistemas embarcados de pouca memória e com processadores mais limitados.

Buscando alternativas que possam tornar o processamento de filtros morfológicos mais viável, este trabalho busca analisar abordagens em *software* e *hardware* para sua execução. O processamento em *software* busca comparar o processamento de filtros morfológicos utilizando-se o MATLAB nas modalidades de processamento serial, em paralelo e em GPU. O processamento em *hardware*, por sua vez, busca avaliar o desempenho da *ZedBoard* nas modalidades de processamento utilizando o processador embarcado sem SO, utilizando FPGA[11] e utilizando o processador embarcado com Linux embarcado, o

PetaLinux[12]. As abordagens citadas serão também comparadas entre si em um âmbito geral. Alguns trabalhos já foram publicados buscando avaliar as diferenças de desempenho entre *software* e *hardware* para o processamento digital de imagens. [13] avalia o desempenho destas duas categorias utilizando tecnologias muito semelhantes às utilizadas neste trabalho, mas abordando o cenário de sistemas reconfiguráveis. [14] avalia as diferenças de desempenho entre *hardware* e *software* na tentativa de acelerar a execução de algoritmos de processamento de imagens coloridas.

Por fim, busca-se avaliar alternativas de processamento paralelo utilizando-se uma cadeia de *ZedBoards*, tentando simular um ambiente *many-core,* inspirado nos trabalhos [15], [16], [17], [18] e [19]. A importância de se iniciar o estudo de processamento *many-core* se dá devido à relevância cada vez maior que esta tecnologia vem ganhando. Várias áreas de pesquisa podem ser abordadas a partir do processamento *many-core*, como alocação inteligente de tarefas para os núcleos de processamento, roteamento de informações, distribuição de energia, entre outros. Alguns destes pontos serão ainda focos de trabalhos futuros dos pesquisadores e podem também ser encontradas nos textos de referência sobre o tópico.

Para esta abordagem, serão feitas considerações sobre modalidades de comunicação e ganhos de processamento, além de serem discutidos possíveis cenários onde este tipo de execução de filtros pode ser utilizado. Esta avaliação será teórica, visto que os pesquisadores não dispõem de uma cadeia de *ZedBoards* para implementação do sistema.

II. MATERIAIS E MÉTODOS

A. Ambiente de Pesquisa e Preparação

Para o desenvolvimento da pesquisa foi utilizado um computador com processador *Intel i5*, com memória RAM de 8GB e *GPU Nvidia GeForce GTX 1050 Ti 4GB*. Neste ambiente foram executados os códigos em MATLAB. Para o caso de processamento parelelo na ferramenta foram utilizados 4 núcleos de processamento, o máximo permitido pelo computador.

Além disso, foi utilizada a placa *ZedBoard*, onde se utilizou a linguagem de programação C para a codificação dos filtros.

Os códigos feitos foram utilizados tanto para a execução dos filtros utilizando o processador embarcado, um *Dual ARM Cortex A9 MPCore com 667MHz* de processamento, sem nenhum tipo de sistema operacional, quanto com o *PetaLinux*, o Linux embarcado utilizado. Em ambos os casos, o programa é executado na seção PS (*Processor System*) da *ZedBoard*, onde está o processador embarcado citado. Para a programação deste processador sem o sistema operacional foi utilizado o *Vivado SDK*[20], que permite a codificação e execução de programas diretamente no *ARM Cortex A9*. Para a programação do processador com o SO *PetaLinux*, um programa foi feito e compilado em computador pessoal e depois gravado na *ZedBoard* com o Linux embarcado. Para a FPGA, o bloco de processamento responsável foi também feito a partir de um código em linguagem de programação C utilizando o *Vivado HLS*[21] para se gerar o *IP Core* responsável pelo filtro morfológico que será colocado na seção PL(*Programmable Logic*), onde está a FPGA. O código de transferência da imagem da seção PS para a PL e de obtenção da imagem resultado foi também feito com a linguagem de programação C utilizando o *Vivado SDK* e programado na seção PS. Ou seja, a seção PS deve enviar a imagem para o *IP Core* da seção PL, que realizará a filtragem e enviará de volta para a seção PS, onde se obtém a imagem filtrada. Para mais informações sobre a arquitetura da *ZedBoard*, podem ser consultadas as referências [22] e [23].

Em todos os casos, o programa foi feito utilizando-se a versão força bruta do algoritmo de convolução, para simplificar a avaliação e a comparação dos cenários onde os filtros foram executados, evitando-se o uso de otimizações que poderiam advir do modo como o programa foi feito.

O caso de teste consiste na execução sistemática de três tipos de filtragens: convolução simples com filtro de média, erosão e dilatação. Para todos os casos foram utilizados elementos estruturantes matriciais com todos os elementos ativos. Para cada caso, foram testados os elementos estruturantes com tamanhos 3x3, 5x5 e 7x7**.** A Fig. 1 mostra a imagem utilizada para os testes. Utilizou-se duas imagens idênticas à da figura, ambas em tons de cinza, mas com resoluções 320x240 e 1280x1024. A Fig. 2 representa um cenário de exemplo, explicitando como é o fluxo da execução dos experimentos e da coleta de dados.

Fig. 1. Imagem utilizada para os testes.

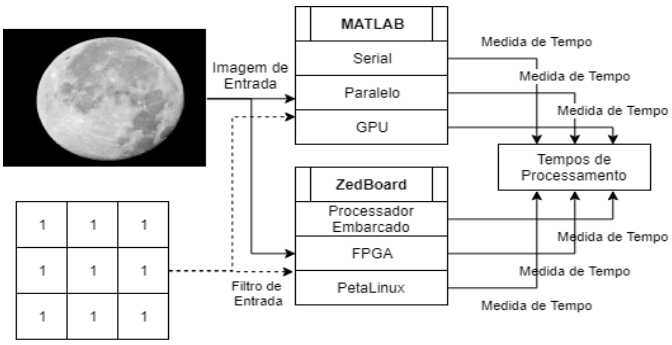

Fig. 2. Esquemático de execução dos experimentos e coleta de dados.

B. Comunicação entre ZedBoards

Com o objetivo de se estudar o processamento paralelo de cadeias de filtros morfológicos, foi feito um experimento de

comunicação e filtragem utilizando a *ZedBoard* e o computador. No experimento, a *ZedBoard*, equipada com o *PetaLinux* e utilizando *socket* para comunicação via *Ethernet*, deve enviar uma imagem para o computador e, após este receber, ambos devem realizar suas respectivas filtragens. O *PetaLinux* com *socket* foi escolhido pois permite maior flexibilidade de programação, embora de fato não seja a alternativa de comunicação de dados mais eficiente disponível. Também seria ideal a utilização de outra *ZedBoard* no lugar do computador, mas infelizmente os pesquisadores não tiveram acesso a este recurso.

A ideia deste experimento é obter dados quantitativos sobre o tempo de comunicação da imagem para outros sistemas.

Assim, consegue-se discutir com mais propriedade as vantagens e desvantagens de se utilizar processamento paralelo para imagens em sistemas embarcados, buscando uma aproximação com o que é possível em sistemas *many-core*, foco de trabalhos futuros dos pesquisadores. Também serviu de base para o início do projeto de um sistema de comunicação que pode ser integrado em uma rede de processamendo de *ZedBoards* com *PetaLinux*. A Fig. 3 exibe um diagrama do sistema montado, onde há um computador pessoal enviando uma imagem via *socket* para a *ZedBoard* e, após isso, cada um executa sua respectiva filtragem. A Fig. 4 exibe uma representação do que o estudo está tentando aproximar: a comunicação e execução de fitlros com inúmeros nós de processamento, como exemplo, inúmeras *ZedBoards*.

Fig. 3. Esquemático da comunicação e filtragem paralela usando a *ZedBoard* e o computador pessoal.

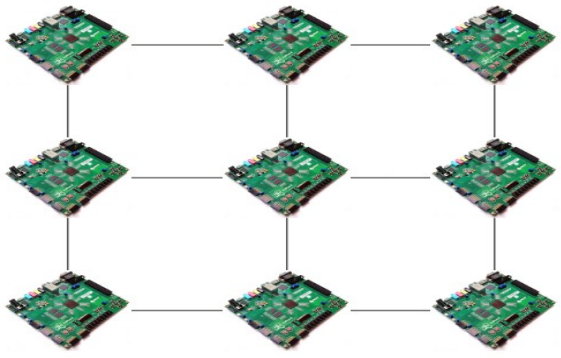

Fig. 4. Exemplo de arquitetura de processamento utilizando várias *ZedBoards*.

C. Execução e Obtenção dos Tempos de Processamento

Para cada imagem de entrada, cada filtro morfológico foi executado dez vezes, medindo-se o tempo de processamento em cada execução. Isso foi repetido para cada modalidade de processamento, ou seja, tanto no MATLAB com processamento serial, em paralelo e com GPU, quando na *ZedBoard* com a execução em processador embarcado, em FPGA e com *PetaLinux*. O tempo de processamento final foi considerado a média destes dez valores obtidos. Isto foi feito para se obter uma maior fidelidade nas medições, menos suscetível a interferências externas às previstas pelos pesquisadores.

É válido ressaltar que no cenário de execução com FPGA, deve-se fazer um sistema onde o processador embarcado, que possui a imagem e o filtro, envia através de um AXI DMA as entradas(imagem e elemento estruturante) para o módulo de processamento em FPGA e, após feita a execução da filtragem, recebe a imagem de resultado pelo mesmo barramento. Isso se dá devido ao fato do processador e da malha FPGA ficarem em áreas diferentes da *ZedBoard*, respectivamente, na região *Processing System*, ou PS, e *Programmable Logic*, ou PL. Assim sendo, o tempo de processamento foi medido obtendo-se o intervalo imediatamente antes de enviada a entrada pelo barramento para a FPGA e imediatamente depois de lido o resultado do barramento. A Fig. 5 exibe o diagrama da implementação desta modalidade de processamento.

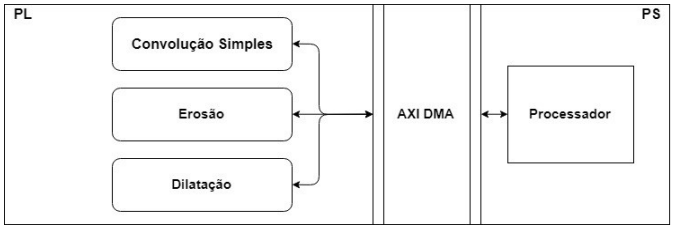

Fig. 5. Esquemático do sistema de execução com FPGA.

III. RESULTADOS E DISCUSSÃO

A. Medidas de Desempenho

O processamento da imagem original resultou nas seguintes imagens de saída. A Fig. 6 mostra as imagens obtidas após a convolução simples com os filtros de dimensões 3x3, 5x5 e 7x7, respectivamente. A Fig. 7 mostra imagens obtidas após a dilatação com os filtros de dimensões 3x3, 5x5 e 7x7, respectivamente. A Fig. 8 mostra imagens obtidas após a erosão com os filtros de dimensões 3x3, 5x5 e 7x7, respectivamente.

Fig. 6. Resultado da convolução com filtros 3x3, 5x5 e 7x7.

Fig. 7. Resultado da dilatação com filtros 3x3, 5x5 e 7x7.

Fig. 8. Resultado da erosão com filtros 3x3, 5x5 e 7x7.

As tabelas a seguir exibem os tempos de processamento obtidos para cada modalidade e serão alvo da discussão principal desta pesquisa. A Tabela I, Tabela II e Tabela III mostram o tempo de processamento obtido com o MATLAB utilizando-se, respectivamente, processamento serial, paralelo e em GPU. A Tabela IV, Tabela V e Tabela VI mostram o tempo de processamento obtido com a *ZedBoard* utilizando-se, respectivamente, o processador embarcado sem sistema operacional, a FPGA e o *PetaLinux*.

Uma observação que deve ser feita é que a medida de tempo de processamento para a modalidade de execução em *ZedBoard* com FPGA para a imagem de tamanho 1280x1024 foi feita através de uma regressão linear dos dados da imagem de tamanho 320x240. Não foi possível implementar na prática o processamento para esta imagem utilizando a *ZedBoard*. Quando se tentou realizar a implementação e a execução, percebeu-se que, embora tanto a imagem quanto o filtro coubessem na memória da placa, não se conseguia fazer a comunicação da imagem da seção PS para a PL através do AXI DMA, possivelmente devido a falhas no *buffer*. Foram feitos grandes esforços para se solucionar o problema da implementação, inclusive com o apoio de engenheiros da *Xilinx*, mas mesmo assim não foi possível fazer este processamento funcionar, por motivos desconhecidos até o momento da publicação deste texto. Os demais dados foram obtidos normalmente, através da implementação da filtragem e medição dos tempos de processamento associados. Estes resultados serão base de dicussões futuras sobre as vantagens do uso de *hardware* em ambientes paralelos, de forma a sustentar seu uso no lugar de computadores mais robustos, como o que foi usado para o processamento em *software* com o MATLAB.

TABELA I
MATLAB – EXECUÇÃO SERIAL

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,0066145760	0,00032131386
5x5 - Convolução Simples	0,0077974820	0,00038036106
7x7 - Convolução Simples	0,0096272510	0,00052655400
3x3 - Dilatação	0,0035808780	0,00214357492
5x5 - Dilatação	0,0546110790	0,00319090416
7x7 - Dilatação	0,0554857200	0,00327495401
3x3 - Erosão	0,0034063030	0,00208873635
5x5 - Erosão	0,0543658990	0,00316267591
7x7- Erosão	0,0548861450	0,00342914097

TABELA II
MATLAB – EXECUÇÃO PARALELA

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,1757301590	0,15908919032
5x5 - Convolução Simples	0,1727721380	0,15668199024
7x7 - Convolução Simples	0,1723455470	0,15570564804
3x3 - Dilatação	0,1859127850	0,18465680669
5x5 - Dilatação	0,1952243000	0,16439561865
7x7 - Dilatação	0,1947253060	0,16634065038
3x3 - Erosão	0,1834672660	0,18590355976
5x5 - Erosão	0,1941942570	0,16963795828
7x7- Erosão	0,1944366850	0,17074382614

TABELA III
MATLAB – EXECUÇÃO EM GPU

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,0001149440	0,00010861559
5x5 - Convolução Simples	0,000097500	0,00010758476
7x7 - Convolução Simples	0,000091500	0,00011004919
3x3 - Dilatação	0,0006587010	0,00068941424
5x5 - Dilatação	0,0011790900	0,00124270486
7x7 - Dilatação	0,0011747790	0,00118934427
3x3 - Erosão	0,0003775960	0,00037360949
5x5 - Erosão	0,0008096040	0,00080312934
7x7- Erosão	0,0007902200	0,00079952145

TABELA IV
ZEDBOARD – EXECUÇÃO EM PROCESSADOR EMBARCADO

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,156929200	0,009078000
5x5 - Convolução Simples	0,713871800	0,041347000
7x7 - Convolução Simples	1,914132000	0,110889000
3x3 - Dilatação	0,834366000	0,048427000
5x5 - Dilatação	2,082462000	0,119584000
7x7 - Dilatação	3,857400000	0,220533000
3x3 - Erosão	0,845210000	0,048679000
5x5 - Erosão	2,083500000	0,119347000
7x7- Erosão	3,856700000	0,219210000

TABELA V
ZEDBOARD – EXECUÇÃO EM FPGA

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,1194154660	0,0069970000
5x5 - Convolução Simples	0,3294378660	0,0193030000
7x7 - Convolução Simples	0,6444373330	0,0377600000
3x3 - Dilatação	0,1194154660	0,0069970000
5x5 - Dilatação	0,3294378660	0,0193030000
7x7 - Dilatação	0,6444373330	0,0377600000
3x3 - Erosão	0,1194154660	0,0069970000
5x5 - Erosão	0,3294378660	0,0193030000
7x7- Erosão	0,6444373330	0,0377600000

TABELA VI
ZEDBOARD – EXECUÇÃO EM PETALINUX

Filtro - Operação	Tempo (s)	
	1280x1024	320x240
3x3 - Convolução Simples	0,2331420000	0,0133440000
5x5 - Convolução Simples	0,5281030000	0,0295120000
7x7 - Convolução Simples	0,9656460000	0,0548410000
3x3 - Dilatação	0,3304250000	0,0206990000
5x5 - Dilatação	1,8547650000	0,0433730000
7x7 - Dilatação	1,2566160000	0,0717470000
3x3 - Erosão	0,2930690000	0,0177550000
5x5 - Erosão	0,6695020000	0,0381800000
7x7- Erosão	1,0878250000	0,0617770000

B. *Comparação entre Hardware e Software*

De fato, a execução dos filtros em *software* se mostrou muito mais rápida se comparada à execução em *hardware*, com atenção especial ao processamento em GPU, que demonstrou uma performance altíssima, mesmo dentro do grupo de processamento em *software*. Um ponto que vale a pena ser ressaltado sobre a GPU, na Tabela III, é que o tempo de processamento não varia muito entre a imagem de dimensões 1280x1024 e a de dimensões 320x240, sendo que possivelmente há um limite mínimo de tamanho da imagem abaixo do qual o tempo de processamento desta modalidade permanece o mesmo. Na mesma tabela estão destacados os pontos onde o processamento da imagem de tamanho 320x240 demorou mais do que o da 1280x1024. Vale observar também a ocorrência de casos onde processamentos com filtros menores tomaram mais tempo do que o mesmo processamento com filtros maiores, como pode ser percebido pela convolução simples com filtro de tamanho 3x3 em relação à convolução simples com filtro de tamanho 7x7. Não se sabe ao certo o motivo desta diferença, mas cogita-se que este resultado possa vir de mecanismos de inicialização e alocação de dados da GPU. Vale ressaltar que devido à magnitude extremamente pequena dos dados, para análise geral dos resultados, estes casos anômalos não invalidam as conclusões deste trabalho.

O processamento paralelo do MATLAB, por outro lado, foi muito mais lento do que o esperado. Isso se deve à grande demora do MATLAB em preparar os núcleos de paralelização. Apesar disso, é perceptível que o processamento paralelo do MATLAB pode ser vantajoso com entradas e saídas ainda maiores do que os utilizados aqui, visto que este tempo de preparação, conforme se aumenta o tamanho dos elementos estruturantes e da imagem de entrada, torna-se irrelevante se comparado ao tempo de processamento necessário para a filtragem em si.

Sobre o *hardware*, o processamento em FPGA foi o mais rápido. Algo impressionante se analisadas as diferenças entre as configurações da *ZedBoard* e do computador.

Contrário ao esperado, a aplicação em *PetaLinux* superou a aplicação rodando em processador embarcado. Isso se deu devido ao fato das versões mais modernas do *PetaLinux* possuírem otimizações de códigos que não são feitas no processador embarcado. Pode-se dizer que o *PetaLinux* conseguiu otimizar bastante o programa executado, mesmo este sendo feito com o algoritmo de força bruta, de forma que a velocidade de processamento conseguiu tornar a presença de um SO e de todas as operações extras que poderiam atrasar a operação insignificantes, superando o processamento em *hardware* sem SO, que não sofre com estas operações extras inerentes ao uso do *PetaLinux*, mas que não possui otimizações inteligentes de código. Na Tabela IV estão destacados os poucos pontos onde o processador embarcado foi mais rápido em relação ao *PetaLinux*.

C. *Análise do Processamento Paralelo com ZedBoards*

O poder desta arquitetura reside no paralelismo. Imaginando um caso onde a *ZedBoard* leva E segundos e D segundos para realizar, respectivamente, as operações de erosão e dilatação utilizando o *PetaLinux*. Ainda, utilizando-se o sistema de transmissão via *socket* do *PetaLinux*, leva-se C segundos para enviar ou receber uma imagem da placa vizinha. Tendo um sistema de duas placas realizando este processamento serialmente, o tempo T que elas levam é de aproximadamente $T = C + E + D$, considerando o cenário onde a primeira placa irá erodir a imagem, depois enviá-la para a segunda placa e então esta irá realizar a dilatação.

Em muitos casos de cadeias de filtros morfológicos, têm-se que os filtros desta cadeia não precisam ser executados de

forma sequencial, podendo ser executados paralelamente sobre a imagem de entrada e, unidos os resultados de cada filtro com operações lógicas, se obter a mesma imagem que se teria executando um filtro de cada vez sobre a imagem gerada pelo filtro imediatamente anterior. Para exemplificar, supõe-se que o resultado em uma dada imagem de se realizar uma erosão na imagem inicial e depois uma dilatação na imagem erodida seja o mesmo que o obtido se realizada a erosão na imagem inicial, a dilatação também na imagem inicial e depois a união dos dois resultados com uma operação OR, que leva *OR* segundos. O trabalho [3] serve de base para este teste, demonstrando casos parecidos. Nele é exemplificado que podem existir cadeias de filtros onde esta propriedade é válida.

O tempo gasto seria de $T = C + D + OR$ apenas, considerando-se que a dilatação tome mais tempo que a erosão.

Estendendo-se este cálculo para 100 placas, considerando-se que cada placa deve dilatar a imagem e transmitir para a placa seguinte, que usará a imagem transmitida em sua dilatação, o tempo gasto serialmente seria dado por $T = 99C + 100D$.

Paralelamente, seria algo próximo de $T = 99C + D + 99OR$, com potencial de ser ainda muito mais rápido do que isso, pois não está se considerando as operações OR's que podem ser feitas em paralelo e também que todas as placas estarão em espera até a última receber a imagem.

Para exemplificar, pode-se imaginar o caso de uma cadeia de filtros sendo executada em uma imagem 1280x1024 com elemento estruturante de tamanho 7x7. A cadeia é formada pela sequência de 100 filtros de erosão, 100 filtros de dilatação e 200 filtros de erosão. Toma-se como premissa que a taxa de transferência por *socket* da *ZedBoard* com *PetaLinux* seja em torno de *250Mbps*, a menor taxa que se consegue com a placa. Além disso, supõe-se que a realização da operação OR tenha tempo de processamento insignificante perto do tempo necessário para se realizar a filtragem e a transmissão.

Vale lembrar que utilizando-se o MATLAB não há taxa de transferência, pois tudo estará sendo executado no mesmo computador.

Nota-se que a transmissão de 1280x1024x8 bits dura 0.0042 segundos e que a erosão e dilatação no MATLAB, de acordo com a Tabela I duram aproximadamente 0.055 segundos, sendo que utilizando a *ZedBoard* com *PetaLinux*, idealmente, leva-se 1.1 segundo para a erosão e 1.2 segundo para a dilatação, como pode ser visto na Tabela VI. Utilizando o MATLAB serialmente, levaria um tempo $T = 400 \times 0.055 = 22.0$ segundos.

Supõe-se que o resultado obtido seja exatamente o mesmo que se obteria filtrando a imagem de entrada separadamente e unindo os resultados com uma operação OR. Ou seja, não será uma filtragem consecutiva entre as operações de filtragem, onde cada filtro irá filtrar a imagem resultado do anterior.

Agora todos irão filtrar a imagem original simultaneamente e unir os resultados. Utilizando-se 400 *ZedBoads* com *PetaLinux*, uma para cada filtragem, considerando-se que todas as *Zedboards* devem receber a imagem da inicial antes de filtrar e que uma recebe de cada vez, tem-se o tempo de processamento $T = 399 \times 0.0042 + 1.2 = 2.87$ segundos. Usa-se o valor da dilatação pois, como de acordo com a Tabela VI, a erosão demora cerca de 1.1 segundo e a dilatação dura cerca de 1.2 segundo, sendo o tempo de execução paralela delimitado pela dilatação. É possível se obter um processamento 7.7 vezes mais rápido.

De fato, a GPU ainda é cerca de 9 vezes mais rápida que a malha de *ZedBoards*, com tempo de processamento $T = 400 \times 0.0008 = 0.32$ segundos.

Observa-se que a necessidade de comunicação é um limitante para se equiparar a aplicação paralela em *ZedBoards* com a totalmente em *software* no MATLAB. Entre as possibilidades de comunicação, para a implementação utilizou-se o *socket* provido pelo *PetaLinux*, que pode operar na faixa de *250Mbps* a *700Mbps* na *ZedBoard*. De fato, existem possibilidades mais eficientes para se conectar a malha de *hardware*. Porém, optou-se por comunicação via *socket* pois permite maior flexibilidade de programação dos sistemas de comunicação e processamento.

IV. CONCLUSÃO

De fato, na maioria das vezes o processamento em *software* se mostrou mais rápido do que o processamento em *hardware* em um cenário serial. Porém, pelos cálculos mostrados, fica evidente que conforme a complexidade da filtragem e o grau de paralelização aumentam a abordagem paralela utilizando *ZedBoards* se torna muito vantajosa embora ainda perca para a GPU.

Em aplicações reais, muitas vezes não é possível usufruir de um computador como o que executou o MATLAB, com alta taxa de processamento e GPU integrada, sendo necessário o trabalho com tecnologias mais limitadas, como aplicações diretamente em *hardware*, com ou sem SO embarcado ou FPGA. Também, tem-se como ponto chave a questão da mobilidade. Aplicações em *hardware* comumente são mais vantajosas em sistemas nos quais a mobilidade seja um fator determinante. Com isso em mente, percebe-se que há ainda vantagens em relação ao tempo de processamento de um sistema embarcado, principalmente se pensarmos em abordagens paralelas como a *many-core*. Pode ser completamente inviável a utilização de 400 placas de prototipação como a *ZedBoard* para se processar uma imagem. Mas o ponto do exemplo é mostrar que, embora 400 *ZedBoards* em série seja algo utópico, 400 nós de processamento distribuídos em um chip, por exemplo, não o são. Têm-se disponíveis hoje processadores de baixa velocidade, com tamanho e consumo energético bastante reduzidos, onde pode ser aplicada a ideia de paralelização descrita neste trabalho com baixo custo e se obter uma alta performance.

A conclusão a que se chega é que se for possível o uso de um computador potente para se fazer o processamento, de fato não há grandes vantagens no uso do *hardware*, principalmente se este computador possuir uma GPU. Porém, sabe-se que este não é muitas vezes o caso. É muito comum se ter aplicações onde os recursos são muito limitados. Assim, pode ser bastante vantajoso o uso de *hardware* e metodologias de paralelização de tarefas. Com as tecnologias de processamento distribuído atualmente em voga, se torna muito interessante a análise de formas de se dividir tarefas entre processadores, buscando um desempenho rápido. Por fim, se mostra necessária também uma análise mais aprofundada de tecnologias de comunicação. Em sistemas *many-core* a velocidade de comunicação de dados se mostrou um ponto chave. Com tecnologias com taxas de transferência mais elevadas, é muito possível que a velocidade de processamento de um sistema *many-core* em *hardware* possa superar computadores bastante potentes.

AGRADECIMENTOS

Agradecemos à Universidade Federal de São Carlos, em especial ao CoPICT – Coordenadoria dos Programas de Iniciação Científica e Tecnológica – e ao CNPq – Centro Nacional de Pesquisa e Desenvolvimento - por ter financiado o projeto de pesquisa que gerou este trabalho através do programa de bolsas PIBIC – Programa Institucional de Bolsas de Iniciação Científica.

REFERÊNCIAS

[1] R. Gonzales and R. Woods, Digital Image Processing, 3rd ed. Pearson, 2009.

[2] M. H. O. Rashid, M. A. Mamun, M. A. Hossain and M. P. Uddin, "Brain Tumor Detection Using Anisotropic Filtering, SVM Classifier and Morphological Operation from MR Images," 2018 International Conference on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2), Rajshahi, 2018, pp. 1-4. doi: 10.1109/IC4ME2.2018.8465613

[3] W. Zhang, X. Zhou and Y. Lin, "Application of Morphological Filter in Pulse Noise Removing of Vibration Signal," 2008 Congress on Image and Signal Processing, Sanya, Hainan, 2008, pp. 132-135. doi: 10.1109/CISP.2008.202

[4] Z. Y. Xie and M. Brady, "Wavelet multiscale representation and morphological filtering for texture segmentation," IEE Colloquium on Morphological and Nonlinear Image Processing Techniques, London, UK, 1993, pp. 2/1-2/8.

[5] K. T. Talele, S. T. Gandhe and M. M. Shah, "Fault Detection In PCB Using Homotopic Morphological Operator," 2006 Annual IEEE India Conference, New Delhi, 2006, pp. 1-5. doi: 10.1109/INDCON.2006.302795

[6] Anjanappa C. and Sheshadri.H.S, "Development of mathematical morphology filter for medical image impulse noise removal," 2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT), Mandya, 2015, pp. 311-318. doi: 10.1109/ERECT.2015.7499033

[7] X. Bai and F. Zhou, "New Alternating Sequential Filters and the application for impulsive noise removal," 2010 3rd International Congress on Image and Signal Processing, Yantai, 2010, pp. 1088-1091. doi: 10.1109/CISP.2010.5646889

[8] L. Dou, D. Xu, Y. Liu and H. Chen, "Application of adaptive generalized mathematical morphology theory in de-noising of ECG signals," 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), Guangzhou, 2017, pp. 870-874. doi: 10.1109/ICCSN.2017.8230236

[9] I. Yoda, K. Yamamoto and H. Yamada, "Automatic acquisition of hierarchical mathematical morphology procedures by genetic algorithms", Image and Vision Computing, vol. 17, no. 10, pp. 749-760, 1999. Available: 10.1016/s0262-8856(98)00151-6 [Accessed 4 October 2019].

[10] M. Quintana, R. Poli and E. Claridge, "Morphological algorithm design for binary images using genetic programming", Genetic Programming and Evolvable Machines, vol. 7, no. 1, pp. 81-102, 2006. Available: 10.1007/s10710-006-7012-3 [Accessed 4 October 2019].

[11] B. Albertini, "A vez do FPGA", http://www.ieee.org.br, 2015. [Online]. Available: http://www.ieee.org.br/wp-content/uploads/2014/05/Ed110_EspacoIEEE.pdf. [Accessed: 04- Oct- 2019].

[1]Xilinx, “PetaLinux”,2019. URL: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[13] A. Mishra, M. Agarwal and K. S. Raju, "Hardware and software performance of image processing applications on reconfigurable systems," 2015 Annual IEEE India Conference (INDICON), New Delhi, 2015, pp. 1-5. doi: 10.1109/INDICON.2015.7443611

[14] J. Mazza, D. Patru, E. Saber, G. Roylance and B. Larson, "A comparison of hardware/software techniques in the speedup of color image processing algorithms," 2014 IEEE Western New York Image and Signal Processing Workshop (WNYISPW), Rochester, NY, 2014, pp. 27-31.doi: 10.1109/WNYIPW.2014.6999480

[15] C. Bonney, P. Campos, N. Dahir and G. Tempesti, "Fault tolerant task mapping on many-core arrays," 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8. doi: 10.1109/SSCI.2016.7850174

[16] M. Ruaro, L. L. Caimi, V. Fochi and F. G. Moraes, "A Framework for Heterogeneous Many-core SoCs Generation," 2019 IEEE 10th Latin American Symposium on Circuits & Systems (LASCAS), Armenia, Colombia, 2019, pp. 89-92. doi: 10.1109/LASCAS.2019.8667590

[17] Shao Zuozhi, Zhang Yingqiang, Mu Hongtao and Cheng Rui, "The research on the CPU intelligent scheduling based on the many-core processors," 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, 2016, pp. 779-782. doi: 10.1109/ICSESS.2016.7883183

[18] V. Venkataramani, A. Pathania, M. Shafique, T. Mitra and J. Henkel, "Scalable Dynamic Task Scheduling on Adaptive Many-Core," 2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Hanoi, 2018, pp. 168-175. doi: 10.1109/MCSoC2018.2018.00037

[19] Y. Maruyama, S. Kato and T. Azumi, "Exploring Scalable Data Allocation and Parallel Computing on NoC-Based Embedded Many Cores," 2017 IEEE International Conference on Computer Design (ICCD), Boston, MA, 2017, pp. 225-228. doi: 10.1109/ICCD.2017.41

[20] Xilinx, “Vivado SDK”, 2019. URL: https://www.xilinx.com/products/design-tools/embedded-software/sdk.html

[21] Xilinx, “Vivado HLS”. 2019 URL: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[22] Xilinx, “ZedBoard Hardware User’s Guide”, 2014. URL: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf

[23] Kirschberger, Daniel & Flatt, Holger & Jasperneite, Juergen. (2014). An Architectural Approach for Reconfigurable Industrial I/O Devices.. 2014 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2014. 10.1109/ReConFig.2014.7032500.

Tiago Bachiega de Almeida Graduando em Engenharia de Computação pela Universidade Federal de São Carlos, ingressante em 2015. Possui experiência nas áreas de desenvolvimento de software, visão computacional e processamento de imagens, desenvolvimento de sistemas embarcados com FPGA e inteligência artificial.

Emerson Carlos Pedrino Possui graduação em Engenharia Elétrica Eletrônica pela Universidade de São Paulo e em Bacharelado em Física Computacional também pela Universidade de São Paulo- EESC (2016) e IFSC (2000) -, Mestrado em Engenharia Elétrica pela Universidade de São Paulo - EESC (2003) - e Doutorado em Engenharia Elétrica pela Universidade de São Paulo - EESC (2008). Também fez Pós doutorado em Engenharia Eletrônica (como Professor Visitante) no Departamento de Engenharia Eletrônica da Universidade de York, Inglaterra, colaborando no desenvolvimento de aplicações em hardware para o projeto: "Continuous on-line adaptation in many-core systems:from graceful degradation to graceful amelioration" com bolsa de pesquisa financiada pela FAPESP.

Marcio Merino Fernandes Possuo formação plena em Ciência da Computação: Universidade de São Paulo (graduação, 1989), Universidade Federal de São Carlos (mestrado, 1993) e The University of Edinburgh, UK (PhD, 1999). Atuou como membro do Comitê Gestor do INCT-SEC, o qual possibilitou diversas oportunidades de coordenação e atuação em atividades envolvendo setores acadêmicos e produtivos. Pró-Reitor de Administração da UFSCar.