Acelerando o Reconhecimento de Pessoas em Vídeos com MPI

Gabriel B. Justo[1], **Adriano Vogel**[1], **Dalvan Griebler**[1,2], **Luiz G. Fernandes**[1]

[1] Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre – RS – Brasil

[2]Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC), Faculdade Três de Maio (SETREM), Três de Maio – RS – Brasil

gabriel.justo@acad.pucrs.br

Resumo. *Diversas aplicações de processamento de vídeo demandam paralelismo para aumentar o desempenho. O objetivo deste trabalho é implementar e testar versões com processamento distribuído em aplicações de reconhecimento facial em vídeos. As implementações foram avaliadas quanto ao seu desempenho. Os resultados mostraram que essas aplicações podem ter uma aceleração significativa em ambientes distribuídos.*

1. Introdução

Aplicações de *stream* processam fluxos contínuos de dados, entre as diversas aplicações que utilizam esse paradigma se destaca o cenário de processamento de vídeo. Neste trabalho será abordado o reconhecimento facial em vídeos, com a aplicação apresentada em [Griebler et al. 2017] que identifica rostos em *feeds* de vídeos. Tais rostos são comparados com um banco de imagens para identificar rostos procurados. Essa aplicação exige poder computacional significativo, por isso, execuções sequenciais tem um desempenho pobre. Uma solução é executá-lo em uma máquina *multi-core* ou em um *cluster*, o que exige modificações no código, com a finalidade de permitir sua execução em paralelo de acordo com a interface de programação paralela usada.

Este trabalho tem como objetivo implementar o suporte a execuções distribuídas para *clusters* em uma aplicação real do cenário de processamento de vídeo. A interface MPI foi utilizada a fim de implementar a versão distribuída com um padrão paralelo de processamento de *stream* e duas estratégias de distribuição de tarefas foram implementadas e validadas. O artigo é dividido em seções, sendo a Seção 2 uma comparação com trabalhos relacionados. A Seção 3 explica como foi implementado a aplicação de reconhecimento facial de forma paralela. A Seção 4 expõe os resultados e demonstra como eles foram obtidos. Ainda, a Seção 5 apresenta a conclusão a partir dos resultados obtidos.

2. Trabalhos Relacionados

Em [Hoffmann et al. 2018] os autores apresentam *Denoiser*, um filtro capaz de detectar e eliminar ruídos em *streaming* de vídeo. A estratégia usada pelos autores para implementação de *Denoiser* foi o paralelismo de *stream*, usando um *pipeline* de três estágios. Para essa implementação foi usada a interface FastFlow. Em comparação, este artigo demonstra a implementação utilizando a interface MPI que permite a execução da aplicação de forma distribuída, otimizando ainda mais o processamento de *stream* de vídeo.

Os autores de [Rodriguez et al. 2004] apresentam uma aplicação de codificação de vídeo MPEG-4 em paralelo Para chegar nos resultados foram consideradas duas abordagens, na primeira o processo 0 envia um grupo de imagens para os demais e na segunda cada processo tinha acesso aos grupos de imagens. A versão escolhida para realização dos testes foi a segunda, que, em comparação com este trabalho, tem diferença na distribuição das tarefas entre os processos. A distribuição das tarefas nesse caso é realizada a partir do processo 0 e as imagens são reagrupadas no último processo.

3. Implementações MPI para Processamento Distribuído

A Figura 1 apresenta o funcionamento da aplicação *Person Recognition* [Griebler et al. 2017]. Primeiramente é recebido uma sequência de imagens e posteriormente é gerado um conjunto de faces. A operação *Recognizer* usa um conjunto de imagens de um *Training Set* para comparar com as faces encontradas e verificar se são compatíveis. Ao final, as imagens processadas são concatenadas para um vídeo de saída.

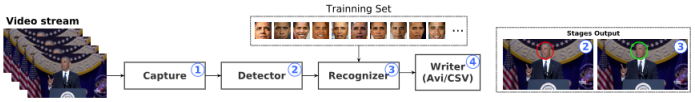

Figura 1. Aplicação *Person Recognition*. Extraída de [Griebler et al. 2017].

Como estratégia para paralelizar a aplicação foi escolhido o padrão *Farm* representado na Figura 2. O Emissor, representado com "E", distribui as imagens para o próximo estágio. A execução das operações *Detector* e *Recognize* acontece nos Trabalhadores representados com "W". Por fim, o Coletor representado com "C" na Figura 2 é responsável por ordenar e escrever o vídeo de saída. O estágio do meio (W) possui réplicas, pois é a etapa mais intensiva da aplicação. Cada réplica representa um processo, que computa simultaneamente diferentes dados para acelerar a execução da aplicação.

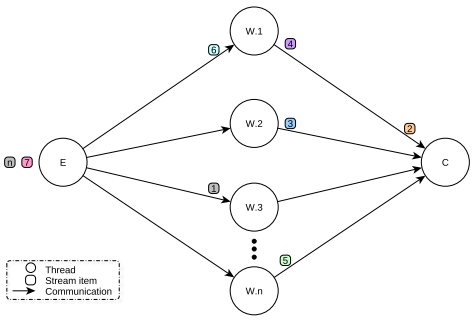

Figura 2. Exemplo do Padrão Paralelo Farm.

Na implementação *Round-Robin* o Emissor distribuí as tarefas de forma estática, em ordem crescente, usando como base o identificador do processo para o qual será enviado a imagem. Para enviar uma imagem foi necessário dividi-la em duas partes. Uma com os dados da imagem e outra com uma *struct* que armazena o número de linhas, colunas e um contador que indica a posição do *frame* no vídeo. O Emissor precisa enviar três mensagens para os Trabalhadores, a primeira com o tamanho dos dados da imagem,

a segunda com os dados da imagem e a terceira com a *struct* que contém as informações para montá-la.

No Trabalhador são recebidas as três mensagens, as quais contém informações para que possa ser montada a imagem novamente.Quando o processamento da imagem é finalizado, ela é novamente separada em duas partes e enviada para o Coletor.O Coletor recebe as mensagens com as informações necessárias para realizar a montagem da imagem. Na *struct* com as informações da imagem há um número que indica a posição dela no vídeo e que é usado em um algoritmo de ordenamento para que as imagens sejam concatenadas em ordem. Essa lógica do ordenamento foi baseada em [Griebler et al. 2018].

Outra estratégia usada é a distribuição dinâmica de tarefas. Nessa implementação, a estratégia de paralelismo, operações de processamento, ordenamento e envio de tarefas é similar a implementação *round-robin*. Porém, a distribuição das tarefas entre o Emissor e Trabalhadores é diferente. Quando o Trabalhador estiver disponível para processar tarefas, este envia ao Emissor uma mensagem, com seu ID, requisitando tarefas. O Emissor por sua vez, recebe mensagens de Trabalhadores disponíveis, coleta o ID do processo e envia as mensagens necessárias para cada tarefa ao respectivo processo.

4. Análise Comparativa

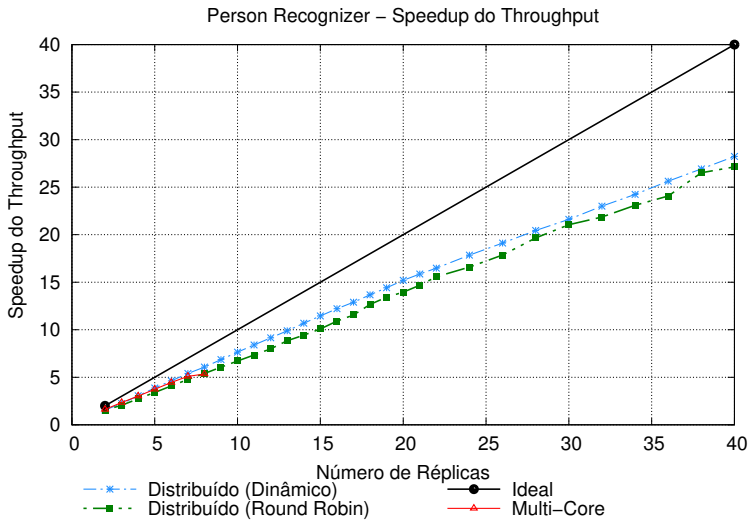

Figura 3. *Speedup* do *Throughput* das implementações.

Os testes foram executados em um *cluster* composto por máquinas equipadas com processador Intel(R) Xeon(R) CPU 2.40GHz (8 cores-16 threads), com memória de 16 GB - DDR3 1066 MHz. As duas implementações foram testadas com diferentes números de processos, que variaram de 4 a 42, na Figura 3, o número de réplicas está representado de 2 a 40, pois cada réplica representa um Trabalhador. O Emissor e o Coletor são implementados de forma sequencial. Cada execução contava com um nodo separado para o Emissor e um para o Coletor. Os processos Trabalhadores, que correspondem ao número de réplicas, foram divididos entre 5 nodos. Para cada número de processos foram realizadas 5 repetições, sendo o número possível considerando o grande número de execuções no tempo disponível no *cluster*. A cada execução foram armazenados os dados do tempo de

execução e *throughput*, para posteriormente serem analisados e comparados. Através dos dados obtidos com a execução da aplicação sequencial foi calculado também o *speedup* do *throughput* da aplicação .

As implementações apresentadas neste trabalho foram comparadas com a implementação em memória compartilhada usando a interface FastFlow (*multi-core*) demonstrada em [Griebler et al. 2017]. As duas versões apresentam resultados semelhantes com até 7 processos, representado na Figura 3, onde a versão *multi-core* atinge o limite de escalabilidade em um único nodo, como visto em [Griebler et al. 2017], onde a versão*multi-core* foi testada em uma máquina com mais cores, e o limite de escalabilidade foi alcançado com 12 réplicas. Entre as versões distribuídas, a distribuição *Round-Robin* de tarefas tem a vantagem de necessitar um menor número de mensagens para distribuir as imagens entre os Trabalhadores. No entanto, os melhores resultados foram obtidos na distribuição dinâmica de tarefas, pois há uma diferença de tempo entre o processamento das imagens, beneficiando um modelo dinâmico que possui um melhor balanceamento de carga.

5. Conclusão

Este trabalho apresentou duas implementações distribuídas da aplicação Person Recognition usando a interface MPI. Os resultados foram comparados com a versão implementada em FastFlow (*multi-core*). A versão distribuída atingiu um desempenho superior com 8 ou mais processos, devido à possibilidade de executá-la usando mais de uma máquina. Dentre as implementações com MPI, a versão com distribuição dinâmica de tarefas teve uma escalabilidade melhor comparado com a versão com distribuição *Round-Robin* de tarefas. Em trabalhos futuros se pretende avaliar a média de tempo de comunicação e comparar com a média de tempo de computação.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal Nível Superior – Brasil (CAPES) – Código de Financiamento 001 e FAPERGS.

Referências

[Griebler et al. 2017] Griebler, D., Hoffmann, R. B., Danelutto, M., and Fernandes, L. G. (2017). Higher-Level Parallelism Abstractions for Video Applications with SPar. In *Parallel Computing is Everywhere, Proceedings of the International Conference on Parallel Computing*, ParCo'17, pages 698–707, Bologna, Italy. IOS Press.

[Griebler et al. 2018] Griebler, D., Hoffmann, R. B., Danelutto, M., and Fernandes, L. G. (2018). Stream Parallelism with Ordered Data Constraints on Multi-Core Systems. *Journal of Supercomputing*, 75:1–20.

[Hoffmann et al. 2018] Hoffmann, R. B., Griebler, D., and Fernandes, L. G. (2018). Paralelização de uma Aplicação de Detecção e Eliminação de Ruídos em Streaming de Vídeo com a DSL SPar. In *Escola Regional de Alto Desempenho (ERAD)*, page 2, Porto Alegre, BR. Sociedade Brasileira de Computação (SBC).

[Rodriguez et al. 2004] Rodriguez, A., Gonzalez, A., and Malumbres, M. P. (2004). Performance Evaluation of Parallel MPEG-4 Video Coding Algorithms on Clusters of Workstations. In *Parallel Computing in Electrical Engineering, 2004. International Conference on*, pages 354–357. IEEE.