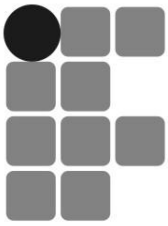

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA FLUMINENSE
Campus Campos-Centro

Secretaria de Educação Profissional e Tecnológica

Ministério da Educação

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO/GERÊNCIA DE PESQUISA

JOÃO FELIPE ROQUE MORAES

SEGMENTAÇÃO E PARALELIZAÇÃO DE VÍDEOS EM SISTEMAS MULTIPROCESSADOS

Relatório parcial de Pesquisa de Iniciação Científica do Instituto Federal Fluminese Campi Campos Centro apresentado à Diretoria de Pesquisa e Pós-Graduação.
Orientador: Fábio Duncan de Souza

CAMPOS DOS GOYTACAZES - RJ
2010

SUMÁRIO

1 INTRODUÇÃO

As aplicações computacionais, tanto as científicas quanto as comerciais, tendem a tornar-se maiores, quer no volume de dados manipulados quer na complexidade do processamento que lhes está associado. Nesse contexto, a programação paralela juntamente com o conceito de sistemas distribuídos surge como uma boa alternativa para melhorar o desempenho de execução das aplicações.

(TANENBAUM; STEEN, 2002) definem sistemas distribuídos como uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente. Dessa forma, os recursos do sistema como hardware, software e dados são compartilhados.

Uma tecnologia atual que incorpora o conceito de compartilhamento de *hardware* são os processadores multi-*core*, onde em um único processador existem vários núcleos operacionais. Para *desktop* são encontrados no mercado processadores contendo de 2 a 4 núcleos Intel (), e ainda em previsão de lançamento há outros que conterão 8, 12, 16, e até 32 unidades de trabalho.

Dispondo de um processador Intel *Core* 2 *Duo* e utilizando uma técnica de paralelização de aplicações chamada paralelismo de dados, este projeto visa paralelizar e otimizar o código de uma aplicação que realiza a segmentação automática em tomadas de vídeo, desenvolvida pelo Núcleo de Pesquisa em Sistemas de Informação do Instituto Federal Fluminense (NSI/IFF).

2 OBJETIVOS

Investigar na literatura ferramentas que viabilizem a paralelização de uma aplicação de segmentação de vídeos, para posteriormente, através da comparação de resultados obtidos em testes, escolher a que obtiver melhor desempenho.

3 CONHECIMENTOS TEÓRICOS

A seguir serão apresentadas informações sobre o funcionamento da aplicação de segmentação de vídeos, conceitos que integram o processamento paralelo e as tecnologias utilizadas na implementação da técnica do paralelismo de dados no algoritmo da aplicação.

3.1 Aplicação de segmentação de vídeos

Arquivos digitais multimídia são volumosos, se caracterizam por possuir grande quantidade de informação e necessitam, portanto, de novas tecnologias para indexação e recuperação de seus conteúdos. A busca manual de um conteúdo em um arquivo de vídeo que se deseja recuperar, implica em encontrar o arquivo certo, assistir a trechos do vídeo em busca da parte que se tem interesse e fazer o download na íntegra do arquivo.

Sendo os arquivos de vídeo disponibilizados desta forma, o número de buscas sem sucesso por um conteúdo poderá ser alto. Assim, o usuário poderá se desmotivar dada a dificuldade em se buscar um conteúdo, além da existência de um consumo desnecessário de parte da banda da rede tanto do lado do cliente quanto do lado do servidor para se recuperar o arquivo na íntegra.

Uma aplicação de segmentação de vídeo visa encontrar componentes mais básicos da estrutura de um vídeo, permitindo que os mesmos sejam disponibilizados independentemente, viabilizando a busca e recuperação da informação de forma mais eficiente. Para isto, as imagens contidas em um vídeo são analisadas e comparadas com sua antecessora e sucessora. Se a diferença de contexto entre as imagens comparadas for maior que um limiar preestabelecido, tem-se o início de um novo segmento. A imagem inicial do novo segmento detectado representa uma transição, e a cada nova transição detectada, a posição desta é armazenada. Dessa forma, ao final do processo de comparação entre as imagens do vídeo, tem-se o tempo inicial de todos os seguimentos detectados.

O último passo é realizar os cortes no vídeo de acordo com as faixas de tempo armazenadas e disponibilizar assim cada seguimento independente um dos outros, podendo o usuário recuperar somente os trechos de seu interesse.

Um exemplo de imagens representando transições detectadas em um determinado vídeo pode ser visto na Figura 3.1 abaixo.

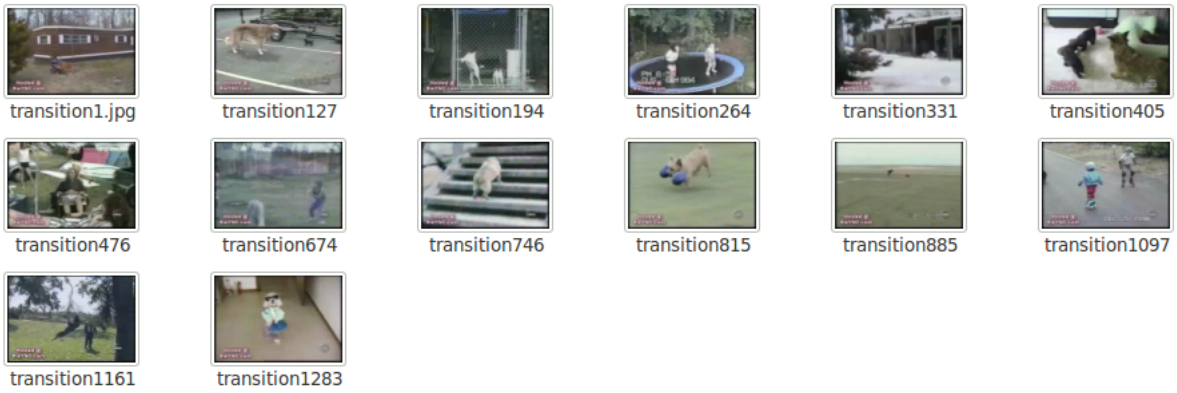

Figura 3.1: Imagens de um determinado vídeo representando as transições

3.2 Processamento Paralelo

3.2.1 Processos

Em sistemas multitarefa como o Linux, múltiplas tarefas podem ser realizadas simultaneamente, como por exemplo a execução de uma música, a impressão de um arquivo e a edição de um texto. Cada uma delas corresponde a um programa que a executa. Um programa em execução é basicamente um processo (TANENBAUM; WOODHULL, 2002).

Cada processo contém suas próprias informações de estado, utiliza endereço de espaços próprios, e só interagem uns com os outros através de mecanismos de comunicação entre processos.

Criar processos é uma das funções do sistema operacional. Comumente são encontradas aplicações que, independente do seu tamanho ou complexidade, são executadas por apenas um processo, no entanto, é possível dividir o algoritmo de uma aplicação em partes e criar processos para recebê-las.

3.2.2 *Threads*

Em um processo tradicional, do tipo que definimos acima, há uma única linha de controle. Entretanto, em alguns sistemas operacionais modernos, é fornecido suporte para múltiplas linhas de controle dentro de um processo. Estas linhas de controle normalmente são chamadas *threads* (TANENBAUM; WOODHULL, 2002). As *Threads* foram introduzidas na tentativa de reduzir o tempo gasto na criação, eliminação e troca de contexto de processos nas aplicações paralelas, bem como economizar recursos do sistema como um todo.

Em um ambiente *multithread*, cada processo pode suportar múltiplos *threads*, cada qual associado a uma parte do código da aplicação. Um *browser* é um exemplo de aplicação *multithreading*, pois várias ações podem ocorrer ao mesmo tempo, como por exemplo *download* de um *applet*, *download* de uma imagem, tocar um som e etc.

A diferença entre processo e *thread* é que a área de dados de um processo é própria, enquanto a de uma *thread* não. Isso significa que cada processo tem a sua própria área de dados, em geral inacessível aos outros, enquanto as *threads* compartilham uma mesma área de dados.

3.3 Linguagem *Python*

Python é uma linguagem de programação de alto nível e eficiente. Segundo (BORGES, 2010), esta possui uma sintaxe clara e concisa, que favorece a legibilidade do código fonte, tornando a linguagem mais produtiva.

Para o *python*, módulos são arquivos fonte que podem ser importados para um programa. Podem conter qualquer estrutura do *python* e podem ser executados quando importados.

Para trabalhar com o processamento paralelo foram avaliados três módulos compatíveis com a linguagem *python*: *parallel python*, *threading* e *multiprocessing*.

3.3.1 Módulo *parallel pyhon*

Como Vanovschi (2005) define, *parallel python* é um módulo em *python* que oferece mecanismos para execução paralela em sistemas com múltiplos processadores ou núcleos.

Suas principais características são: facilidade de implementação, possui código fonte aberto, portabilidade e interoperabilidade com diferentes sistemas operacionais, como *Linux*, *Wndows* e *Unix*.

3.3.2 Módulo *multiprocessing*

Segundo a Python Software Foundation (a), *multiprocessing* é um módulo que permite que desenvolvedores executem vários processos em uma determinada máquina. Este é compatível com as plataformas *Unix* e *Windows* e já vem integrado nas versões superiores a 2.5 do *Python*.

3.3.3 Módulo *threading*

O módulo *Thread* em *python* oferece métodos de baixo nível para trabalhar com múltilplas *threads*. Deste módulo deriva-se outro chamado *Threading*, que oferece métodos em alto nível, facilitando seu uso. Este é nativo do *python* versão 2.5 ou superior e é compatível com *Windows*, *Linux*, *SGI IRIX*, *Solaris* 2.x, bem como em sistemas que possuem um segmento *POSIX* (*aka "pthread"*) de execução Python Software Foundation (b).

4 METODOLOGIA

A aplicação de segmentação de vídeos por realizar a análise e comparação de muitas imagens requer um alto poder de processamento (SMITH, 1993). Processadores multi-*core* surgem como boa alternativa para aumentar o poder computacional, mas mesmo sendo executada por tais tecnologias, a aplicação não utiliza o potencial total dos núcleos existentes, pois seu algortimo foi desenvolivdo para ser executado sequencialmente.

Utilizando a liguagem de programação *python* juntamente com três módulos de multiprocessamento, foi proposta uma estratégia de paralelização do algoritmo de segmentação de vídeo pelo Núcleo de Pesquisa em Sistemas de Informação do Instituto Federal Fluminense (NSI/IFF). A definição dos módulos a serem utilizados foi baseada em pesquisas realizadas na literatura, e de acordo com as informações adquiridas foram testados os módulos *parallel python*, *multiprocessing* e *threading*.

A seguir serão apresentadas informações sobre a técnica de paralelismo de dados, as implementações realizadas por cada um dos módulos e o funcionamento da aplicação de segmentação de vídeos após ter seu algoritmo paralelizado.

4.1 Paralelismo de dados

Segundo (FERNANDES, 2000), paralelismo de dados é uma técnica usada normalmente em tarefas grandes e complexas para obter resultados mais rápidos, dividindo-as em tarefas pequenas que serão distribuídas em vários processadores para serem executadas simultaneamente.

A Figura 4.1 apresenta um cenário onde uma grande quantidade de dados é processada por apenas um processador, enquanto os outros três permancem ociosos.

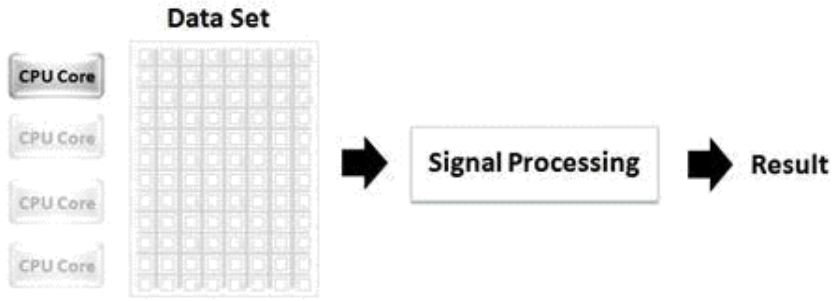

Figura 4.1: Em métodos tradicionais de programação, uma grande quantidade de dados é processada em um único núcleo da *CPU*, enquanto os outros núcleos permanecem ociosos. National Instruments ()

Na Figura 4.2 o cenário apresentado é diferente. É utilizado o paralelismo de dados para aproveitar completamente a força de processamento oferecida por um processador quad-core (quatro núcleos de processamento). Neste caso, a grande quantidade de dados é dividida em quatro sub-conjuntos. Cada sub-conjunto é associado a um núcleo individual para processamento. Após o processamento ser finalizado, estes sub-conjuntos são reagrupados em um único e completo conjunto de dados.

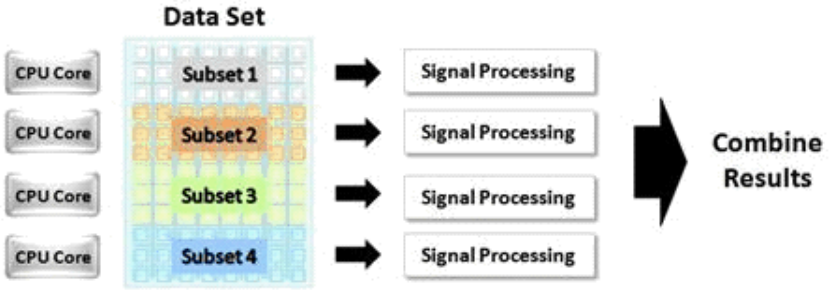

Figura 4.2: Utilizando a técnica de programação com paralelismo de dados, uma grande quantidade de dados pode ser processada em paralelo em múltiplos núcleos da *CPU*. National Instruments ()

Este tipo de estratégia de execução é conhecida como paralelismo de dados e consiste na execução de uma mesma operação sobre conjuntos distintos de dados.

4.2 Implementação paralela

Visando paralelizar o algoritmo sequencial da aplicação de segmentação de vídeos, a estratégia criada terá como objetivo implementar o algoritmo da mesma com a técnica do paralelimo de dados. Para viabilizar tal implementação serão testados os módulos *parallel python*, *multiprocessing* e *threading*. É importante ressaltar que as alterações a serem realizadas no algoritmo da aplicação serão as mesmas para os três módulos, porém tais alterações serão feitas de formas distintas pelo fato de cada módulo utilizar as suas próprias funcionalidades.

Os módulos *parallel python* e *multiprocessing* por possuírem funcionalidades objetivas, como por exemplo extrair o número de núcleos existentes no processador do sistema, permitiram implementar a aplicação de forma eficiente, pois o cálculo da divisão dos dados entre os processadores depende da quantidade de *cores*. Uma relevante diferença entre as funcionalidades dos módulos está entre as classes responsáveis pela criação de processos. Enquanto a classe do módulo *parallel python* requer diversos argumentos, a do *multiprocessing* é mais simples e direta.

O módulo *threading*, diferentemente dos módulos anteriores, é baseado na técnica de utilização de *threads*. A classe responsável pela criação das *threads* é quase idêntica à classe de criação de processos do módulo *multiprocessing*. Os argumentos são recebidos da mesma maneira, sendo as classes diferenciadas apenas pelos seus nomes. Um fator negativo é a inexistência da funcionalidade para extração do número de *cpus* do sistema, o que tornará necessário recorrer à mesma funcionalidade do módulo *multiprocessing*.

Após a paralelização do algorimto da aplicação de segmentação de vídeos, o volume de dados poderá ser dividido entre os processadores disponíveis no sistema computacional. Processos ou *threads* serão criados de acordo com o módulo utilizado para implementar a aplicação com a técnica de paralelismo de dados. Tanto a divisão dos dados quanto a criação de processos serão realizados de acordo com o número de processadores existentes no sistema. Os dados a serem processados serão enviados aos seus respectivos processos, que serão executados paralelamente cada um por um núcleo. Por exemplo: a aplicação ao trabalhar sobre um vídeo de 1.000 *frames* e sendo executada por um processador contendo dois núcleos, irá criar dois processos e enviará para cada um, um bloco de 500 *frames*, sendo um bloco referente à primeira metade do vídeo e o outro à segunda.

5 COMPARAÇÃO DOS RESULTADOS

O passo seguinte à paralelização do algoritmo da aplicação de segmentação de vídeos, foi a criação de uma estratégia de teste com o objetivo de obter os resultados de cada uma das aplicações, visando a busca pelo melhor desempenho.

O ambiente computacional que será utilizado para realizar as execuções dos testes é composto pelo sistema operacional *Linux Ubuntu* versão 10.04 LTS, processador Intel *Core* 2 *Duo* P800 2.4 GHz e memória RAM de 3 GB.

A seguir serão apresentadas informações sobre a forma como os testes foram realizados, a utilização dos *cores* do processador no momento da execução de cada uma das aplicações e a comparação dos resultados obtidos.

5.1 O teste

O teste se baseia num conjunto de cinco vídeos, com conteúdos e tamanhos distintos. O objetivo do teste é executar com cada umas das aplicações de segmentação de vídeo, tanto a sequencial quanto as paralelas, todos os cinco vídeos em razão de obter o tempo de processamento gasto em cada um. Uma vez tendo estes resultados, eles serão comparados a fim de descobrir a aplicação de melhor desempenho.

Vídeos	Nº de *frames*	Tempo de duração (s)
Vídeo 1	1947	78
Vídeo 2	1978	79
Vídeo 3	2509	100
Vídeo 4	2643	88
Vídeo 5	3458	115

Figura 5.1: Informações sobre os vídeos incluídos no teste

A Figura 5.1 apresenta a quantidade de *frames* ou imagens existentes em cada vídeo, e a sua duração em segundos.

Durante a execução dos testes pelas aplicações foram encontrados dois problemas. O primeiro é que devido a uma incompatibilidade com um tipo de objeto da aplicação de segmentação de vídeos, a aplicação implementada pelo módulo *parallel python* apresentou um erro que im-

possibilitou a recuperação dos seus resultados. O segundo problema é referente à aplicaçao implementada pelo módulo *threading*. Esta não apresentou erro durante sua execução, porém os resultados recuperados foram incompatíveis com os que eram esperados.

Em razão dos problemas ocorridos, irão ser apresentados apenas os resultados recuperados da aplicação sequencial e da aplicação implementada pelo módulo *multiprocessing*.

5.2 Uso dos *cores*

O sistema operacional Linux Ubuntu versão 10.04 LTS disponibiliza um aplicativo chamado monitor do sistema, que apresenta informações e gerencia diversos recursos do sistema. Um destes recursos é o histórico da *cpu*, que informa em tempo real o percentual de utilização dos processadores enquanto determinada tarefa é executada.

Dessa forma, durante a execução dos testes pela aplicação de segmentação de vídeo sequencial e pela aplicação paralela implementada pelo módulo *multiprocessing*, foi possível observar a utilização dos *cores* do processador e constatar a diferença do uso das *cpus* entre a aplicação sequencial e paralela.

- Aplicação sequencial

Figura 5.2: Histórico da *cpu* da aplicação sequencial

- Aplicação paralela implementada pelo módulo *multiprocessing*

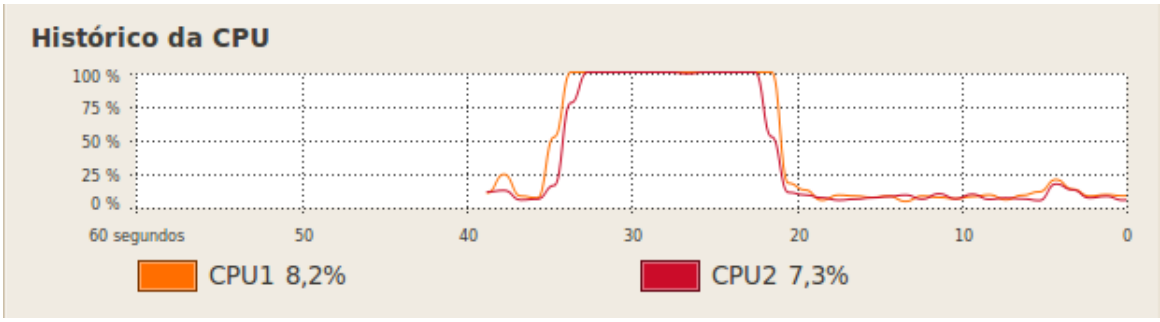

Figura 5.3: Histórico da *cpu* da aplicação implementada pelo módulo *multiprocessing*

A Figura 5.2 apresenta a utilização dos núcleos do processador pela aplicação sequencial. Enquanto um núcleo utiliza todo o seu potencial, o outro oscila na faixa de 25 a 50%.

A Figura 5.4 apresenta o percentual de utilização dos *cores* feita pela aplicação implementada pelo módulo *multiprocessing*. Através desta figura obeserva-se que potencial total das duas unidades de trabalho foi alcançado.

Dessa forma, conclui-se que a aplicação paralela implementada pelo módulo *multiprocessing* realizou um melhor aproveitamento dos núcleos do processador do que a aplicação sequencial.

5.3 Performance das implementações

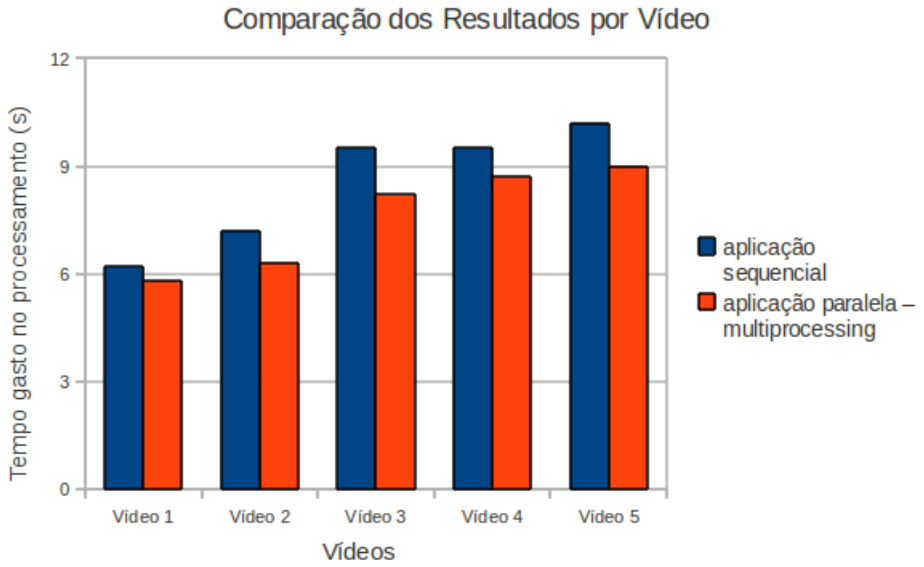

Figura 5.4: Comparação dos resultados por vídeo

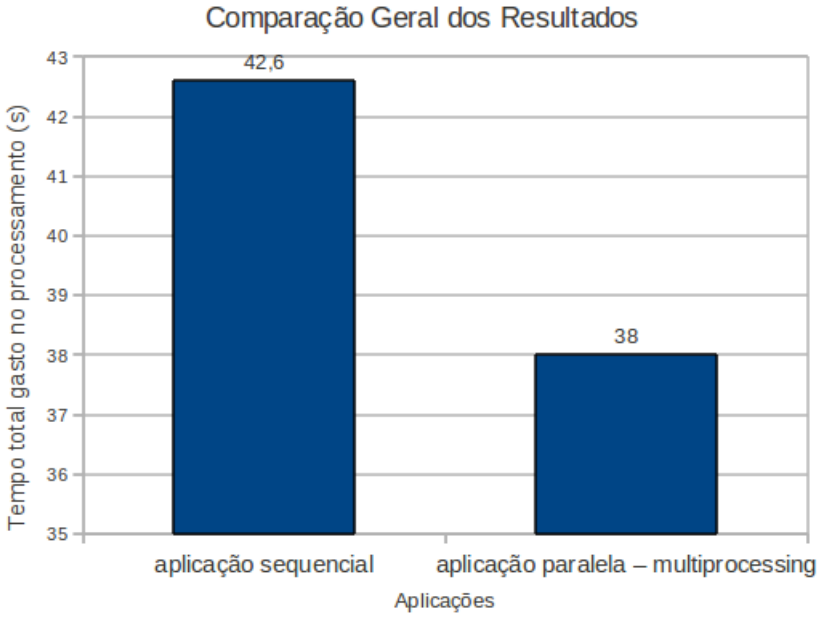

Figura 5.5: Comparação geral dos resultados

Os gráficos acima apresentam os desempenhos da aplicação de segmentação de vídeo sequencial e e da aplicação paralela implementada pelo módulo *multiprocessing* em relação ao tempo de processamento gasto na execução dos vídeos.

Através do gráfico da Figura 5.5, nota-se que a aplicação paralela implementada pelo módulo *multiprocessing* reduziu o tempo de processamento em todos os vídeos quando comparado com os tempos da aplicação sequencial. Este aumento de desempenho é notado melhor

através do gráfico da Figura 5.6, onde é apresentado o tempo de processamento total gasto por cada aplicação. A diferença entre os desempenhos é de 4,6 segundos, ou seja, a aplicação paralela reduziu em aproximadamente 11% o tempo de processamento total gasto pela aplicação sequencial.

6 CONCLUSÃO

Após a comparação dos resultados, conclui-se que a paralelização do processamento foi uma boa alternativa para a aplicação de segmentação de vídeos, pois o objetivo principal de reduzir seu tempo de processamento foi alcançado.

No entanto, a vantagem adquirida pela aplicação paralela implementada pelo módulo *multiprocessing* em relação ao desempenho da aplicação sequencial foi de apenas 4,6 segundos. Devido ao funcionamento incorreto do comando específico que realiza o posicionamento do ponteiro do vídeo no *frame* exato onde cada processo inicia seu trabalho, este procedimento está sendo realizado de uma forma mais demorada, ocasionando uma perda de tempo.

Ainda assim, a aplicação paralela implementada pelo módulo *multiprocessing* é a dona do melhor desempenho, e em razão disso é a escolha atual para ser utilizada pelo projeto de Segmentação de Vídeos.

7 PERSPECTIVAS FUTURAS

Investigar novas estratégias de paralelização para a aplicação de segmentação de vídeo utilizando o módulo *multiprocessing* e avaliar o motivo pelo qual a diferença de desempenho foi aquém do esperado, utilizando para testes vídeos de diferentes durações e pesquisando uma forma mais eficiente de posicionar o ponteiro de vídeo em um determinado *frame* do mesmo.

REFERÊNCIAS BIBLIOGRÁFICAS

BORGES, L. E. *Python para Desenvolvedores - 2ª edição*. Rio de Janeiro: Edição do autor, 2010.

FERNANDES, S. de F. *Processamento Paralelo - Introdução ao MPI*. São Paulo: [s.n.], 2000.

INTEL. *Processadores para desktop*. Disponível em http://www.intel.com/portugues/products/desktop/processors/index.htm, acesso em 08/04/2011.

NATIONAL INSTRUMENTS. *Estratégias de Programação para Processamento Multicore: Paralelismo de Dados*. Disponível em http://digital.ni.com/worldwide/brazil.nsf/web/all/B8B5B0B0E278DFAA862573B400706C85, acesso em 27/01/2011.

PYTHON SOFTWARE FOUNDATION. *multiprocessing - Process-based "threading" interface*. Disponível em http://docs.python.org/library/multiprocessing.html, acesso em 21/01/2011.

PYTHON SOFTWARE FOUNDATION. *threading - Higher-level threading interface*. Disponível em http://docs.python.org/library/threading.html#module-threading, acesso em 22/01/2011.

SMITH, J. R. *The Design and Analysis of Parallel Algorithms*. USA: Oxford University Press, 1993.

TANENBAUM, A. S.; STEEN, M. van. *Distributed Systems: Principles and Paradigms*. [S.l.]: Us ed edition, 2002.

TANENBAUM, A. S.; WOODHULL, A. S. *Sistemas Operacionais - Projeto em Implementação - 2ª Edição*. Porto Alegre: Bookman, 2002.

VANOVSCHI, V. *Parallel Python*. 2005. Disponível em http://www.parallelpython.com/, acesso em 18/01/2011.