Implementação e avaliação de desempenho, usando vídeos H.264 High Profile, do Componente Media Processing para o Middleware Ginga

Tiago H. Trojahn, Lisane Brisolara, Eliane S. A. Diniz, Luciano V. Agostini, Juliano L. Gonçalves, Leomar S. da Rosa Junior,

Universidade Federal de Pelotas
Grupo de Arquitetura e Circuitos Integrados
Campus Universitário s/n - Pelotas - RS - Brasil
Cx.p.: 354 Cep: 96010-900
Fone: +55 53 3275 7432
{tiagot.ifm, lisane, eliane, agostini, juliano.lucas, leomarjr}@ufpel.edu.br

RESUMO

No sistema brasileiro de TV digital (SBTVD) são definidos dois ambientes de execução: um procedural (GingaJ que usa a linguagem de programação procedural Java) e outro declarativo (GingaNCL baseado na linguagem declarativa Nested Context Language), executados simultaneamente no mesmo *middleware*. Tais ambientes compõem o Ginga, o *middleware* do SBTVD. Atualmente, não há um *middleware* capaz de prover suporte a ambos os ambientes GingaJ e GingaNCL e, por essa razão, foi criado o projeto Ginga *Code Development Network* (GingaCDN) cujo objetivo é desenvolver uma implementação completamente integrada e modular do *middleware* Ginga. Para realizar essa integração entre os dois ambientes, está sendo desenvolvido um núcleo comum, chamado de Ginga *Common Core* (GingaCC). Um dos principais componentes do GingaCC é o Media Processing, responsável por decodificar fluxos de vídeo, áudio e de legendas. O objetivo deste trabalho é apresentar duas implementações para o componente Media Processing. Na primeira implementação é usada a biblioteca gráfica libVLC e, na segunda, a biblioteca Xine. Também são realizados testes de desempenho, em dois microcomputadores com configurações diferentes, para avaliar o funcionamento do componente na decodificação de fluxos de mídias.

Categorias e Descrição de Assuntos

D.3.3 [**Linguagens de Programação**]: Linguagem C++.

Termos Gerais

Desempenho, Linguagens, Experimentos.

Palavras-chave

Ginga, *middleware*, TV Digital, Media Processing.

1. INTRODUÇÃO

No Sistema Brasileiro de TV Digital (SBTVD) está definido a utilização de dois ambientes, o declarativo chamado Ginga Nested Context Language (GingaNCL) baseado no *middleware* proposto por Moreno [8], e o procedural, o Ginga-J [2]. Atualmente, estes ambientes não são suportados simultaneamente por um mesmo middleware, resultando na impossibilidade de se criar aplicativos híbridos contendo código NCL e Java.

Isso se torna um limitador para os desenvolvedores, porque além de se preocuparem com o tipo de aplicação a ser desenvolvida (declarativa ou procedural), também se torna necessário instalar o ambiente de execução que dê suporte a mesma, acarretando um esforço adicional, o qual poderia ser evitado com a existência de um único sistema com suporte tanto a aplicações declarativas quanto procedurais.

Em virtude dessas dificuldades , em fevereiro de 2009, foi criado o projeto Ginga *Code Development Network* (GingaCDN) [3]. O objetivo do GingaCDN é produzir um núcleo comum chamado de Ginga *Common Core* (GingaCC), que dê suporte aos sistemas GingaNCL e GingaJ. Os componentes estão sendo desenvolvidos na linguagem C++ e usando um modelo de componente chamado de FlexCM [11] para facilitar a integração entre todos os componentes do núcleo comum.

O foco deste trabalho é o componente chamado de Media Processing, um dos principais módulos do GingaCC. Para explorar diferentes soluções, foram desenvolvidos dois componentes com as mesmas funcionalidades. A primeira solução é baseada na biblioteca libVLC [7], enquanto o segundo é baseada na biblioteca Xine [13]. Embora a decodificação de mídias já seja implementada por outros *middlewares* do SBTVD, como apresentado em [2], [6] e [8], nenhum deles provê suporte a ambos os ambientes simultaneamente, GingaNCL e GingaJ, sendo por isso o Media Processing implementado um componente inédito.

Este trabalho apresenta, também, comparações entre ambas as soluções e está estruturado da seguinte forma: A seção 2 fala sobre o componente Media Processing, bibliotecas libVLC e Xine, apresenta um exemplo de implementação de ambas as versões do Media Processing e também descreve o modelo de componentes FlexCM; a seção 3 descreve os experimentos realizados e compara os resultados obtidos; a seção 4 apresenta as conclusões e trabalhos futuros.

2. COMPONENTE MEDIA PROCESSING

O GingaCC é responsável por prover suporte às aplicações desenvolvidas para o GingaNCL e o GingaJ através de um

conjunto de funcionalidades fundamentais, dentre as quais a decodificação do fluxo de vídeo. No GingaCC, o componente responsável por esta função é o Media Processing.

A versão atual do Media Processing possuí um conjunto de métodos capazes de realizar funcionalidades básicas, como a decodificação de vídeo e a exibição de legendas. Uma descrição de alto-nível dessas funcionalidades está descrita abaixo:

- Decodificação de fluxo de vídeo, incluindo, mas não se limitando, ao formato H.264/ *Advanced Video Coding* (AVC) usados pelo SBTVD.
- Métodos para controle básico do fluxo de vídeo, como *play*, *pause* e *stop*.
- Prover um conjunto de informações do fluxo de vídeo, como duração total, tempo de execução atual, resolução, proporção de tela (*aspect ratio*) e taxa de quadros por segundo, conhecido pela sigla FPS (*Frame Rate per Second*).
- Realizar captura de tela e salvá-la em um determinado caminho, com uma miniatura sendo exibida na tela.
- Suporte a vídeos em streaming usando protocolos como o *Hypertext Transfer Protocol* (HTTP) e o *File Transfer Protocol* (FTP).

Dois outros componentes, Demux, responsável por separar os vários fluxos presentes na *Transport Stream* (fluxo de mídia enviada pela emissora) e provê-los ao componente correto para ser processado e o Graphics, componente responsável por exibir vídeo e legendas na tela, interagem diretamente com Media Processing, como descrito abaixo:

- Demux - O Media Processing recebe fluxos de áudio, vídeo e legendas do Demux.
- Graphics - Recebe o fluxo decodificado do Media Processing e o exibe utilizando um driver de saída de vídeo customizado.

As interfaces e as conexões entre os componentes Demux, Media Processing e Graphics do GingaCC são mostrados na Figura 1.

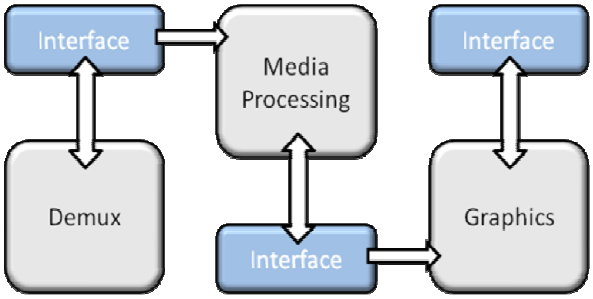

Figura 1. Interfaces e conexões entre os componentes Media Processing, Demux e Graphics.

Para explorar a eficiência do componente Media Processing foram desenvolvidas duas implementações. A primeira foi implementada com a biblioteca libVLC e oferece todas as funcionalidades descritas nesta seção. A segunda implementação utiliza a biblioteca Xine e para fins de teste usa um módulo programado em X.Org para exibir o vídeo decodificado resultante. Esta segunda implementação oferece métodos para abrir, decodificar e executar um fluxo de vídeo de entrada.

2.1 LibVLC

A versão 1.0.6 da biblioteca libVLC foi utilizada para a implementação do Media Processing. LibVLC é uma biblioteca gráfica desenvolvida pela VideoLAN sobre a licença GNU *General Public License* (GPL) versão 2. Esta biblioteca possui como características principais:

- Compatibilidade com vários tipos de mídia, incluindo o padrão H.264/AVC do SBTVD, o padrão de áudio MPEG *Layer* 2, MPEG *Layer* 3 (MP3) e o MPEG-4 *part* 3, também conhecido como *Advanced Audio Coded* (AAC).
- A biblioteca em si foi escrita na linguagem C, oferecendo um alto desempenho necessário para processar mídias. A biblioteca foi portada, também, para outras linguagens, como o Java.
- Possui suporte nativo a operações multithread como, por exemplo, a separação entre o fluxo principal de controle e a exibição de vídeo.

As características apresentadas, sobretudo a compatibilidade com diferentes mídias e suporte a operações *multithread*, justificam a escolha por esta biblioteca.

A organização desta biblioteca, no seu mais alto nível de abstração, pode ser dividida em duas classes complementares, a *libVLC_media_player* e o *libVLC_media*. As principais funcionalidades de ambas as classes estão descritas abaixo:

- *libVLC_media_player*: Provê métodos de controle de reprodução e retorno de informações do fluxo de entrada, além de algumas outras funções, como adicionar legendas ao fluxo de vídeo.
- *libVLC_media*: Provê métodos de baixo nível que permitem controlar a mídia diretamente, como calcular a duração total da mídia e retornar uma série de informações da mesma. Pode ser dividida no *libVLC_video* e no *libVLC_audio*, as classes básicas para controlar vídeo e áudio, respectivamente.

2.2 Xine

A biblioteca Xine, conhecida como Xine-lib, é uma biblioteca do tipo *back-end* que provê demuxagem e decodificação de fluxos de áudio e vídeo. Foi desenvolvida pelo Xine *Project* sob a licença GNU *General Public License* (GPL) versão 2, criada como um *player* de alta eficiência cuja função, originalmente, era a de facilitar a reprodução de DVD em sistemas Unix/Linux. Xine é uma biblioteca poderosa criada para ser simples e independente de arquitetura, deixando os detalhes do *front-end* para outros módulos. As principais funcionalidades da Xine-lib são:

- Suporte nativo a um grande número de formatos de áudio e vídeo, como o padrão de vídeo H.264/AVC e o padrão de áudio AAC, usados pelo SBTVD.
- Portabilidade com todos os sistemas operacionais do tipo “Unix-like” e com o Microsoft Windows através do uso de um *wrapper*.

- A biblioteca foi desenvolvida na linguagem C.

O Media Processing implementado com a biblioteca Xine usa um módulo X.Org básico e otimizado implementado para fins de avaliação.

A versão da biblioteca Xine utilizada na implementação do Media Processing foi a de número 1.1.16.3 e a versão do X.Org usada foi a 7.5.

2.3 Exemplificação da implementação do componentes

Para exemplificar a operação realizada pelo Media Processing implementado com a biblioteca libVLC e Xine, é apresentado um trecho do método capaz de reproduzir o fluxo de vídeo de ambas versões. A Figura 2 apresenta o método implementado na versão utilizando-se a libVLC e a Figura 3 apresenta o método implementado na versão que se utiliza da biblioteca Xine e X.Org.

```
libvlcPlayMedia(Media video) {
        LibVLC instance = libvlcInitialize();
        mediaProcessingAllocate(instance);
        mediaProcessingPlay(instance, video);
        if(mediaProcessingGetState(instance) == ENDED) {
            mediaProcessingDeallocate(instance);
            libvlcDeallocate(instance);
        }
}
```

Figura 2. Método responsável por reproduzir um fluxo de vídeo implementado no Media Processing com a biblioteca libVLC.

A implementação usando a libVLC abstrai diversos detalhes necessários para a exibição do vídeo decodificado, como por exemplo a detecção do *driver* de saída, o tamanho e o *aspect ratio* da janela, resultando em uma implementação reduzida em termos de número de métodos e em número de linhas de código.

```
xinePlayMedia(Media video) {
        Xine instance = xineInitialize();
        Xorg display = xorgInitialize();
        xorgSetResolution(display, video->getWidth(), video-
        >getHeight());
        xorgSetAspectRatio(display, video->getAspectRatio());
        xineSetDisplay(instance,display);
          xineSetFPS(instance,video->getFPS());
        mediaProcessingAllocate(instance);
        mediaProcessingPlay(instance, video);
        if(mediaProcessingGetState(instance) == ENDED) {
            mediaProcessingDeallocate(instance);
            xineDeallocate(instance);
            xorgDeallocate(display);
        }
}
```

Figura 3. Método responsável por reproduzir um fluxo de vídeo implementado no Media Processing com a biblioteca Xine.

A implementação usando a biblioteca Xine requer a definição de uma série de métodos adicionais de forma a possibilitar a decodificação e a exibição do fluxo de vídeo. O módulo de exibição, por exemplo, requer a especificação da resolução da janela, a taxa de decodificação e exibição do fluxo de vídeo.

2.4 FlexCM

Os componentes do projeto GingaCDN estão sendo desenvolvidos utilizando-se o modelo de componentes chamado de FlexCM [11]. Cada componente que siga o modelo deve especificar as interfaces requeridas e, também, as interfaces providas a outros componentes. A responsabilidade de conectar os componentes é do FlexCM que realiza as conexões em tempo de execução.

Cada implementação de um componente deve especificar, também, dois arquivos, o *Architecture* e o *Registry*. O primeiro descreve os dados para a execução, como o caminho para a biblioteca dinâmica do componente e um identificador único para o componente. O segundo especifica quais conexões são utilizadas pelo componente, usando a identificação única definida na implementação da interface para cada componente. A versão do FlexCM utilizada na implementação do Media Processing foi a de número 0.2.

3. RESULTADOS EXPERIMENTAIS

3.1 Metodologia

Para testar os componentes implementados, foram utilizados dois computadores diferentes, um desktop, chamado de "Computador A" e um *netbook*, chamado de "Computador B". O Computador A possui um processador Intel Core 2 E6320 1.86GHz com 2 Gigabytes (GB) de RAM. O Computador B usa um processador Intel Core 2 Solo ULV SU3500 1.40GHz e conta com 3 GB de memória RAM. Ambos os computadores executam o sistema operacional Ubuntu 9.10.

Os dados dos experimentos foram obtidos através da análise de seus processos, usando o aplicativo *Procps* para capturar tais dados. Cada vídeo de teste foi executado por três minutos, repetindo-se por três vezes. Os dados relativos ao consumo de memória e a taxa de uso de processador pelo Media Processing foram coletados a cada segundo enquanto executava o fluxo de vídeo, resultando em 540 amostras para cada vídeo. No total, 2160 amostras foram analisadas para cada versão do componente, resultando no total de 8640 amostras, considerando as duas arquiteturas diferentes.

Essa metodologia provê uma análise de desempenho consistente para o componente Media Processing em casos reais de uso. Entretanto, apenas a reprodução de fluxos de vídeo foi analisada, devido à ausência de alguns recursos, como o redimensionamento de vídeo, no Media Processing implementado com a biblioteca Xine.

O Media Processing foi gerado através do compilador GCC, não sendo utilizada nenhuma otimização disponibilizada pelo mesmo. As bibliotecas libVLC e Xine foram compiladas e instaladas utilizando a configurações padrão das mesmas.

O conjunto de testes consiste de quatro vídeos progressivos (p) em três diferentes resoluções, o 848x480 (480p), conhecido como definição padrão (*Standard Definition*, SD) e o 1280x720 (720p) e 1920x1080 (1080p), conhecidos como de alta definição (*High Definition*, HD). O vídeo STS116 foi obtido de [10], o Taxi3 French, nomeado de Taxi, foi obtido de [9] e o Saguaro National Park, chamado de Park, e o Space Alone, chamado de Space, estão disponíveis em [1]. Os detalhes do vídeo são apresentados na Tabela 1 (vídeos 480p), Tabela 2 (vídeos 720p) e na Tabela 3 (vídeos 1080p). Todos os vídeos possuem uma proporção de tela (*aspect ratio)* de 16:9, utilizam-se do contêiner MP4, possuem uma taxa de 30 quadros por segundo (FPS) e não possuem faixa de áudio.

Tabela 1 – Detalhes do conjunto de vídeos 480p.

Nome	480p			
	Tamanho (MB)	Duração (m:ss)	Video Bitrate (kbps)	Resolução (pixel por pixel)
Park	95.7	5:20	2500	848x480
Space	55.7	3:06	2500	848x480
STS116	62.1	3:31	2500	848x480
Taxi	48.5	2:42	2500	848x480

Tabela 2 – Detalhes do conjunto de vídeos 720p.

Nome	720p			
	Tamanho (MB)	Duração (m:ss)	Video Bitrate (kbps)	Resolução (pixel por pixel)
Park	191	5:20	5000	1280x720
Space	111	3:06	5000	1280x720
STS116	124	3:31	5000	1280x720
Taxi	96.7	2:42	5000	1280x720

Tabela 3 – Detalhes do conjunto de vídeos 1080p.

Nome	1080p			
	Tamanho (MB)	Duração (m:ss)	Video Bitrate (kbps)	Resolução (pixel por pixel)
Park	382	5:20	10000	1920x1080
Space	222	3:06	10000	1920x1080
STS116	248	3:31	10000	1920x1080
Taxi	193	2:42	10000	1920x1080

Os vídeos foram codificados de fontes 1080p usando o codificador x264 [12] na versão 1376. O H.264 *High Profile* e o AVC nível 5.1 foram utilizados com uma taxa de *bits* constante para cada resolução. A taxa de quadros por segundo de todos os vídeos foram convertidos para 30 FPS. Alguns dos detalhes da codificação estão listados abaixo, elas são o padrão do *High Profile* e do codificador x264 com AVC nível 5.1 [12]:

- Codificador de Entropia CABAC (*Context-adaptive binary arithmetic coding*).
- Filtro de Deblocagem ativado, *strength* e *threshold* com valor 0.
- Estimação de Movimento Fracionária com *hexagonal algorithm* de tamanho 16x16 *pixels*, para as camadas de luminância e crominância, usando RDO para os quadros I e P [14].
- Transformada DCT *(Discrete Cosine Transform)* adaptável usando I4x4, I8x8, P8x8 e B8x8.
- Três *B-Frames* com bias igual a 0. Procura rápida por *B-Frames* adaptáveis e *B-Pyramid* desativado.
- Três quadros (*frames*) de referência com *Adaptive I-Frame Decision* ativado.

A avaliação foi realizada entre as duas versões do componente devido ao fato da inexistência de outro componente decodificador de mídia, *open-source*, que siga o padrão definido para o SBTVD.

3.2 Análise dos Resultados

3.2.1 Computador A

Estes experimentos mostram a eficiência do componente Media Processing em termos de custo de memória e taxa de uso de processador em um *desktop* (Computador A).

O uso de processador, em porcentagem, observado para as implementações do Media Processing usando a biblioteca Xine e libVLC ao rodar o conjunto de testes no Computador A é mostrada na Figura 4.

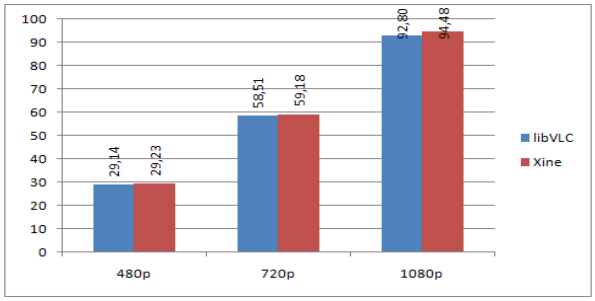

Figura 4 – Resultados para uso de processador, em porcentagem, utilizando o Computador A.

A variação no uso de processador entre as implementações usando a libVLC e a biblioteca Xine para os vídeos 480p foi de 0.30%, para os vídeos 720p foi 1.14% e, para os vídeos 1080p, foi de 1.81%.

Os custos de memória, em Megabytes, das implementações do Media Processing usando a libVLC e a biblioteca Xine é apresentada na Figura 5. A variação no custo de memória para os vídeos 480p foi de 67.17%, para os vídeos 720p foi de 48.48% e, para os vídeos 1080p, foi de 34.26%. A implementação com a libVLC usou mais memória que a implementação usando a biblioteca Xine para todas as resoluções de vídeos testadas.

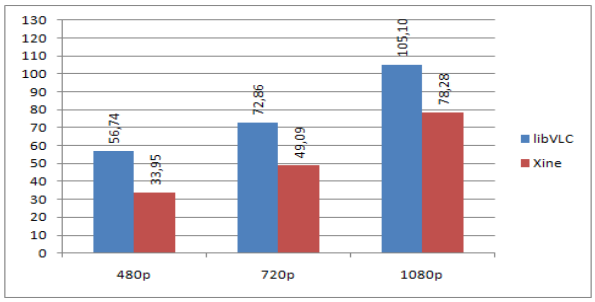

Figura 5 – Resultados para custo de memória, em Megabytes, utilizando o Computador A.

O processador, um Intel Core 2 Duo E6320 possuí um consumo de energia médio de 65.5W [4]. Esse consumo pode ser aceitável em um *set-top box*, já que não se trata de um aparelho portátil ou que utiliza-se de uma bateria. Contudo, essa taxa de consumo é inaceitável para a maioria dos sistemas embarcados atuais, como celulares, onde também serão usados futuramente recursos do

SBDTV. Em vista disso, experimentos com um processador de mais baixo consumo foram também realizados e os resultados são apresentados na seção 3.2.2.

3.2.2 Computador B

O principal objetivo de se realizar experimentos no Computador B foi para avaliar a eficiência das duas versões do componente quando rodando em um computador pessoal com um processador de baixo consumo de energia, característica desejada para diversos sistemas embarcados e aparelhos móveis. O Computador B possui um processador Intel Core 2 Solo ULV SU3500 baseado na arquitetura de núcleo único com freqüência de 1.4GHz e consumo de energético nominal aproximado de 5.5W [5].

As taxas de uso do processador, em porcentagem, encontradas para as implementações do Media Processing usando a biblioteca Xine e libVLC ao rodar o conjunto de testes no Computador B são mostradas na Figura 6.

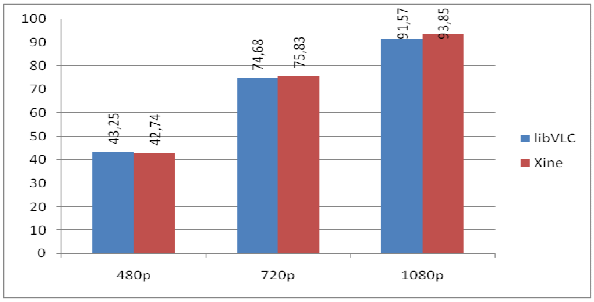

Figura 6 – Resultados para uso de processador, em porcentagem, utilizando o Computador B.

A variação no uso de processador entre as implementações usando a libVLC e a biblioteca Xine para os vídeos 480p foi de 1.21%, para os vídeos 720p foi 1.55% e, para os vídeos 1080p, foi de 2.5%.

Os custos de memória, em Megabytes, das implementações do Media Processing usando a libVLC e a biblioteca Xine é apresentada na Figura 7. Como previamente observado, quando utilizado o Computador B, a implementação usando a libVLC consome mais memória, independente da resolução do vídeo. A variação no custo de memória para os vídeos 480p foi de 28.99%, para os vídeos 720p foi de 20.74% e, para os vídeos 1080p, foi de 34.26%.

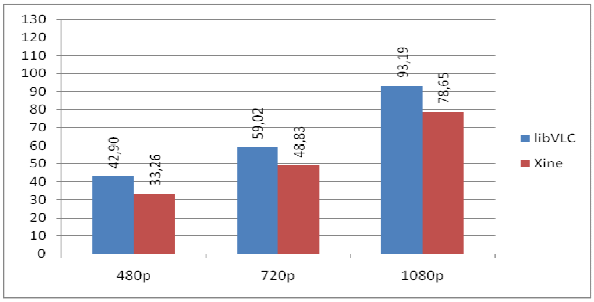

Figura 7 – Resultados para custo de memória, em Megabytes, utilizando o Computador B.

3.2.3 Resultados Gerais

A implementação usando a biblioteca Xine, em termos de uso do processador, apresentou um consumo 1.34% maior que a libVLC, por outro lado, em termos uso da memória apresentou um consumo 45.51% menor. Esse mesmo comportamento se manteve, nos testes realizados com o Computador B, onde a biblioteca Xine, em termos de uso do processador, apresentou um consumo 1.38% maior que a libVLC. Já em termos de uso da memória, a Xine obteve um consumo 21.39% maior que a libVLC.

Os experimentos no Computador A e no Computador B não podem ser comparados diretamente devido às diferenças do hardware utilizado, como placa-mãe, memória disponível, taxa de acesso ao disco rígido e, também, aos processos executando em cada sistema operacional. Entretanto, os resultados demonstram que, em termos de uso de processador, a implementação do Media Processing usando a libVLC obteve melhores resultados. Por outro lado, em termos de custo de memória, a melhor escolha é a implementação usando a biblioteca Xine, que requer menor quantidade de memória para ser executada.

4. CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho apresentou duas implementações do componente Media Processing para o GingaCC usando as bibliotecas libVLC e Xine. Experimentos foram realizados para avaliar a eficiência de ambas as implementações em termos de uso de processador e custo de memória em duas arquiteturas diferentes.

Os resultados dos experimentos demonstram um desempenho superior da implementação utilizando a biblioteca Xine em termos de uso de processador, com uma diferença de aproximadamente 2% em comparação com a versão baseada na libVLC. Já no custo de memória, a implementação usando a biblioteca Xine apresentou melhores resultados com um consumo de 45.51% e de 21.39% menores em relação à implementação utilizando a libVLC no Computador A e Computador B, respectivamente.

Embora ambas as implementações do componente sejam semelhantes, o maior consumo de memória percebido na implementação da biblioteca libVLC deve-se ao fato de a biblioteca incluir nativamente módulos X11, XVideo e *FrameBuffer* para saída de vídeo. Há, ainda, módulos para recepção de mídia via Internet através de diversos protocolos e módulos para processar e renderizar legendas em diversos formatos como o ASS. Na biblioteca Xine, esses componentes devem ser implementados pelo programador, reduzindo o tamanho da biblioteca e, em conseqüência, o consumo de memória.

A implementação do Media Processing usando Xine apresentou um leve acréscimo no consumo de processador devido ao método utilizado para renderizar o vídeo no *display*, tendo sido necessário adaptar o módulo X.Org para aceitar a saída nativa da biblioteca Xine, causando um gargalo e aumentando o uso de processador.

Para trabalhos futuros, pretendemos adicionar suporte a fluxos de áudio em ambas as implementações do componente Media Processing, assim como métodos diretamente relacionados, como controle de volume, seleção da faixa de áudio, entre outros. Após

isso, a integração do Media Processing com outros componentes do GingaCDN será realizada, a fim de prover uma implementação de referência integrada e modular do *middleware* Ginga.

5. AGRADECIMENTOS

Agradecemos ao Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC) e a Rede Nacional de Ensino e Pesquisa (RNP) pelo suporte financeiro e ao Laboratório de Aplicações de Vídeo Digital (LAVID) pelo apoio técnico, os quais tornaram possível o desenvolvimento desse trabalho.

6. REFERENCIAS

[1] Adobe, “Adobe Flash HD Gallery”, adobe.com, 2008. [Online]. Disponível em: http://www.adobe.com/mena/products/hdvideo/hdgallery/ [Acesso: 05 abr. 2010].

[2] G. L. S. Filho; L. E. C. Leite; C. E. C. F. Batista. “Ginga-J: The Procedural *Middleware* for the Brazilian Digital TV System”. Journal of the Brazilian Computer Society, 2007, vol.12, pp.47-56.

[3] GingaCDN. Ginga Code Development Network. Disponível em: http://ginga.lavid.ufpb.br/. [Acesso: 14 abr. 2010].

[4] Intel, “Intel Core 2 Duo Processor E6320”, intel.com, 2010. [Online] Disponível em: http://ark.intel.com/Product.aspx?id=29754. [Acesso: 05 abr. 2010].

[5] Intel, “Intel Core 2 Solo Processor ULV SU3500”, intel.com, 2010. [Online] Disponível em: http://ark.intel.com/Product.aspx?id=37133. [Acesso: 05 abr. 2010]

[6] L. F. G. Soares; R. F. Rodrigues; M. F. Moreno, “Ginga-NCL: the declarative environment of the Brazilian Digital TV System”. Journal of the Brazilian Computer Society, 2007, pp.37-46.

[7] LibVLC, “libVLC – VideoLAN Wiki”, videolan.org, 2010. [Online] Disponível em: http://wiki.videolan.org/Libvlc [Acesso: 05 abr. 2010].

[8] M. F. Moreno, “Um *middleware* declarativo para Sistemas de TV Digital Interativa”. Dissertação (Mestrado em Informática), PUC-Rio, Rio de Janeiro, 2006. p.105.

[9] Microsoft, "WMV HD Content Showcase", microsoft.com, 2004. [Online]. Disponível em: http://www.microsoft.com/windows/windowsmedia/musicandvideo/hdvideo/contentshowcase.aspx [Acesso: 05 abr. 2010].

[10] Nasa, "NASA High Definition Video", nasa.gov, Dec. 16, 2009. [Online]. Disponível em: http://www.nasa.gov/multimedia/hd/index.html. [Acesso: 05 abr. 2010].

[11] S.M. Filho, et al, "FLEXCM - A Component Model for Adaptive Embedded Systems", in Proc. COMPSAC (1), 2007, pp.119-126.

[12] X264, “x264 – VideoLAN”, videolan.org, 2010. [Online] Disponível em: http://www.videolan.org/developers/x264.html. [Acesso: 05 abr. 2010].

[13] Xine, “The Xine project”, Xine-project.ogg, 2010. [Online] Disponível em: http://www.Xine-project.org/ [Acesso: 05 abr. 2010].

[14] W. T; S. G. J; B.G; L. A. “Overview of the H.264/AVC Video Coding Standard”. IEEE Transactions on Circuits and Systems for Video Technology, 2003, vol.13, n7, pp.560-576.