Avaliando o impacto de otimizações em diferentes níveis em aplicações embarcadas com interface gráfica

Deivide Possamai
Ciência da Computação
UNIOESTE- Campus Cascavel
Cascavel, Brasil
Deivide.possamai@unioeste.br

Marcio Seiji Oyamada
Ciência da Computação
UNIOESTE- Campus Cascavel
Cascavel, Brasil
Marcio.oyamada@unioeste.br

***Abstract*— O uso de crescente de soluções embarcadas com interface com o usuário, ampliam ainda mais o espaço de otimização possível no desenvolvimento de tais sistemas. Este artigo apresenta algumas otimizações realizadas em uma lupa digital implementada na plataforma Beagleboard. As otimizações foram realizadas em três níveis distintos: utilização de instruções SIMD, configuração do servidor gráfico e escolha do formato de captura de imagem. Os resultados mostraram um fator de ganho de até 2,27 vezes em relação à configuração inicial, reforçando a necessidade de um desenvolvimento integrado que contemple os vários componentes que compõem um sistema embarcado.**

Keywords—Sistemas embarcados, Interface gráfica com o usuário, Linux embarcado

I. INTRODUCTION

Um sistema embarcado (SE) envolve diferente componentes de hardware e software onde a integração otimizada de tais componentes é fator crítico para atender aos requisitos restritos de desempenho, potência, custo entre outros.

Este artigo apresenta as otimizações e seus impactos em diferentes níveis do ampliador digital xLupa embarcado. A solução envolve a captura de imagens via webcam, processamento e saída em uma TV ou monitor. O desempenho geral do sistema é influenciado por diversos fatores tais como o sistema operacional, *drivers* (tanto dos dispositivos de entrada como de saída), e a aplicação em si.

II. TRABALHOS RELACIONADOS

O desenvolvimento de sistemas embarcados é guiado por requisitos restritos, o que tipicamente resulta na adoção de processadores com menor desempenho e tamanho de memória. Em [1] é analisado o impacto da utilização de um sistema operacional (SO) no desenvolvimento de uma estação de temperatura embarcada. Os resultados apontam que o uso do SO pode reduzir em aproximadamente 40%, e aumentar em até 275% o uso da memória.

O desenvolvimento de câmeras inteligentes para monitoramento e reconhecimento é um exemplo típico dos requisitos impostos no desenvolvimento de um sistema embarcado. Em [2] é apresentado um estudo de caso do desenvolvimento de uma câmera inteligente utilizando processadores DSP, onde otimizações em diferentes níveis podem acelerar em até 6 vezes a taxa de frames.

III. AMPLIADOR DIGITAL XLUPA

O xLupa embarcado é uma lupa digital para ampliação de documentos impressos para pessoas com baixa visão [3]. O objetivo xLupa embarcado é oferecer uma lupa digital compacta, que possa ser acoplada em monitores e TVs com entrada HDMI. O xLupa oferece diversos perfis, eles são necessários devido aos diferentes tipos de documentos impressos e características de usuários. Dados os requisitos citados anteriormente, foi adotado como plataforma de desenvolvimento o dispositivo Beagleboard-xM[4], construída utilizando a solução SoC DM3730, da Texas Instruments. A mesma é composta pelo ARM Cortex-A8 de propósito geral, e o DSP C64x+. O xLupa embarcado utiliza a distribuição Ubuntu versão 12.04.

IV. OTIMIZAÇÃO DO XLUPA

O desenvolvimento do xLupa embarcado é uma adaptação de um ampliador de tela para *desktop* chamado de xLupa. A diferença fundamental é que a versão *desktop* deve ampliar a imagem da tela do sistema operacional, enquanto a versão embarcada utiliza uma webcam conectada a porta USB. A seguir são apresentadas algumas otimizações realizadas e seus impactos no desempenho. Os testes foram realizados habilitando-se previamente o nível máximo de otimização do compilador e uso de instruções de ponto flutuante em hardware. O compilador utilizado foi o gcc versão 4.8.

A. Configuração do servidor X

Em computadores executando o sistema operacional Linux o ponto inicial para melhorar o desempenho de aplicações gráficas é a instalação do *driver* específico para a placa gráfica. No caso da Beagleboard o *driver* mais atualizado com otimizações NEON já estava previamente instalado e portanto tal otimização já havia sido realizada. Com os dados do *profiling* foi possível notar que a função *cairo_paint* responsável pelo envio da imagem para o dispositivo de saída era responsável por mais de 50% do tempo de processamento

por frame. A configuração padrão do sistema utiliza a resolução 1280x720, com a profundidade de cor de 16 bits. Uma alternativa testada foi alterar a configuração para uma profundidade de cores de 24 bits, visto que os algoritmos de aplicação de perfil e a imagem utilizam o formato RGB24. Como resultado, o tempo de processamento por frame caiu de 386 ms para 225 ms para o perfil normal, uma redução de aproximadamente 71%. No caso da utilização do perfil com alteração de contraste, o tempo de processamento por frame reduziu de 581 ms para 420 ms. A redução é resultado direto da adoção da mesma profundidade de bits da imagem tratada pelo xLupa e do servidor gráfico, evitando-se a necessidade de qualquer conversão ou ajuste.

B. Formato de captura da webcam

A webcam utilizada pelo xLupa suporta os formatos de captura MJPEG (*multiple* JPEG) e YUV. A biblioteca V4L utilizada no processo de captura de imagens suporta inúmeros formatos de captura, e caso este seja diferente do nativo uma conversão é realizada internamente na biblioteca. No caso do xLupa o formato utilizado é o RGB24. Configurando a captura para este formato, a biblioteca V4L internamente escolhe o formato de captura MJPEG da webcam. A escolha é guiada principalmente, devido a maior taxa de captura neste formato que é de 30 FPS, enquanto que o formato YUV tem taxa de captura de 10 FPS. Uma outra possibilidade é forçar a captura em YUV e depois realizar na aplicação a conversão para RGB24. Esta alternativa foi testada inicialmente em um notebook com processador i5. Nos experimentos realizados, o diferença no tempo de processamento por frame entre o captura MJPEG ou YUV é quase nulo, necessitando de aproximadamente de 16 ms para ambos os casos. No entanto, devido as diferentes taxas de captura dos formatos, na versão original é obtido um desempenho de 30 FPS, e no formato YUV de 10 FPS.

Os testes foram repetidos na Beagleboard executando-se a versão modificada, com a captura em YUV e conversão para RGB na aplicação. Na plataforma embarcada o resultado difere do observado no notebook, e a versão modificada resultou em um ganho de aproximadamente 28%, reduzindo o tempo de processamento por frame de 225 ms para 175 ms no perfil normal, e de 420 ms para 370 ms no caso do perfil com contraste. Este ganho é resultado da menor complexidade da conversão de YUV para RBG24 se comparada com a conversão JPEG para RGB24.

C. Otimização utilizando instruções SIMD

Dado o resultado anterior, uma possibilidade direta de ganho de desempenho é a utilização da extensão de instruções multimídia SIMD, que no processador ARM é denominado de NEON.

Em um primeiro momento, foram ativados os *flags* do compilador para que as operações SIMD sejam utilizadas. Analisando o código Assembler gerado, verificou-se que na função de conversão nenhum código utilizando instruções SIMD foi gerado. Desta forma, a solução foi explorar manualmente o paralelismo de dados, através da codificação explícita utilizando instruções NEON.

A utilização de instruções NEON, resultou em um ganho de 17 %, reduzindo o tempo de processamento por frame para 150 ms para o perfil normal. O ganho obtido foi muito baixo se comparado com a o paralelismo teórico de 8 instruções, pois é necessário um rearranjo dos dados nos registradores NEON, comprometendo assim o ganho de desempenho.

A Tabela 1 apresenta a síntese dos resultados obtidos com as três otimizações realizadas: configuração do servidor X, utilização da captura em YUV e utilização de instruções NEON no algoritmo de conversão YUV para RGB. O fator de ganho considera a versão original como base.

Tabela 1. Tempo de processamento e ganho em relação a versão original

	Original	**Xorg config**	**YUV captura**	**SIMD**
Perfil normal (ms)	386	225	175	150
Fator de ganho	1,00	1,72	2,21	2,57
	Original	**Xorg config**	**YUV captura**	**SIMD**
Perfil contraste (ms)	581	420	370	346
Fator de ganho	1,00	1,38	1,57	1,68

V. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou o impacto de algumas otimizações realizadas em diferentes componentes que fazem parte de um sistema embarcado caracterizado pelo uso intensivo da interface gráfica. A otimização envolvendo o servidor X é a de maior impacto, mas é obviamente restrita ao tipo de aplicação, servidor gráfico e plataforma. A segunda otimização apresenta particularidades relacionadas ao sistema embarcado que podem diferir das soluções adotadas em sistema *desktop*. Em um *desktop* o impacto de uma decodificação JPEG é importante, mas é deixada em segundo plano quando se considera a taxa de captura. No caso de um sistema embarcado, a mesma consideração não pode ser feita, e o custo da decodificação JPEG aumenta em 28% o tempo de processamento por frame.

Uma terceira otimização foi a utilização de instruções SIMD visando extrair o melhor desempenho usando os recursos do processador. Em conjunto, as três otimizações resultaram em um fator de ganho de até 2,57 vezes.

Como trabalhos futuros, um estudo da migração de algumas partes para execução no processador DSP existente na plataforma será realizado.

Referências

[1] R.P. Jasinski, M.R. Moroz and V.A Pedroni. “The impact of operating system adoption in an embedded project: A case study”. VIII Southern Conference on Programmable Logic (SPL), 2012, pp.1-7, March 2012.

[2] Wolf, W.; Ozer, B.; Tiehan Lv, "Smart cameras as embedded systems," Computer , vol.35, no.9, pp.48,53, Sep 2002.

[3] F. Fernandes, D. Possamai, M. Oyamada. “Desenvolvimento de um ampliador digital para pessoas com baixa visão”. Competição Intel-Simpósio Brasileiro de Sistemas Computacionais, 2013.

[4] BeagleBoard. Disponível em http://www.beagleboard.org