Suporte ao Processamento Vetorial em Arquiteturas x86

Lucas Correia Villa Real*, Atila Bohlke Vasconcelos**,
Gerson Geraldo H. Cavalheiro*

*UNISINOS – Universidade do Vale do Rio dos Sinos
Centro de Ciências Exatas e Tecnológicas
São Leopoldo, RS, Brasil
{lucasvr, gersonc}@exatas.unisinos.br
**UFRGS – Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre, RS, Brasil
bohlke@inf.ufrgs.br

Introdução

Atualmente existe um grande número de agregados de computadores compostos de poderosas arquiteturas, capazes de atingir bons níveis de desempenho. Este nível de desempenho, no entanto, limita-se à exploração dos recursos de processamento convencionais: processadores e módulos de memória. No entanto, tanto em agregados de computadores como em redes de estações (NOW), outros recursos de processamento encontram-se disponíveis. Por exemplo, o poder computacional de impressoras, dos DSPs (*Digital Sound Processor*) em placas de som e o processador das placas gráficas. Este trabalho aborda o uso da capacidade de processamento do hardware das placas gráficas (GPU) como auxílio na busca por desempenho nas mais diversas aplicações, não restringindo-se apenas à computação gráfica (CG).

Este artigo apresenta a arquitetura baseada em pipelines vetoriais, na qual baseiam-se placas desenvolvidas pela ATI, NVidia e Matrox, entre outras. Algumas formas de como explorar estas GPUs, vantagens e problemas que podem ser encontrados durante o desenvolvimento de aplicativos que farão uso destes recursos também são abordados.

Potencialidades

As arquiteturas SIMD atuais mais comuns, como a família *x86*, utilizam pipelines para inteiros e ponto flutuante, quando estas necessitam realizar operações sobre os dados. É bastante comum no meio acadêmico e científico estes dados estarem representados por vetores. Seria natural, portanto, processá-los de maneira vetorial, utilizando-se de instruções específicas para tal. Algumas arquiteturas (usualmente RISC) implementam o pipeline vetorial, como por exemplo a família *PowerPC*, cujos principais representantes são o IBM e a Motorola, com o AltiVec. Uma das vantagens da utilização de instruções vetoriais, além da facilidade de representação das instruções, está no fato de que muitas destas instruções podem ser executadas em um único ciclo de clock – possibilitando ganho de desempenho na manipulação de vetores.

* ITI-CNPq

O desvio de fluxos de execução para a GPU possibilita a execução de instruções vetoriais diretamente no hardware gráfico, o que permite que a CPU seja utilizada para outros fins. Isto possibilita cálculos paralelos entre estes processadores, de forma a obter *multi-processamento assimétrico* (paralelismo real). A simetria com a CPU pode ser definida através de uma API que garanta que as instruções enviadas à GPU tenham sido decodificadas e executadas.

Arquitetura

A arquitetura apresentada nos recentes hardware gráficos baseia-se em um modelo de pipeline. Este modelo é composto por mecanismos de processamento de vértices onde os programas executados neste componente são chamados de *vertex shaders*, e de pixels (*pixel pipeline*), onde os programas que nele executam são chamados de *pixel shaders*. Ambos mecanismos de processamento são programáveis, e cada um destes é uma unidade de processamento independente, contendo um programa, uma região para armazenamento de dados e uma unidade lógica e aritmética.

Os dados de entrada no pipeline gráfico são *streams* de vértices conectadas de forma a gerar superfícies, como triângulos e superfícies polinomiais. Assim que os vértices são processados pelo *vertex engine* eles são agrupados em triângulos, e a rasterização é feita para discretizar a superfície em pixels. Cada pixel é enviado ao *pixel pipeline*, onde um programa é executado para determinar a cor deste conjunto de pixels. Nestas operações, os *vertex engines* e o *pixel pipeline* cooperam como máquinas SIMD (*Single Instruction, Multiple Data*) [ERA 2001], pois eles processam vários vértices ou pixels em paralelo usando as mesmas instruções.

É de se notar também que hardware mais modernos de vídeo podem incluir diversos controladores de memória, conectados aos bancos de memória através de dispositivos de *fábricas de switchs*. Outro fator que pesa na decisão de realizar processamento na GPU, é que muitas vezes ela é também *superscalar*. No caso do modelo ATI Radeon 9700, a GPU possui 4 *vertex shaders*, isto é, 4 processadores vetoriais, cada um consistindo de um pipeline de 32 bits de largura para inteiros, e outro pipeline vetorial de 128 bits de largura. Essa mesma GPU possui ainda 8 *pixel shaders*, cada um composto de um pipeline de ponto flutuante. Se estas características forem comparadas ao número de pipelines que uma CPU tem em geral (no caso do AMD K7 são 3 pipelines de ponto flutuante e 3 de inteiros), percebe-se por que é interessante delegar processamento para a GPU.

Recursos de Programação Disponíveis em GPUs

Para que os recursos disponíveis no hardware gráfico sejam utilizados, estes precisam ser acessados de alguma forma. As GPUs modernas permitem que as operações sejam realizadas através de instruções diretas no hardware.Linguagens como CG e a Pixar's RenderMan dispõem de APIs de programação de alto nível para utilizar os recursos de processamento da GPU. Outra proposta, apresentada em [NAV 95], define uma API para acesso a hardware especializado. Abstrações como estas são desejáveis, principalmente quando deseja-se que o software desenvolvido seja portável entre diferentes arquiteturas.

Compiladores atuais apresentam funções para utilizar recursos específicos dos processadores, como o suporte às extensões do conjunto de instruções MMX, SSE e 3DNow!. A versão 3.2 do compilador GCC [USI 2002] dispõe de uma interface para lidar com este conjunto de instruções. Estas mesmas extensões contêm instruções vetoriais SIMD que operam em múltiplos dados contidos em um registrador vetorial ao mesmo tempo. A interface fornecida pelo GCC permite que instruções vetoriais presentes nas extensões MMX, SSE e 3DNow! sejam feitas através de funções *built-in*. Esta mesma interface pode ser utilizada para encapsular as instruções suportadas pelas GPUs. Para isto, é necessário definir os tipos de dados necessários, bem como as funções básicas que irão manipular este tipo de dados. A definição dos tipos de dados pode ser feita com o uso de um *typedef*, como *typedef int v4si __attribute__ ((mode(V4SI)));*

Desta forma, o tipo de dados representado por *__attribute__* define o modo a ser utilizado: para tipos vetoriais, o formato é VnB, onde n é o número de elementos do vetor, e B é a base dos elementos. As seguintes bases são suportadas pelo GCC:

- QI: Inteiro, de tamanho da menor unidade de endereçamento, geralmente 8 bits;
- HI: Inteiro, com o dobro do tamanho de QI, geralmente 16 bits;
- SI: Inteiro, quatro vezes maior que o tamanho de QI, geralmente 32 bits;
- DI: Inteiro, oito vezes maior que o tamanho de QI, geralmente 64 bits;
- SF: Um valor de ponto flutuante, do tamanho de SI, geralmente 32 bits;
- DF: Um valor de ponto flutuante, do tamanho de DI, geralmente 64 bits.

Os modos válidos, no entanto, são determinados pela arquitetura alvo. As extensões *MMX* do *x86*, por exemplo, suportam apenas os modos V8QI, V4HI e V2SI. Desta forma, o suporte para a utilização dos recursos de uma GPU em um compilador deve ser feito a partir da definição de funções *built-in*, que irão então tratar os tipos de dados definidos previamente.

Mapeamento de Aplicações no Hardware Gráfico

Algumas operações comumente utilizadas por programas que busquem alto desempenho são realizadas sobre matrizes e vetores. Visto que a GPU suporta a execução de instruções vetoriais, é possível que uma operação de multiplicação de matrizes, por exemplo, seja feita no hardware gráfico, enquanto a CPU pode executar alguma outra rotina em paralelo. No entanto, como estas GPUs não suportam saltos condicionais, não é possível realizar laços em hardware, e isto faz com que a CPU precise ser utilizada para manter controle sobre cada iteração mapeada na GPU. Isto pode ser feito de maneira transparente ao programador, através da criação de instruções e registradores virtuais auxiliares para realizar o controle destes laços.

Desta maneira, é possível enviar dados para serem processados pela GPU, realizar algum outro cálculo na CPU, e então sincronizar os dados processados na GPU, vindos em uma imagem *RGBA* na forma de pixels. Isto permite que duas tarefas sejam executadas em paralelo, e a aplicação pode vir a obter ganhos substanciais quando este mapeamento

for bem definido. Os problemas relativos ao barramento podem ser aliviados caso os dados gerados pela GPU sejam mantidos na memória de vídeo, nos *buffers off-screen*.

Baseado na arquitetura presente nas placas ATI Radeon 8500, que apresenta registradores vetoriais compostos por 4 unidades de ponto flutuante, uma matriz 8×3 poderia ser multiplicada de forma a utilizar esta característica. Para tal, uma possível abordagem é realizar o ajuste da matriz para dimensões múltiplas de 4, preenchendo os novos elementos com zeros.

Conclusão

A utilização do hardware gráfico no auxílio à CPU para a realização de cálculos computacionais já é comum em aplicações gráficas, tais jogos e simuladores. Nestes ambientes o auxílio do hardware gráfico costuma ser aproveitado para fins de renderização e aceleração destes processos, aplicações estas que baseiam-se nos mesmos tipos de dados suportados pelas GPUs, aproveitando-se de vários recursos destas placas. Neste trabalho foi realizado um estudo das potencialidades deste hardware para um futuro uso no processamento de alto desempenho.

Para que os recursos de processamento da GPU possam ser aproveitados pelo programador, a escolha de um compilador que permita extender suas funcionalidades, como o GCC, é de vital importância para o desenvolvimento de um ambiente que permita este multiprocessamento assimétrico. A interface oferecida por compiladores como o GCC para a extensão de instruções encontradas em vários processadores pode ser utilizada para encapsular as instruções suportadas por diversas GPUs, criando um primeiro nível de abstração para o programador. APIs definidas a partir de bibliotecas podem explorar o uso de *threads* para gerenciar as tarefas entre os diferentes processadores, permitindo, além disto, a criação de programas independentes da GPU.

Um fator que ainda deve ser considerado é a portabilidade de tal projeto: a implementação realizada utilizando-se apenas de funções *built-in* fornecidas pelo compilador executam apenas nesta mesma GPU. Esta característica já acontece com processadores de diferentes arquiteturas. No entanto, a tecnologia utilizada nas placas gráficas desenvolve-se com uma grande velocidade, e recursos encontrados nas placas atuais podem desaparecer de futuras placas gráficas.

Como trabalho futuro, uma aplicação protótipo será desenvolvida para verificar a aplicabilidade e as potencialidades reais da solução proposta. Especial atenção será dada à exploração destes recursos em um agregado de computadores.

Referências

[ERA 2001] ERAD'2001 ESCOLA REGIONAL DE ALTO DESEMPENHO, 2001, Gramado. **Anais...** [S.l.: s.n.], 2001.

[NAV 95] NAVAUX, P. O. A.; ROSE, C. A. F. D.; CAVALHEIRO, G. G. H. Performance evaluation in image processing with gapp array processor. **Microprocessing and Microprogramming, The Euromicro Journal**, v.41, p.71–82, 1995.

[USI 2002] **Using the GNU Compiler Collection (GCC)**. [S.l.]: Free Software Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307 USA, 2002.