INSTITUTO DE COMPUTAÇÃO - UNICAMP

Aceleração de uma Aplicação Científica com OpenCL: Regularização de Dados Sísmicos

Vanderson M. do Rosario

25 de junho de 2016

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

1 INTRODUÇÃO

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

[1]

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermen-

tum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

2 Materiais e Métodos

Este trabalho apresenta uma sequência de transformações aplicadas sobre um código, inicialmente sequêncial, para que esse obtenha máximo desempenho sobre um processadores *multicore* e uma placa de vídeo por meio do *framework* OpenCL.

Durante o desenvolvimento do trabalho, diversos experimentos foram realizados para medir a eficiência de cada implementação e de cada transformação, tanto para guiá-las quanto para mostrar os impactos das mesmas. Nessa seção, apresentamos a lista dos materias utilizados e as técnicas utilizadas para realização e medição dos experimentos.

2.1 Materiais

Todos os experimentos foram realizados sobre uma mesma máquina, um Dell Optiplex 9020, equipado com um processador Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz com quatro unidades de processamento e uma GPU Intel(R) HD Graphics 4600 com 20 unidades de processamento com frequência máxima de 1150 MHz. Ainda, a máquina é equipada com 2 pentes de 4GB DDR3 SDRAM de 1600MHz em multiplos canais.

Para compilar o código fonte, foi utilizado o GCC versão 4.8.5 e sobre a plataforma CentOS com um *third-party kernel*: 4.4.131.el7.elrepo.x86_64. Entre os frameworks utilizados, o OpenMP na versão x.x.x e o OpenCL 1.2 com o driver da Intel na versão 16.4.4.47109.

Para todos os experimentos, o código fonte foi compilado com o seguinte comando:

```
$ gcc -O3 --std=c99 reg.c semblance.c su.c -lm -I. -lOpenCL -fopenmp
```

Listing 1: Comando para compilar o código fonte dos experimentos.

As variáveis de ambiente do sistema ....

2.2 Experimentos e Código Fonte

Falar como os experimentos foram feitos e calculados. Falar como o tempo foi medido, sobre o perf e sobre o código para pegar o tempo e como dividimos o tempo em inicialização e execução.

Falar como conseguir o código fonte e como eles são apresentados durante o artigo.

3 Desenvolvimento e Resultados

Falar sobre as otimizações aplicadas sobre o código sequêncial, sobre o OpenMP e como o objetivo é usar OpenCL para paralelizar o código para arquiteturas heterogênias e testar o desempenho em CPU e GPU.

3.1 Código Sequêncial

Explicar a divisão do código em 3: inicialização, kernel e finalização.

Falar sobre o impacto de cada etapa e quais os trexos mais quentes.

Mostrar o impacto das otimizações do compilador, O0, O1 e O2 no código.

3.2 Paralelização com OpenMP

Mostrar como foi implementado a paralelização com OpenMP.

Falar que o OpenMP só garante paralelização para CPUs e como o objetivo do trabalho não era OpenMP as otimizações não foram aplicadas nele.

Mostrar o impacto das otimizações O0, O1 e O2 no código com OpenMP.

Falar que o compilador não conseguiu vetorizar o código.

3.3 Paralelização com OpenCL

Falar sobre o OpenCL e como ele funciona.

Falar sobre o que foi necessário para transformar o código do OpenMP para rodar com OpenCL.

Falar sobre CPU bond e Memory Bond e como podemos contornar esses problemas com OpenCL. Falar sobre como fazer vetorização com OpenCL e como a hierarquia de memória funciona. Falar dos manuais.

Instroduzir as otimizações que seram aplicadas.

3.3.1 MULTIDIMENSÕES

Falar que diretamente, a GPU ainda não funcionava.

Falar que ao adicionar mais dimensões houve melhoria no desempenho da CPU e a GPU começou a funcionar.

Mostrar resultados de desempenho para 2D e 3D.

3.3.2 REMOVENDO DADOS DESNECESSÁRIOS

Mostrar que no código original grande parte dos dados não estava sendo utilizado. Mostrar como tirar esses dados e como isso manteve a corretude.

Mostrar impacto no desempenho.

3.3.3 *Inlining* DAS FUNÇÕES

Falar sobre o fato que mesmo os manuais dizendo que o inlining é feito SEMPRE. Vários relatam a mudança de desempenho com inlining manual.

Argumentar que o inlining manual pode criar oportunidade de otimizações manuais que o compilador talvez não consiga fazer.

Mostrar impacto no desempenho.

3.3.4 MELHOR *Local Work Size*

Comentar sobre a escolha dos work-local-size.

Falar sobre manual do hardware e clinfo.

Mostrar os melhores valores encontrados.

3.3.5 SIMPLIFICAÇÕES ALGÉBRICAS

Mostrar que os inlinings no kernel permitiram a aplicação de várias simplificações algébricas.

Argumentar que o compilador pode não estar fazendo as simplificações porque os dados são floats, comentar sobre as otimizações (https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml

Mostar o impacto no desempenho CPU e GPU.

3.3.6 *Loop Invariant Code Motion*

Mostrar como alguns trexos do código são independentes das variáveis de indução e por isso podem ser movidos para fora do loop e até mesmo para fora do kernel.

Mostrar o impacto no desempenho do código.

3.3.7 *Constant Memory Space*

Explicar sobre a constant memory space nas GPUs, falar sobre o tamanho dessa memória na GPU da Intel.

Falar quais dados foram escolhidos para ir memória constant.

Mostrar o impacto no desempenho.

3.3.8 Reduzindo a Pressão Sobre os Registradores

Falar sobre a hipótese fato que o desempenho do kernel estava sendo limitado por memory bound.

Mas, tirar os acessos diretos a memória não estavam surtindo efeito e eram sequências.

Falar que havia a desconfiança que estava ocorrendo register spill.

Mostrar como os dados foram levados a memória local.

Mostrar o impacto no desempenho.

3.4 Otimizações Não Implementadas ou Não Mantidas

Falar sobre o fato que algumas otimizações não foram aplicadas.

3.4.1 Vetorização

Falar sobre vetorização no OpenCL e como tentamos aplicar, mas que não houve nenhum benefício.

3.4.2 Redução

Falar sobre a posibilidade de fazer alguma espécie de redução, mas que não conseguimos encontrar espaço para essa otimização.

3.4.3 *Loop Blocking*

Argumentar que o código não é memory bound e que blocking não seria eficiente.

4 Problemas Encontrados

Falar sobre manter corretude sem a alocação;

Falar que não foi escolhido adicionar diversos defines.

Argumentar que muitas das otimizações podem ter diminuido a corretude do código para outras entradas.

5 Trabalhos Futuros

Falar sobre analisar o código sobre uma ferramenta sofisticada e investigar se estamos próximos do limite teórico do hardware.

Falar sobre aplicar essas otimizações sobre o OpenMP e medir o desempenho.

Falar sobre as posiveis otimizações que ainda podem ser aplicadas.

6 CONCLUSÃO

Tentar analisar o desempenho teórico do hardware. Tentar argumentar o desempenho alcançado.

Falar sobre o OpenCL e as otimizações encontradas. Falar sobre os problemas e os trabalhos futuros.

REFERÊNCIAS

[1] D. Knuth, D. Bibby, and I. Makai. *The texbook*, volume 1993. Addison-Wesley, 1986.