Explorando OpenCL na Paralelização de uma Sub-rotina do Modelo Ambiental CCATT-BRAMS

Cezar Augusto Contini Bernardi[1], **Otavio Migliavacca Madalosso**[1], **Andrea Schwertner Charão**[1], **Haroldo de Campos Velho**[2], **Renata Ruiz**[2]

[1]Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria – UFSM

[2]Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais – INPE

{cbernardi,omadalosso,andrea}@inf.ufsm.br, {haroldo,renata}@lac.inpe.br

Resumo. *Este artigo descreve uma experiência preliminar de paralelização de código com OpenCL, tendo como alvo uma sub-rotina do sistema de modelagem ambiental CCATT-BRAMS. Os resultados evidenciam fontes de overhead em OpenCL e servem de ponto de partida para análises mais aprofundadas visando à aceleração do modelo.*

1. Introdução

OpenCL™(Open Computing Language) [Khronos Group 2012] é um *framework* para programação paralela em sistemas heterogêneos, que utilizam GPUs ou outros dispositivos junto à CPU como aceleradores do processamento. Por ser um *framework* aberto e padronizado, independente de fabricante, OpenCL aparece como uma alternativa conveniente para sistemas computacionais que demandem desempenho e portabilidade.

O sistema de modelagem ambiental CCATT-BRAMS [Freitas et al. 2007, CPTEC-INPE 2012] é um código de grande porte empregado em estudos e previsões sobre qualidade do ar no Brasil. Trata-se de um programa já paralelizado para execução em cluster, que vem sendo alvo de pesquisas visando explorar também o potencial de GPUs. O presente trabalho visa contribuir com tais pesquisas, explorando o uso de OpenCL aplicado ao CCATT-BRAMS. O trabalho se restringe a uma sub-rotina do sistema, a fim de reunir subsídios para um uso mais generalizado.

2. Fundamentação

2.1. OpenCL

O OpenCL funciona através da execução de um *kernel* em GPU, coordenado pela CPU. A CPU é responsável pelo reconhecimento da GPU, alocação dos *buffers*, definição dos argumentos do *kernel*, número de *threads* e forma que essas farão a execução, tudo isso através de chamadas OpenCL.

A execução do *kernel* é dividida entre as *threads*, diferenciando-se através de ThreadIDs. Cada uma delas manipula diferentes dados transferidos a GPU. O retorno dos dados é feito pela leitura dos *buffers* que foram definidos como *buffers* de escrita, através de outras chamadas feitas pela CPU.

2.2. CCATT-BRAMS

O Sistema de Modelagem Ambiental CCATT-BRAMS é bastante utilizado para previsão climática em todo o Brasil [Freitas et al. 2007, CPTEC-INPE 2012]. Sendo desenvolvido pelo INPE, este software requer alto poder de processamento, além disso, seu tempo de execução é elevado. Conta com 389.469 linhas de codigo, divididas em 539 arquivos.

A instalação do CCATT-BRAMS foi bastante trabalhosa, por requerer diversos arquivos e ajustes nas configurações de compilação. Além disso, muitas das dependências precisam ser compiladas de forma adequada para funcionarem.

A sub-rotina escolhida para paralelização foi a TKEMY, responsável pela parametrização de turbulência de Mellor-Yamada [Freitas et al. 2007, CPTEC-INPE 2012]. Essa rotina possui cerca de 220 linhas e conta com laços de repetição em cascata, o que pode ser um ponto positivo para paralelismo em grande número de threads. Sua execução é relativamente rápida (36,79 ms, em média), mas mesmo assim há interesse em melhorar seu desempenho, uma vez que é uma sub-rotina chamada muitas vezes durante uma simulação.

3. Paralelização

O ponto de partida da tradução do código em Fortran para execução com OpenCL foi a análise de parâmetros. Após este estágio, foi necessário um aprofundamento a respeito de como utilizar estes parâmetros, tendo em vista o acesso a matrizes tridimensionais, visto que a alocação de memória em Fortran difere da forma feita em C.

No passo seguinte foram as definições de variáveis, vetores e matrizes de entrada e saída, preparação dos dispositivos, contextos, filas de comandos, alocação dos buffers na memória da GPU, compilação e execução do kernel de forma paralela, ou seja, dividindo em pequenas partes que são executadas cada uma em uma thread da GPU, e leitura dos buffers de retorno.

3.1. Análise e Passagem de Parâmetros

Se tratando de transformações serial-paralelo, existem diversos pontos que devem ser verificados para avaliar a possibilidade de paralelização. Um dos requisitos mais marcantes é a necessidade de que os laços não tenham dependências com os laços anteriores. Por exemplo, em um *laço de i a n*, a rodada *n-1* não pode ter um acesso à posição *n-2*, o que poderia causar acesso a dados indefinidos.

Após a não identificação de nenhuma dependência que impossibilitaria a paralelização e definição de parâmetros necessários, analisou-se como seriam os recebimentos e acessos a esses parâmetros.

A passagem de parâmetros em Fortran sempre é feita através de referências a endereços de memória, ou seja, ponteiros em C. Sendo assim, os argumentos da chamada foram todos recebidos dessa forma, através de uma chamada em Fortran, fazendo com que as matrizes n-dimensionais, fossem interpretadas como unidimensionais. Para essa função, foi necessária a passagem de 49 parâmetros.

3.2. Acesso a Posições de Memória

Um empecilho na comunicação entre C e Fortran, é a forma de alocação de vetores n-dimensionais. Em C, os vetores com mais de uma dimensão (matri-

zes) são alocados como *row-major*, enquanto em Fortran, essa alocação é *column-major* [Agay and Vajhoejo 1998]. Por esse motivo, foi necessário que os índices de acesso a vetores n-dimensionais em C e OpenCL fossem invertidos.

Outro passo dado neste aspecto foi o acesso às posições de vetores n-dimensionais em forma de vetor unidimensional, o que requer um cálculo de acesso. Considerando a alocação em Fortran "real matrix(n1,n2,n3)" e o acesso feito da forma "matrix(i,j,k)", sendo i, j e k índices inteiros, o acesso a posição correspondente em C, tendo recebido a matriz *"matrix"* por referência, fica: *matrix[(k*n2*n1)+(j*n1)+i]*.

3.3. Preparação e Execução

Sendo o OpenCL um framework compatível com várias fabricantes de GPUs presentes no mercado e a grande diversidade de diferentes modelos desses fabricantes, foi necessário fazer a identificação do dispositivo no qual será executado o *kernel*. Esse processo envolve a utilização das chamadas que identificam e alocam um ou mais dispositivos, contextos e filas de comando.

Além disso foi necessária a alocação de buffers na memória da GPU. Esta alocação é necessária para que os dados possam ser lidos e alterados durante a execução do *kernel*. Esses *buffers* também são necessários para as chamadas OpenCL responsáveis pela definição dos parâmetros de entrada do *kernel* para só então estarem realmente disponíveis para a GPU. O código fonte do *kernel* é compilado através de uma chamada OpenCL, em tempo de execução e alocado em um programa. Este programa é utilizado para a criação da fila de comando a ser executada em GPU. Após a execução foi feita a leitura dos *buffers* de saída para coletar os dados processados.

4. Resultados

O código foi instrumentado a fim de medir tempos envolvidos nas diferentes partes do código, como apresentados na tabela 1.

O computador utilizado para os testes contava com um processador Intel Xeon E5620 2,4Ghz, 12 GB DDR3 de RAM e uma GPU Nvidia Tesla M2050 3 GB GDDR5. O sistema operacional instalado é o Debian 6 (Squeeze), com OpenCL 1.2, icpc 12.1.4, ifort 12.1.3, mpif90 4.7.2 e mpicc 4.6.3. A tabela 1 apresenta os tempos médios obtidos em cada parte (média de 23 execuções, correspondentes a um período simulado de 24h). Os dados na tabela mostram que a sub-rotina adaptada para OpenCL gastou a maior parte do tempo em inicializações (identificação do dispositivo). Além disso, gastou-se um tempo significativo na criação (*clCreateBuffer*) e liberação de *buffers (clReleaseMemObject)* na memória do dispositivo. Com todos esses fatores de perda significativa de tempo, registrou-se que o tempo da chamada OpenCL, utilizada para fazer os mesmos cálculos realizados de forma serial anteriormente, foi de apenas 26% do tempo total da chamada.

O método utilizado para fazer os cálculos não foi o que aproveita da melhor forma os recursos oferecidos pela GPU e ainda pode ser aperfeiçoado para gerar resultados mais satisfatórios. A possível causa da demora para criação de buffers é o tamanho dos vetores, uni e multidimensionais. Tais vetores tem tamanhos variados, sendo os mais importantes os tridimensionais com tamanho 30x70x70. Consequentemente, a execução original, de forma serial, continuou sendo mais rápida (36,79 ms).

Tabela 1. Tempos de execução do programa paralelizado

Parte do programa	Tempo de execução (μs)*	Percentual do tempo total
Inicializações	78180,22	47,39
clCreateBuffer	21020,39	12,74
clCreateProgramWithSource	481,55	0,29
clSetKernelArg	2,11	0,0013
clCreateComandQueue (execução do kernel)	43202,66	26,19
clEnqueueReadBuffer	1967,05	1,19
clReleaseMemObject	16691,44	10,12
Total	164968,89	100

5. Conclusão

A utilização de OpenCL, da forma feita neste trabalho, mostrou-se ineficiente para a paralelização da sub-rotina escolhida. O tempo gasto em inicializações e manuseio da memória torna a operação lenta e ineficiente, a menos que a quantidade de cálculos seja suficiente para tornar os tempos de identificação e alocação irrelevantes. Como próximos passos, será feita uma tentativa de melhora da organização do paralelismo no *kernel*, visando ganho em tempo de execução. Ainda, é possível aproveitar a experiência de adaptação de código a outras rotinas, que em serial possuam tempos de execução mais elevados.

Referências

Agay, A. and Vajhoejo, A., editors (1998). *User Notes on Fortran Programming*, chapter Array Storage Order. Ibiblio.

CPTEC-INPE (2012). Modelo ccatt-brams / qualidade do ar. Disponível em: `http://meioambiente.cptec.inpe.br/modelo_cattbrams.php` Acesso em: 29 dez. 2012.

Freitas, S. R., Longo, K., Dias, M., Chatfield, R., Dias, P., Artaxo, P., Andreae, M., Grell, G., Rodrigues, L., Fazenda, A., and Panetta, J. (2007). The coupled aerosol and tracer transport model to the brazilian developments on the regional atmospheric modeling system (CATT-BRAMS). part 1: Model description and evaluation. *Atmos. Chem. Phys. Discuss.*, 7:8525–8569.

Khronos Group (2012). The OpenCL specification – version 1.2. Disponível em: `http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf` Acesso em: 30 dez. 2012.