MAC 0219/5742 – Introdução à Computação Concorrente, Paralela e Distribuída

Prof. Dr. Alfredo Goldman

MiniEP4 - Branch Divergence `versão 2.0`

Monitores: Giuliano Belinassi e Matheus Tavares

1 Introdução

Controles de fluxo como `for`, `while`, `if` e `if-else` estão sempre associados à uma condição. No momento em que o código sendo executado atinge um destes controles, a avaliação da condição determina o caminho (ou `branch`) a ser seguido.

Nas GPUs atuais da NVIDIA, conjuntos de 32 threads formam o que chamamos de `warps`. As threads de uma mesma `warp` executam código de forma SIMT (Single instruction, multiple threads) [1]. Isso significa que todas as 32 threads executam instrução por instrução de forma simultânea.

A pergunta é, como as `warps` lidam com controles de fluxo? Isto é, se os dados são diferentes, pode ser que uma dada condição (em um certo `if`, por exemplo) seja avaliada como verdadeira em uma thread e falsa em outra. Quando isso acontece temos o efeito chamado de `branch divergence`. Nestes casos, ambas as threads vão executar tanto o código do `if` quanto o do `else`, aproveitando apenas o que lhe for útil. Na prática, pode ser que uma esteja apenas executando uma série de *NOOP*s, mas o fato é que ela perde tempo que poderia estar gastando em instruções relevantes.

Sabendo da arquitetura de `warps`, o programador CUDA deve sempre buscar organizar seu código, dados e lançamento de kernels de forma a evitar `branch divergence`. Neste miniEP lhe é fornecido um código que não toma estes cuidados. Sua tarefa é bolar uma estratégia de refatoração do código de modo a evitar divergências e aumentar a performance.

[1] `https://en.wikipedia.org/wiki/Single_instruction,_multiple_threads`

2 `Software` fornecido

2.1 Compilar e Rodar

Foi fornecido no PACA o código `divergence`. Este código preenche aleatóriamente um vetor de `doubles`, envia-o para a GPU e aplica funções custosas sobre cada elemento.

Compile o código com:

```
$ make
```

E execute com:

```
$ ./divergence
```

O programa ira executar o trabalho na GPU em duas versões. No arquivo `gpu_work_v1.cu` está a versão de referência (onde temos problemas de `branch divergence`). No Arquivo `gpu_work_v2.cu` você implementará uma versão que tenta reduzir este problema. Ao final da execução, o programa ira imprimir os tempos das duas versões e o speedup. Além disso, é informado se a v2 retornou os resultados corretos e teve o speedup esperado. Seu objetivo é passar nestes dois testes que, junto com o relatório, e código, irão compor a nota do miniEP.

Certifique-se de que você entendeu o código e execute-o uma vez antes de prosseguir.

2.2 Reduzindo `branch divergence`

Você é livre para modificar o arquivo `gpu_work_v2.cu` como quiser, desde que mantenha a função `launch_gpu_work_v2()` e sua assinatura. Você pode adicionar novas funções neste arquivo e escolher os parâmetros para lançar o kernel (número de blocos e threads por bloco). Porém, você não deve alterar nenhum outro arquivo e nem o que é impresso nas saídas `stdout` e `stderr`.

Dicas:

1. Veja a versão já implementada em `gpu_work_v1.cu` para referência.
2. Mesmo tendo passado no teste de speedup uma vez, rode o programa mais uma ou duas vezes. Isto é importante pois este teste pode ser bastante afetado pela execução concorrente de outros processos na GPU.

3. Idealmente, tente executar seu código quando a GPU não tiver outros processos em execução. Você pode usar o `nividia-smi` para ver o uso atual da placa.

4. Note o comportamento das funções `laborious_func_le_half` e `laborious_func_gt_half`. É possível dizer, dado um valor inicial de `d_arr[i]`, se a execução ocilará muito entre os trechos `if` e `else`, ou se ela passará mais tempo em algum destes dois? É possível pensar em alguma estruturação dos dados que maximize a quantidade de threads de uma mesma `warp` executando o mesmo bloco (`if` ou `else`)?

5. Se quiser saber um pouco mais sobre `branch divergence` e algumas técnicas propostas para mitigar seus efeitos, recomendamos o seguinte paper: `https://www.researchgate.net/publication/220939034_Reducing_branch_divergence_in_GPU_programs`. **Não é garantido** que as técnicas deste paper sejam aplicáveis no problema deste miniEP, ficando a sugestão apenas como referência para aprofundamento do tema. Mas fique a vontade para aplicá-las em sua solução, caso julgue possível.

Nota: Não basta que o speedup esteja dentro do esperado. É preciso que as técnicas empregadas visem reduzir divergencia.

3 Relatório

Você deverá produzir um relatório em formato `.txt` ou `.pdf` contendo os seguintes itens:

1. Explique a(s) técnina(s) que você empregou para reduzir `branch divergence`. Deixe bem claro como o `speedup` alcançado foi resultado da redução de divergencia e não de outras otimizações. Inclua referências, caso tenha usado técnicas de fontes externas.

2. Comente se a técnica empregada traria o mesmo *speedup* se aplicada em um código paralelizado em CPU (e porque).

Seu relatório pode conter imagens, gráficos e/ou tabelas.

4 Entrega

Deverá ser entregue um pacote no PACA com uma pasta com o nome e o sobrenome do estudante que o submeteu no seguinte formato: `nome_sobrenome.zip`. Essa pasta deve ser comprimida em formato `ZIP` e deve conter:

- O relatório em formato `.txt` ou `.pdf`. Arquivos em formato `.doc`, `.docx` ou `.odt` não serão aceitos.
- O código `gpu_work_v2.cu` com suas alterações. (Não é necessário entregar outros arquivos de código)
- (opcional) Quaisquer imagens que você queira anexar

Em caso de dúvidas, use o fórum de discussão do Paca. A data de entrega deste Exercício Programa é até às **23:55h do dia 01 de julho**.

Boa Sorte!