MAC0219 Introdução à Computação Concorrente, Paralela e Distribuída

3o. Exercício Programa - Balanceamento de Carga em Aplicações Híbridas

Integrantes

Caio Henrique Silva Ramos 9292991

Julio Kenji Ueda 9298281

Código

Compilação

$ **make**

Execução

$ **mpirun ./main <N> <k> <M>**

Descrição dos módulos

main.cu: responsável pela execução do programa.

calculus.cu: responsável pelos cálculos comuns a todas as outras implementações.

seq_calculus.cu: responsável pelo cálculo da integral por implementação sequencial (CPU com uma thread apenas).

omp_calculus.cu: responsável pelo cálculo da integral por implementação com OpenMP (CPU com várias threads).

gpu_calculus.cu: responsável pelo cálculo da integral por implementação com CUDA (CPU com uma thread e GPU).

load_balance.cu: responsável pelo balanceamento de carga.

Métodos e Soluções

Integração de Monte Carlo

Suponha por simplicidade que seja possível alocar um vetor de tamanho N de dupla precisão (*double*) na memória. Para dados N, k e M seguimos os seguintes passos:

1. criamos um vetor de tamanho N com $x \in \mathbb{R}$ aleatórios no intervalo $]0, 0.5[$.
2. calculamos $f(x)$ *inplace* com k e M para cada x do vetor em (1).
3. calculamos $\langle f \rangle$ com o vetor de (2).

4. calculamos $f(x)^2$ *inplace* para cada $f(x)$ do vetor de (2).
5. calculamos $\langle f^2 \rangle$ com o vetor de (4).
6. calculamos a integral de Monte Carlo com $\langle f \rangle$, $\langle f^2 \rangle$ e N .

Levando em conta que não é possível alocar um vetor de tamanho *N* na memória, a solução é dividir o problema de tamanho N em k iterações, guardar a soma parcial de todos os elementos do vetor de (2) e de (4) até acabar as iterações e calcular $\langle f \rangle$ e $\langle f^2 \rangle$.

Balanceamento de Carga

Dois processos são responsáveis para realização do Balanceamento de Carga. O processo 0 (master) é encarregado de enviar o trabalho para a GPU, e o processo 1 é encarregado de distribuir a carga usando OpenMP. Um passo de treinamento é feito para realizar este balanceamento.

Dado um número de amostras pequeno x_0 e um número de amostras grande x, cada processo calcula o tempo de execução para o cálculo da integral dado esses valores.

Obtemos assim gpu_y0 e gpu_y como os tempos de execução em GPU, omp_y0 e omp_y como os tempos de execução em OpenMP.

Com esses dados, é possível calcular a taxa de crescimento do tempo de execução em função do número de amostras para cada processo (supondo crescimento linear).

Seja $g(x)$ e $o(x)$ as funções lineares que calcula o tempo de execução na GPU e em OpenMP respectivamente. Então o número de amostras que a GPU deve executar dado N é:

$$gpu_N = \frac{N * o'(x)}{o'(x) + g'(x)}$$

E o número de amostras para o processo com OpenMP é definido como:

$$omp_N = N - gpu_N$$

Desse modo, o processo mais rápido recebe mais carga de modo que ambos terminem o cálculo no mesmo instante.

Por exemplo, $g'(x) = 1$ e $o'(x) = 2$ significa que a taxa de crescimento do tempo de execução do processo com OpenMP é duas vezes mais rápido do que o processo com GPU, ou seja, quanto maior o tempo de execução mais demorado é o processo do cálculo. Desse modo, para um $N = 3000$, o balanceamento de carga define $gpu_N = 2000$ e $omp_N = 1000$.

Resultados e Conclusões

O sistema utilizado é: Intel Core i7-4700MQ (4 cores, 8 threads, 32KB L1, 256KB L2, 6MB L3) @ 2.4GHz, 8GB RAM e NVIDIA GeForce GTX765M @850MHz e memória de 2GB @ 4GHz e OS Linux Mint 18.3.

```
$ mpirun ./main 300000000 20000 10000
treinando ...
gpu_N: 276240005, omp_N: 23759995
--------------------------------------------------------------------------------
Tempo com balanceamento de carga em segundos: 8.934994
Erro no calculo com a soma: 0.006852
Erro no calculo com a subtracao: 0.004681
--------------------------------------------------------------------------------
Tempo na GPU com uma thread na CPU em segundos: 8.352549
Erro no calculo com a soma: 0.007285
Erro no calculo com a subtracao: 0.004277
--------------------------------------------------------------------------------
Tempo na CPU com 8 threads em segundos: 92.407664
Erro no calculo com a soma: 0.010369
Erro no calculo com a subtracao: 0.001173
--------------------------------------------------------------------------------
Tempo sequencial em segundos: 102.924209
Erro no calculo com a soma: 0.005697
Erro no calculo com a subtracao: 0.005838
--------------------------------------------------------------------------------
```

Predições do Balanceamento de Carga na prática

Do ponto de vista teórico, é fácil observar que após a distribuição de carga:

1. Se o tempo de execução na GPU for igual o tempo de execução em OpenMP, então o tempo de execução é o menor possível.
2. O ganho de tempo com a divisão de carga deve compensar o tempo de transferência de dados entre CPU e GPU e o tempo de comunicação entre processos MPI.

Porém, não é isso que acontece na prática:

1. O tempo de execução com OpenMP possui muita flutuação (alta variância) devido ao escalonamento do S.O., de modo que para calcular a média é necessário um grande número de treino. Isso torna o passo de treino muito lento.
2. A distribuição de carga é definida através da média dos treinos, mas sempre possui uma margem de erro razoável considerável devido à alta variância de (1).
3. Como os dois processos não terminam de forma sincronizada devido ao erro de (2), o pequeno atraso causado muitas vezes não compensa o balanceamento de carga.

Processos competindo por recursos na CPU muitas vezes possuem desempenho pior do que processos isolados.

1. O tempo gasto para resolver o problema de tamanho N na GPU usando apenas uma thread na CPU muitas vezes possui desempenho melhor do que com Balanceamento de Carga, que precisa competir o uso de uma thread da CPU com o processo com OpenMP.
2. Da mesma forma, o tempo sequencial usando apenas uma CPU e uma thread muitas vezes possui desempenho melhor do que a versão com OpenMP.

CUDA versus OpenMP

A implementação em CUDA é visivelmente mais rápida do que as versões em CPU. Isso se deve ao fato de que o problema é facilmente paralelizável.

Multithread versus Single Thread

O ganho por usar múltiplas threads não implica em performance quando o tamanho do problema N é pequeno.

Conclusões

- Teoria é diferente da prática na maioria das vezes;
- É recomendado solucionar problemas SIMD como a Integração de Monte Carlo utilizando GPUs;
- Tentar paralelizar tudo muitas vezes não compensa;
- Balanceamento de Carga em sistemas distribuídos é difícil.