# Paralelização do algoritmo de Barnes Hut

Daniel de Almeida Cruz da Cunha, 2020/2

**Solução**

A paralelização do algoritmo de Barnes Hut foi realizada utilizando **GPROF** e **OpenMP**. O GPROF foi utilizado para detectar quais segmentos do código tinham maior impacto no tempo de execução, e o OpenMP para implementar o paralelismo em si.

A escolha pelo OpenMP foi tomada pela simplicidade da tecnologia e pelos materiais disponibilizados durante o curso. Uma dificuldade que tive foi definir onde e como usaria os pragmas, porém isso foi facilmente solucionado utilizando profiling.

Após realizado o profiling, foi detectado que o trecho mais impactante do código era a subrotina *update_forces*. Esta rotina implementa um laço responsável por realizar os cálculos do algoritmo, e foi nela que adicionei apenas um pragma para melhorar a performance.

O pragma em questão foi o *num_threads*, utilizado para paralelizar determinado segmento em N threads. Isso permitiu que a execução do laço fosse separada em até 16 threads, de acordo com o input, o que mostrou uma excelente melhoria de desempenho.

**Resultados**

O algoritmo foi adaptado para trabalhar com 1, 2, 4, 8 ou 16 threads. Os primeiros testes (*output-1.txt*), utilizando apenas uma thread, levaram cerca de 210 segundos para executar. Após realizado o profiling, seu relatório (*report.txt*) apontou que mais de 94% do tempo de execução era realizado em duas funções, *update_forces_help* e *calculate_force*.

A função que foi otimizada utilizando o OpenMP foi a *update_forces*, a qual possui um laço que chama o *update_forces_help*. Nele foi utilizado um pragma para habilitar o paralelismo no laço, permitindo até 16 threads. Opcionalmente eu poderia ter colocado para o algoritmo inteiro, porém optei por limitar apenas para o laço principal. Segue a tabela de tempo de execução de acordo com o número de threads:

| **Threads** | **Tempo de execução (s)** |
|---|---|
| 1 | 215,46 |
| 2 | 120,43 |
| 4 | 66,57 |
| 8 | 42,565 |
| 16 | 36,342 |

Percebesse que utilizando apenas duas threads o tempo de execução diminui cerca de 45%, e com 16 threads chegou a 84% de redução. Os tempos de cada thread pode ser encontrado na pasta *data* do projeto anexado.