Método de Monte Carlo em OpenMP

elc139 – Programação Paralela

Filipe Simões e João Vitor Machado de Mello

Estratégia de Paralelização nº 1

Estratégia de Paralelização nº 1

- Paralelizar os cálculos de percentual queimado e probabilidade

```
#pragma omp parallel private(ip, it)
{
  Forest* forest = new Forest(forest_size);
    #pragma omp for
    for (ip = 0; ip < n_probs; ip++) {
      prob_spread[ip] = prob_min + (double) ip * prob_step;
      percent_burned[ip] = 0.0;
      rand.setSeed(base_seed+ip); // nova seqüência de números aleatórios
      // executa vários experimentos
      for (it = 0; it < n_trials; it++) {
        // queima floresta até o fogo apagar
        forest->burnUntilOut(forest->centralTree(), prob_spread[ip], rand);
        percent_burned[ip] += forest->getPercentBurned();
      }

      // calcula média dos percentuais de árvores queimadas
      percent_burned[ip] /= n_trials;

      // mostra resultado para esta probabilidade
      printf("%lf, %lf\n", prob_spread[ip], percent_burned[ip]);
    }
}
```

Modificações

- Foi feita uma declaração dos contadores (ip e it) antes dos laços de repetição do programa para deixá-los privados para cada thread
- A declaração de new Florest(forest_size) foi alterada para dentro do primeiro laço de repetição tornando-o compartilhável.

Estratégia de Paralelização nº 2

```
//Opção 2: paralelizar calculo de percentual de forma dinâmica com schedule
#pragma omp parallel private(ip, it)
{
    Forest* forest = new Forest(forest_size);
        #pragma omp for schedule(dynamic)
        for (ip = 0; ip < n_probs; ip++) {
           prob_spread[ip] = prob_min + (double) ip * prob_step;
           percent_burned[ip] = 0.0;
           rand.setSeed(base_seed+ip); // nova seqüência de números aleatórios
           // executa vários experimentos
           for (it = 0; it < n_trials; it++) {
              // queima floresta até o fogo apagar
              forest->burnUntilOut(forest->centralTree(), prob_spread[ip], rand);
              percent_burned[ip] += forest->getPercentBurned();
           }

           // calcula média dos percentuais de árvores queimadas
           percent_burned[ip] /= n_trials;

           // mostra resultado para esta probabilidade
           printf("Thread %d: %lf, %lf\n", omp_get_thread_num()+1, prob_spread[ip], percent_burned[ip]);
        }
    }
```

- Foi feita uma declaração dos contadores (ip e it) antes dos laços de repetição do programa para deixá-los privados para cada thread
- A declaração de new Florest(forest_size) foi alterada para dentro do primeiro laço de repetição tornando-o compartilhável.
- Foi usado schedule Dinâmico para esta solução, deixando o particionamento na mão do OpenMP

Experimentos

- Em ambas as soluções, foram feitos experimentos com X, Y e Z de tamanho de vetor
- Os números de probabilidades não foram alterados

Resultados Obtidos