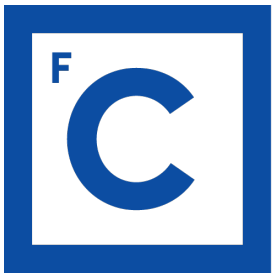

Ciências ULisboa

Faculdade de Ciências da Universidade de Lisboa

Programação Paralela e Concorrente

José Eduardo Madeira
fc51720

Assignment 2

Este relatório foi escrito com base em testes executados numa máquina com 4 cores e 8 threads.

A maneira de como paralelizei o problema, foi usando a Framework forkjoin do Java, uma implementação do algoritmo Divide and Conquer. Sendo esta uma forma eficaz de abordar problemas recursivos, pois permite dividir uma tarefa em tarefas menores para serem processadas usando threads disponíveis. Com isto quero dizer que uma thread que tenha realizado todas as subtarefas na sua "queue de trabalho", vai "roubar" subtarefas livres de outras threads, o que permite realizar tarefas mais rapidamente, assim fazendo uso do algoritmo de Work Stealing. Dito isto criei uma classe ForkJoinCoin que dá extends á super classe RecursiveTask<Integer>, onde dentro do método compute() temos os mesmos casos base que a implementação sequencial. De seguida ainda no compute() tratei do controlo de granularidade com o método Surplus, mudando a visibilidade do método sequencial da class Coin para protected para que possa ser chamado nesta nova classe (ForkJoinCoin), por fim utilizo o fork join. Em relação à classe Coin, no método paralelo é criado um objeto da class ForkJoinCoin e um ForkJoinPool, onde o objeto da class ForkJoinCoin será "invocado" pelo ForkJoinPool.

Uma implementação da solução em paralelo para este problema, sem controlo de granularidade, demoraria em media 6 -7 vezes mais tempo do que a implementação sequencial e por sua vez demoraria mais ainda que uma implementação com controlo, pois a divisão de trabalho pelas threads não é uniforme, dentro das técnicas de controlo de granularidade aprendidas nas aulas teóricas, o método Surplus, revelou-se ser o mais rápido e eficiente, pois melhora em média 4-5 vezes o tempo do que a resolução sequencial.