RELATÓRIO DO SEGUNDO EXERCÍCIO DE PROGRAMAÇÃO PARALELA E CONCORRENTE

Relatório por
Guilherme Lopes – 52761

- Qual foi a estratégia de paralelização? A estratégia utilizada foi o ForkJoin com o objetivo de dividir entre cada Thread disponível, em paralelo, certas Tasks definidas. Para esta resolução em concreto do problema, ForkJoin seria a melhor maneira de manter a lógica e alcançar paralelismo eficiente por se obter o processo "dividir para conquistar" para dividir Tasks apenas de forma eficaz e quando necessário. No caso, ao observar a resolução sequencial consegui perceber que poderia definir as Tasks a partir dos pedidos recursivos onde criaria instâncias da classe Coin obtendo algo semelhante à lógica recursiva da resolução sequencial. Através disso ocorre os processos e "work stealing" dessas Tasks entre Threads o que permite um processamento mais equilibrado e eficiente entre cada Thread que faz as Threads nunca ficarem sem trabalho na sua fila de trabalho, resultando num programa mais eficiente em geral por conseguir lidar com problemas mais imprevisíveis e irregulares sobretudo ao percorrer espaços de exemplos desbalanceados e irregulares.

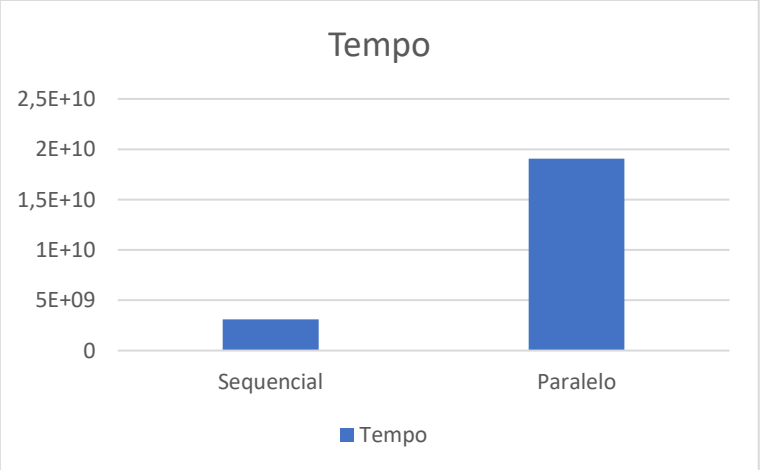

Figura 1 - Tempo de processamento do modo Sequencial e modo Paralelo sem Granularidade

- Como foi resolvido o problema de granularidade? Foi aplicado o mecanismo de controlo de granularidade Surplus, que, no caso da minha resolução, muda o processamento dos dados para sequencial após uma das Threads estar com mais duas Tasks na sua fila de trabalho do que o número de Tasks médio nas restantes Threads, por exemplo se a maioria das Threads tiverem todas duas Tasks na fila e uma das Threads tiver cinco Tasks na fila, o programa passa a correr em modo sequencial. Foi utilizado Surplus por duas razões principais, primeiro porque o problema é irregular e o trabalho por cada Task pode variar em complexidade (cada Task pode ser mais custosa, demorada e trabalhosa que outra) o que faria o mecanismo MaxLevel ser menos útil, caso o problema fosse regular, provavelmente a média de Tasks seria sempre consistente entre cada Thread e o mecanismo MaxLevel seria o mais útil, e segundo porque o mecanismo MaxTasks demonstrou-se ser bastante pouco eficiente ao ponto de bloquear completamente o meu computador quando o programa era corrido durante alguns segundos e sem apresentar qualquer resultado (este problema manteve-se mesmo após mudar os valores na "configuração" do mecanismo). O mecanismo Surplus apresentou uma melhoria imensa na velocidade do programa (estabiliza-se em cerca de cinco vezes mais rápido, que o modo sequencial, após algumas iterações), que antes seria até mais lento que a resolução sequencial (Figura 1) devido à quantidade extrema de Tasks criadas e que sobrecarregavam a fila de trabalho de cada Thread. Após modificar alguns valores neste mecanismo, por tentativa e erro, o valor que apresentou melhores resultados foi o que impossibilita cada Thread ter mais de duas Tasks acima da média de Tasks, ou seja, Surplus(2). Surplus(X) corresponde a "if(RecursiveTask.getSurplusQueuedTaskCount() > X)", sendo X o número de Tasks a cima da média de Tasks que cada Thread suporta antes de processar o resto do problema em modo sequencial.

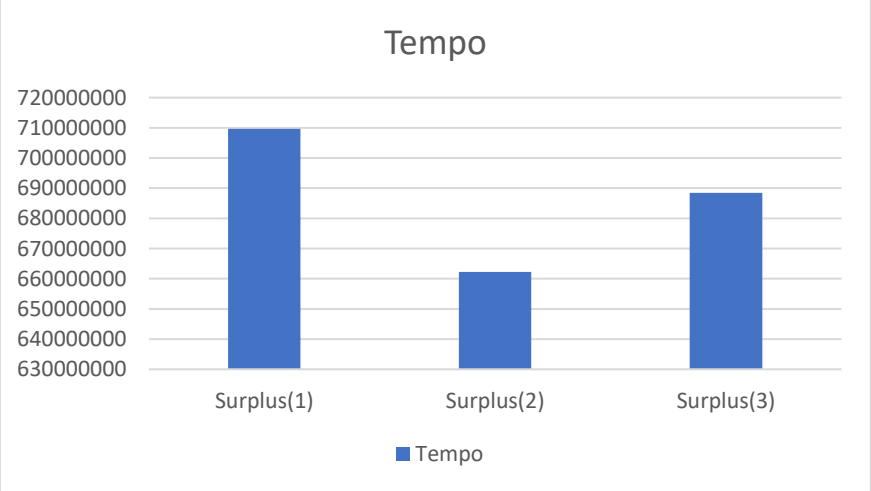

Figura 2 - Tempo de processamento do modo Paralelo com o mecanismo de Granularidade Surplus para certo número X

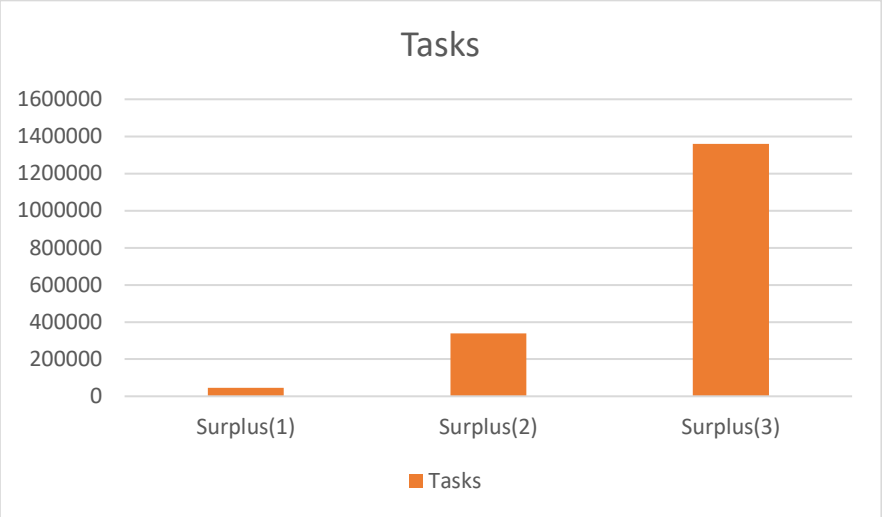

Figura 3 - Número de Tasks criadas no modo Paralelo com o mecanismo de Granularidade Surplus para certo número X

Mesmo que o mecanismo Surplus aplicado a X = 2 tenha um maior número de Tasks criadas, a velocidade média do programa é maior que em X = 1, por isso o programa continua mais eficiente ao criar uma maior quantidade de Tasks do que a correr em modo sequencial. Tendo X = 3 continua a ser menos eficiente que X = 2 devido a ser mais lento e criar mais Tasks.

O resolução sem mecanismo de granularidade tem tendência a criar mais de um milhar de milhão de Tasks, que é muito superior a qualquer resultado com mecanismo de Granularidade e possivelmente é a origem do problema de velocidade em comparação ao modo sequencial.

NOTA: Todos os gráficos foram criados com a média de valores ao longo de 40 iterações com listas de 30 moedas. Todos os tempos apresentados estão em nanossegundos. Testes feitos em uma máquina com 8 núcleos.