Relatório Computação Paralela (Junho 2021)

Carlos Freitas 47353, António Morais 47774 e André António 50118

Abstract—Este relatório visa a tirar conclusões da paralelização feita pela nossa equipa. Foi dado o código sequencial do cálculo da energia no embatimento de partículas com uma superfície e elaboramos uma versão paralela.

Index Terms—Computação Paralela; OMP; Threads; Parallel Programming

1 Introduction

Este trabalho visa compreender melhor o uso da computação paralela e de que forma a mesma pode ser explorada, aumentando, assim, o desempenho dos programas. É uma abordagem natural nos dias de hoje visto que qualquer computador tem vários processadores. É nos proposto o uso destes mecanimos, em particular, usamos o OMP para a exploração e concretização de uma versão paralela. O código que escrevemos para a solução da maior parte das disciplinas é sequencial e a maior parte do que fazemos e a maneira como pensamos na programação tende a ser sequencial. A computação paralela é de facto uma ferramenta essencial e este trabalho fez-nos compreender melhor os desafios que esta impõe. A computação paralela tem desafios novos onde é preciso pensar nos problemas com uma perspectiva diferente, certos algoritmos precisam de uma solução totalmente nova para ser possível paralelizar e outros por outro lado são muito fáceis de adaptar uma solução.

Com estre trabalho, foi pedido para paralelizar uma solução já implementada de um algoritmo sequencial que calcula o impacto de partículas numa superfície, foi nos possível analisar o problema e identificar as dependências que a solução continha. Depois da análise profunda do algoritmo foi relativamente fácil chegar a uma solução para o particionamento do trabalho pelos diferentes processadores. Numa primeira fase, não tivemos em conta certos problemas e fez-nos levar algum tempo até perceber o porque do resultado não determinista que obtivemos. Mas após uma análise mais cuidada do código descobrimos o problema na divisão do trabalho e prosseguimos para uma implementação alternativa. Ponderamos os prós e os contras de cada solução e como devemos "atacar" o problema. Neste relatório fazemos uma análise da solução implementada e dos desafios que tivemos ao longo da implementação.

2 Análise da Solução

2.1 Uma primeira tentativa

Numa primeira fase tentamos paralelizar o programa dividindo o cálculo das partículas pelos diferentes processadores. Cada processador tinha uma parte das diferentes partículas a calcular. Era utilizado um vetor para os guardar os cálculos das diferentes partículas e no final era feita a soma destes vetores para calcular a energia total das partículas distribuídas pela superficie. Esta seria uma solução ideal se o número de partículas a calcular fosse elevado e número de partições da superfície relativamente pequeno. Quanto maior o número de partículas face ao particionamento da superficie melhor seria esta solução. Após a implementação descubrimos que os resultados da solução eram não deterministas. Após análise do código e suscetíveis erros na implementação da paralelização dividindo as partículas pelas threads, descubrimos que o que fazia com que o resultado final fosse diferente em todas as iterações foi o facto da soma de floats não ser comutativa. A soma de floats pode causar a acumulação de erro, pois só estamos a representar um número com finitas casas decimais. Esta aproximação nos cálculos intermédios e na junção dos vetores faz com que o resultado final não seja o pretendido. Após muita reflexão de como deveríamos paralelizar o problema, começamos com uma solução errada, mas que nos fez perceber de facto como lidar com cálculos deste tipo.

2.2 Uma segunda tentativa

Após uma primeira abordagem na paralelização do problema, chegamos à conclusão que era preciso repensar na forma como iriamos proceder. Com os resultados obtidos na primeira tentativa concluímos que as somas de cada partição da superfície tem de ser feita em sequencia, pela mesma ordem que o programa sequencial, para obter exatamente os mesmos resultados. Sendo assim esta segunda abordagem levou-nos a repartir o problema pelos processadores de uma nova forma que calcula exatamente pela mesma sequência todos os embatimentos das partículas. Concluimos que o cálculo de cada partição da superfície é independente, a fórmula apenas depende da localização da mesma, da posição de embate da partícula e da sua energia. Sendo assim cada thread ficaria com uma parte da superfície dividida de igual forma. Cada uma executa o cálculo para todas as partículas, mas apenas numa partição da superfície. Isto faz com que esta solução tire partido de

superfícies com um elevado número de partições, sendo que perde eficiência quando o número de partições da superfície é menor.
Quanto maior a superficie, melhor se torna esta solução face à solução sequencial.

3 Estudo do Desempenho

A partir dos resultados obtidos, confirmamos a hipótese de que a solução paralela obtem melhores resultados face ao algoritmo sequencial, quando o tamanho da superfície é maior.

Para os testes efectuados numa superfície de tamanho 20, (teste03 para as varias threads) a solução paralela tende a piorar com cada thread adicional. Ou seja, os cálculos são executados mais rápidos na versão sequencial. Obtendo em média um resultado na volta dos 0.000005 segundos, usando threads o tempo passa para 0.000035, 0.000234, 0.000377, 0.014892, com 1, 2, 4, 8 threads respetivamente. Podemos observar que com o uso de uma única thread o resultado é pior, era esperado que o tempo de execução aumente, neste caso passa para 7 vezes mais lenta, isto é devido ao tempo adicional das preparações e do código corrido para usar as threads do omp, sendo que a solução sequencial é muito rápida devido ao teste ser pequeno, leva a que este tempo usado para o omp não seja diluído no tempo dos cálculos principais do algoritmo. Nos casos que a solução demore mais a fazer os cálculos já não irá ser tão pronunciada a diferença entre usar a versão sequencial e usar a versão omp com apenas 1 thread.

Já os testes para uma superfície de tamanho 100M, (testes_08) obtivemos um resultado cada vez melhor com o aumento de threads. Os resultados dos "speedups" obtidos foram 0.77, 1.35, 2.08, 2.48, 2.77, para 2, 4, 8, 16 threads respetivamente. Neste teste é usada apenas uma partícula, sendo que o tempo do cálculo na versão sequencial é à volta de 20 segundos, não sendo este tempo muito elevado. Significando que há poucos cálculos a serem feitos em compração por exemplo com o teste 2 que corre sequenciamente em 116 segundos, leva a que não haja tanta vantagem entre as diferentes threads. Claro que observamos um "speedup" significativo sendo que usada 16 threads temos um speedup de 2.77 face ao programa sequencial, contudo é um valor idêntico ao usar 8 threads com um "speedup" de 2.48. O estudo da eficiência irá ser feito de seguida, mas podemos observar que de facto, obtemos resultados melhores não só com uma maior partição da superfície, mas também com maior carga de calculos, isto é, com mais partículas a serem calculadas.

Usando o teste 2 que é efectuado com 30 000 partições da superfície e 20 000 particulas podemos de facto observar os resultados a serem cada vez melhores, neste caso a versão paralela obtem um "speedup" significativo face à versão sequencial. O primeiro motivo é o tempo que este teste demora a correr sequencialmente, 116 segundos, isto faz com que haja uma maior margem para os cálculos efectuados em paralelo. Os tempos de utilização de threads e o tempo que o programa corre sequenciamente (preparação dos dados e inicializações) são diluídos na parte paralela do programa. Correndo o teste 2 observamos os seguintes "speedups", 1.797, 3.702, 7.106, 9.49, 21.68, 34.78, para 2, 4, 8, 16, 32, 64 threads respetivamente. Podemos observar os melhores resultados em termos de "speedup" em comparação ao teste sequencial, mais uma vez observamos que a utilização de mais threads melhora o desempenho, que desta vez é verdadeiramente significativo. Este resultado é justificado pelo número de cálculos que cada thread executa, (número de partículas * (partição da superfície/numero de threads)) sendo este muito maior, 600 milhões de cálculos, comparando com por exemplo o teste 8, com um total de 100 milhões de calculos.

O teste 7, tendo 5 000 particulas e partição da superfície 1 milhão apresenta resultados ainda melhores que os testes anteriores devido à quantidade de cálculos que contem, sendo este o melhor rácio entre a parte sequencial do programa e a parte paralela e também devido ao numero elevado na partição da superfície. Obtivemos os seguintes resultados no calculo do "speedup": 1.28, 3.93, 5.97, para 2, 4 e 8 threads respetivamente.

4 Estudo da Eficiencia

Realizando testes com uma partição da superficie relativamente pequena (valores: 20 e 35) a versão paralela tem um desempenho cada vez pior com o aumento do numero de threads, isto significa que a eficiência segue o mesmo gráfico, ficando cada vez pior com o aumento do numero de threads ao correr estes testes.

Quando usado testes significativamente maiores, com uma partição da superfície mais elevada como por exemplo o teste 3, usando 30 000 valores para o cálculo, observamos uma melhoria significativa no uso de threads. A eficiência para os diferentes números de threads foram: 0.89, 0.92, 0.88, 0.54, para 2, 4, e 8 threads respetivamente. Assim sendo é possível concluir que para este teste específico usar 4 threads é o mais eficiente. Embora o "speedup" seja continumanete maior, o custo de usar mais threads não melhora igualmente os tempos dos cálculos. Isto advém do programa não ser completamente paralelo, é necessário efectuar uma inicialização dos parâmetros e também de um "setup" para preparar a computação em paralelo. Após esta computação é necessária novamente realizar vários passos sequencialmente o que faz com que o tempo reduzido pela computação em paralelo seja apenas uma percentagem significativa do programa como um todo. Sendo esta uma percentagem faz com que seja possível melhorar os tempos apenas numa parte do programa, (parte paralela), sendo que este cada vez mais é diluída na parte sequencial do programa. Quanto mais threads usamos e mais rápida a solução, maior é a percentagem do programa que é corrida sequencialmente. Isto faz com que haja um "threshold" no quão mais rápida poderá ser a versão paralela.

No teste 8, usando 1 particula e uma partição da superfície com 100 milhões, obtemos os seguintes resultados no cálculo da eficiência, 0.675, 0.52, 0.31, 0.17,

0.031, para 2, 4, 8, 16 threads respetivamente. Observamos uma eficiência menor neste teste, devido ao mesmo motivo enunciado anteriormente, este teste tem um número de cálculos total menor relativamente ao teste 3, fazendo assim com que a parte paralela do algoritmo seja menor relativamente à percentagem total do mesmo. Isto faz com que usando mais threads a parte sequencial tenha um peso cada vez maior.

No teste 7, com 5 000 particulas e uma partição da superfície com o valor de 1 milhão, obtivemos os melhores resultados para a eficiência, isto devido a que a nossa solução se comporte melhor com números elevados para a superfície e também pelos cálculos totais serem maiores (5 mil milhões de cálculos), o que faz com que a parte sequencial do algoritmo se dilua na parte paralela. Os resultados obtidos para a eficiência foram: 0.64, 0.98, 0.75 para 2, 4 e 8 threads respetivamente.

5 Conclusões

Para concluir, após o estudo do algoritmo, verificamos as nossas hipóteses de como iria ser o comportamento do programa, tanto o seu desempenho com a eficiência do mesmo. Conseguimos perceber melhor como otimizá-lo e que partes são fulcrais para uma solução paralela eficaz. Tentamos dividir o problema de várias formas, sendo que a nossa primeira hipótese tinha um problema que não conseguimos ultrapassar. Após esta primeira análise e implementação, apartir dos resultados obtidos, verificamos que de facto, a soma de floats não poderia ser paralelizada, surgindo assim um interesse especial acerca das várias especifidades da computação paralela e dos desafios impostos pela mesma. Como esta situação deve haver inúmeras outras que precisam de um especial cuidado.

Para manter a ordem destas somas, foi implementada uma nova solução, dividindo o trabalho pelas threads de uma maneira diferente. Concluimos que, de facto, na paralelização dos algoritmos, o maior trabalho está na análise dos mesmos, é necessário perceber o código como um todo para a identificação das dependências e para criar uma solução paralela para a mesma.

Posemos ao longo da elaboração do projecto suposições do desempenho do algoritmo para diferentes testes e achamos interessante verificar se iria de encontro às nossas espectativas. O uso do cluster foi particularmente interessante, pois foi possível testar o nosso código com um maior número de threads.

Na análise dos testes, verificamos que só haveria um melhoramento no desempanho face à versão sequencial para superfícies com um valor mais elevado, pois para superfícies pequenas, o custo de usar threads é maior que o ganho da mesma.

Agredecimentos

Queremos agradecer ao grupo 15 pelo script em python que elaboraram. Tendo dado uma ajuda nos testes automáticos do trabalho, assim como a geração de ficheiros automática. Os membros do grupo são: João Antão, Frederico Aleixo e o Bruno Rodrigues.

Contribuições Individuais

Todas as partes foram divididas pelos três membros do grupo sendo que houve certas partes que foram mais trabalhadas por uns membros, mas com a ajuda de todos. Codigo e relatório: Carlos Freitas, Testes: António Morais, contribuição geral: André António

A percentagem atribuída pelo trabalho foi: Carlos Freitas 40%, António Morais 40%, André António 20%

References

[1] Slides da cadeira de Computação Paralela

[2] McCool M., Arch M., Reinders J.; Structured Parallel Programming: Patterns for Efficient Computation; Morgan Kaufmann (2012); ISBN: 978-0-12-415993-8-history.html. 1986. (URL link *include year)

[3] Mattson T., Sanders B., Massingill B.; Patterns for Parallel Programming; Addison-Wesley(2004); ISBN: 0-321-22811-1R. Nicole, "The Last Word on Decision Theory," *J. Computer Vision,* submitted for publication. (Pending publication)

[4] Raynal M.; Concurrent Programming: Algorithms, Principles, and Foundations; Springer-Verlag Berlin Heidelberg (2013); ISBN: 978-3-642-32026-2D.S. Coming and O.G. Staadt, "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection," *IEEE Trans. Visualization and Computer Graphics*, vol. 14, no. 1, pp. 1-12, Jan/Feb 2008, doi:10.1109/TVCG.2007.70405. (IEEE Transactions)