Trabalho Prático de Computação Paralela

Parte 2

1st Carlos Silva
Mestrado em Engenharia Informática
Universidade do Minho
Braga, Portugal
PG57518

2nd Tiago Silva
Mestrado em Engenharia Informática
Universidade do Minho
Braga, Portugal
PG57614

***Abstract*—Esta fase de trabalho visa explorar o paralelismo de memória compartilhada (baseado em OpenMP) para melhorar o desempenho geral de tempo de execução.**

I. Introdução

O objetivo principal desta 2º Fase é a de melhorar a performance do nosso programa, utilizando técnicas de paralelização e consequentemente reduzindo o seu tempo de execução.

II. Mudanças em Relação à Primeira Entrega

Em relação a entrega anterior, tivemos um aumento do tamanho dos dados de SIZE=42 para SIZE=84, e um novo solver que permite uma abordagem especialmente projetada para execução paralela, aproveitando as vantagens do paralelismo de memória compartilhada

III. Metodologia

Para conquistar este objetivo, usamos uma metodologia que sabemos que dá frutos. Começamos por usar ferramentas de profiling, no nosso caso, *gprof*, para concluir qual seria o hotspot do nosso programa(as partes do código com maior tempo de execução). Em seguida, são analisadas e avaliadas as alternativas para explorar paralelismo nestes mesmos hot spots. Com base em análises de escalabilidade e eficiência, foram então implementas as optimizações

IV. Implementação/OpenMP

Com o uso do *gprof*, foi identificado o hotspot do programa, a lin_solve. Foram então aplicadas técnicas de parelelização para a diminuição do tempo de execução desta mesma . Através do OpenMP, foi usado **#pragma omp parallel for reduction(max:max_c) schedule(static) collapse(2)**, em cada um dos dois nested loops, presentes no red e no black, tendo ambos a mesma lógica. O ”parallel for” paraleliza a execução dos loops, aproveitando múltiplos threads para processar diferentes partes da malha tridimensional. Isto é importante para processar de maneira mais eficiente a grande carga de volume. O reduction usado serve para calcular o valor, neste caso de max_c, de forma segura e paralela pois cada thread mantêm o seu valor de max_c e no final as cópias locais são usadas para calcular o valor global máximo. Isto aqui garante que nunca acontecem *data races* para atualizar este valor. O uso de scheduling, neste caso, static, funciona muito melhor que o scheduling dynamic pois existem um workload muito parecido ou igual e a distribuição deste trabalho é feita de maneira mais uniforme pois este trabalho é de tamanho previsível(o uso de dynamic é para workloads diferentes/dificilmente previstos). A cláusula collapse combina os dois primeiros loops de índices i e j em um único ïoop.̈ Isso melhora a distribuição de trabalho entre as threads. Apenas pode ser usado com collapse(2) pois o 3º ciclo (k) precisa do valor dos dois primeiros,isto é, é dependente o resultados de ambos. Embora a lin_solve fosse a óbvia hotspot, foram aplicadas, seguindo a mesma lógica, paralelização nos outros for loops presentes no programa. Temos então presente um uso de todas as diretivas faladas anteriormente nos outros for loops, tirando o reduction. Mais uma vez, a utilização destas serve para, garantir que são processados em paralelo, para dividir de forma uniforme o workload das threads, e o collapse(3), que serve para combinar os 3 loops em apenas 1 loop ”conceptual”, pois nestes já não existe dependência entre nenhum deles. O reduction não foi usado pois não há atualização concorrente de variáveis compartilhadas, pois cada thread trabalha em uma parte independente do problema.

V. Análise de Escabalidade / SpeedUP

Testes feitos nas máquinas com propriedade c=24, que são uma pool de 6 máquinas muito parecidas com a da partition cpar

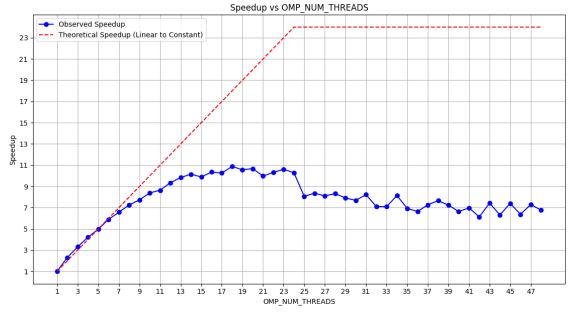

Fig. 1. Análise de SpeedUP

VI. ANEXOS

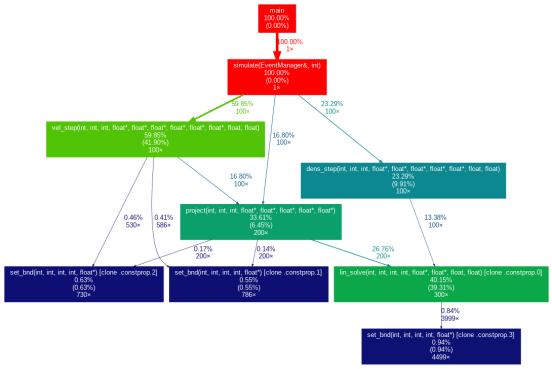

Fig. 2. Callgraph Inicial

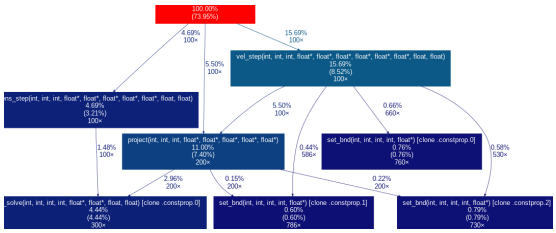

Fig. 3. CallGraph depois de Paralelizar lin_solve