Análise de multiplas implementações do paralelismo na decomposição de Cholesky

Tempos de execução

Threads	Tempo OMP	GFlops OMP	Tempo DF	GFlops DF
1	31.954131	7.604651	31.954412	7.604651
2	20.556654	11.820990	16.404920	14.812629
4	15.051850	16.144195	8.629030	28.160755
8	12.695412	19.140773	4.978011	48.814678
12	12.552987	19.357942	4.065391	59.772847
16	12.626203	19.245691	3.488394	69.659567

Speedup

Threads	OMP	DF
2	1.554442	1.947855
4	2.122937	3.703129
8	2.516982	6.419112
12	2.545540	7.860107
16	2.530779	9.160207

Funcionamento do programa

1. Ifs aninhados:

   O if externo determina se a computação será feita na mesma linha do elemento da diagonal principal sendo calculada ou não. No caso de estar na mesma linha que a diagonal principal, o primeiro if aninhado divide a computação em dois casos: no primeiro de ser o elemento da diagonal principal, caso no qual se calcula a decomposição de choleski do bloco utilizando-se de POTRF, o segundo de ser um elemento anterior à diagonal principal, caso no qual se atualiza o elemento da diagonal principal com os blocos anteriores com a função SYRK. Já no caso de não estar na mesma linha da diagonal principal, o segundo if aninhado ocorre a atualização das linhas abaixo da diagonal sendo calculada, dividindo em dois casos: o primeiro no qual se atualiza (m, k) atraves da função TRSM, e o segundo no qual o elemento (m, k) atraves da função GEMM, caso TRSM já tenha sido calculado para os elementos anteriores.

2. Laços:

O laço mais interno é o laço em n, n cresce até o valor de k, voltando para 0 e incrementando k e m quando isso acontece, n serve para iterar pelas colunas da matriz. Os laços de m e k são de maior complexidade, pois dependem da relação entre o numero de threads e o numero de blocos. O laço de m itera pelas linhas da matriz a partir de um determinado k, seu incremento é do numero de threads, exceto no caso em que ele ultrapassa o numero de blocos, no qual m e k sao atualizados para o proximo valor de k, assim m varia de k até o numero de blocos. O laço de k determina o atual elemento da diagonal principal que está sendo utilizado na computação e incrementa toda vez que m ultrapassa o numero de blocos. Dessa forma, k varia entre 0 e o numero de blocos, incrementando toda vez que m ultrapassa esse numero (só será maior que 1 em casos nos quais o numero de threads for maior que o de blocos).

3. Caso de poucas threads e muitos blocos:

   Nesse caso, next_m >= nBK ocorre raramente e, portanto, cada thread é atribuida uma linha da matriz e a calcula inteiramente, recebendo em seguida outra linha para o mesmo valor de k. Cada thread recebe uma linha consecutiva da computação, incrementando do numero de threads a cada fim de linha.

4. Caso com nThreads >= nBK:

   Nesse caso next_m >= nBK ocorre toda vez que os valores de n ultrapassam k, assim, mesmo que cada thread compute todos os valores da linha, apenas uma linha será computada para cada valor de k. Nesse caso provavelmente muitas threads ficarão presas por algum tempo nos loops de while(core_progress[...]), esperando as threads anteriores calcularem os elementos necessários para a continuação da computação.

5. Paralelismo por fluxo de dados:

   A matriz core_progress, compartilhada entre todas as threads, é responsavel pela correção do paralelismo do fluxo de dados. Ao se colocar os loops while(core_progress[...]) na execução de cada thread e atualizando o valor da matriz quando os elementos necessários são calculados nas funçÕes POTRF e TRSM, é garantido que cada thread espere que todas as suas dependencias estejam resolvidas.

Análise e Comentários

A codificação heroica do algoritmo mostra resultados muito surpreendentes, com speedups mais de 3 vezes melhores, mostrando que apesar de muito mais complexa e de dificil implementação, a paralelização especifica para cada problema pode se mostrar extremamente útil.

Na paralelização heroica é garantido que as threads estão o maximo do tempo

possível fazedo alguma computação. A alocação estática permite essa possibilidade, contrastando com a implementação simples de OpenMP, na qual as threads passam uma grande parte do tempo esperando barreiras computacionais.