Processamento Paralelo
MiniEp1

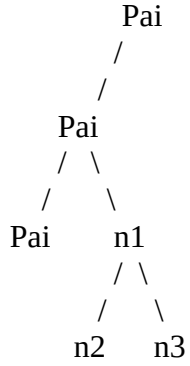
André Thiago Borghi Couto
2016103024

Relatório

O arquivo EP1Original, versão que foi disponibilizada como base para a construção do trabalho, um template de Objeto T que irá ser usado, consiste em criar uma LIST<double> de tamanho N, e preenchida por um valor aleatório, mas sempre o mesmo. Assim prosseguindo com o TIMENOW que representa o contador de tempo que será usado nas operações, junto com a função APLICAFUNCAOESOMA, nesta função temos a criação de um objeto T que será usado como retorno da FUNCAO, usamos um ITERATOR, para armazenar valores da FUNCAO no objeto T, para toda a lista T que foi passada para a função, após temos a soma de todos os valores em outro ITERATOR, utilizando uma variável double, e retornamos o resultado da soma. Após temos a contabilização do tempo e a exibição do resultado.

Em minha versão otimizada sequencial (EP1Sequencial) retirei o template de objeto T, substituindo-o por um LIST<double>, permanecendo o mesmo modelo de preenchimento da mesma lista a grande mudança está na função APLICAFUNCAOESOMA, nesta função temos a redução de repetições que eram inúteis, neste caso a absorção da soma na própria repetição de aplicação da FUNCAO, então retornando a soma, lembrando que havia um objeto temporário que não fazia parte do objetivo, era usado apenas para armazenar valores de resultado, criando assim um loop desnecessário. Permanecendo o restante igual.

Em minha versão paralelizada (EP1ParaleloList) utilizei a mesmas funções do sequencial, e utilizando o FORK para dividir em processos diferentes, neste caso tive problemas em como dividir um LIST<> então como podemos supor que independente do momento o valor aleatório vai ser verdadeiro, não existe problema em dividir tais listas em listas menores, fazendo o calculo e somando os resultados em um variável global com o compartilhamento de memória, sendo assim a divisão foi feita de acordo com a árvore de processos, no caso:



Assim, temos a subdivisão de cada processo, e em qual instante irá ser criado, então o resultado da execução de cada um é armazenada na SOMAGLOBAL, que é uma variável global que foi alocada com o MEMORYMAP, criando assim um espaço de memória comum a todos os processos derivados do PID original. Com isso é possível fazer com que os processos se comuniquem, que foi feito nesse código. Do mais o restante do código tem o mesmo procedimento dos anteriores.

Não fiquei satisfeito com o resultado da versão paralelizada pela utilização de Objetos LIST<>, então criei uma versão pensando em melhorar o desempenho com vetores simples (EP1Paralelo), que não acumulam bibliotecas, logo substitui os LIST<>'s por vetores. Nesse caso tive um aumento de desempenho, considerável, onde alocava vetores separados para cada processo e durante a execução utilizava a variável global para o acumulo da soma.

Resultados

Além do arquivo EP1Grafico.pdf, em testes com o arquivo run.sh, obtive e armazenei o tempo de execução total, com compilação normal em 2406.5s e compilação com -O3 em 1690.6s.

Por meio dos testes tivemos uma melhora no desempenho do paralelo com relação ao original de 5x com a compilação normal, já com a opção -O3 ativada tivemos um pouco mais de 5.2x de melhora no desempenho. Se levarmos em conta o código sequencial otimizado, a melhora ficou em quase 3x e 0.75 utilizando a compilação normal e com -O3 respectivamente.

O arquivo run.sh segue em anexo a os códigos.

Descrição da máquina

Os códigos foram implementados em C++ e compilados com GNU gcc versão 7.3.0 de dois modos, com a opção -O3 ativada e sem essa opção para efeitos de comparação. Os experimentos foram realizados em uma máquina Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz, com os caches de: L1d - 32K, L1i - 32K, L2 - 256K, L3 - 4096K. Tendo 8 GB de memória RAM e com S.O. Linux Mint 19.