Document Classification

CPD
Computação Paralela e Distribuida
Serial+OpenMP - 2012-13

1. Introducão

O objectivo deste projecto é com um conjunto de D documentos, cada documento classificado de acordo com S temas e como um número C de gabinetes, atribuir os documentos aos gabinetes baseando-se nos temas.

2. Descrição do algoritmo e abordagem utilizada para paralelização

Começamos por carregar o ficheiro em memória colocando os totais de documentos, temas e gabinetes em variaveis globais e usando duas estruturas, Document e Cabinet, que representam um documento e um gabinete, criamos dois vectores de ponteiros, um para o conjunto de documentos e outro para o conjunto de gabinetes, ambos estes vectores são globais e declaros *static volatile*.

A distribuição inicial dos documentos é feita automaticamente à medida que se lê o ficheiro, depois de ler o ficheiro todo podemos começar com o algoritmo principal:

1. Calcular para cada gabinete a média de cada tema baseado nas classificações dos temas dos documentos atribuidos ao gabinete;
2. Calcular as distâncias entre cada documento e todos os gabinetes e mover o documento para o gabinete com a menor distância;
3. Voltar ao 1 se algum documento foi mudado de gabinet

No passo 1, em cada gabinete colocamos os valores dos temas a 0 e depois para cada documento verificamos se pertence ao gabinete e se pertencer adicionamos o valor dos seu temas aos valores do gabinet, depois disto para cada gabinete dividimos os valores dos seus temas pelo número de documentos atribuidos a esse gabinete.

No passo 2, para cada documento calculamos a sua distância a cada gabinete calculando uma normal de um vector que tem como coordenadas os valores dos temas do documentos e do gabinete, e alteramos o gabinete a que o documento pertence se encontrar uma distancia mais curta.

Após termos terminado a versão de série, começamos por identificar as tarefas primitivas de entre as várias funções criadas. Depois efectuou-se a alteração do código de forma a permitir aplicar um maior paralelismo ao código e garantir a independência do acesso à memória no acesso às estruturas de dados. Com a análise inicial passou-se a uma análise mais aprofundada apoiada nos resultados obtidos.

3. Decomposição, Sincronização e Balanceamento de Carga

3.1. Loading Data

Na paralelização dos dados existem duas tarefas primitivas que se podem identificar: - leitura dos dados do ficheiro - conversão de strings para valores numéricos que podem ser int (id do documento) ou doubles (rate do subject)

Nesta parte verificou-se que havia problemas com a ordem de leitura tendo em conta o formato do ficheiro de entrada que não permite que haja muita paralelização. Esta observação surge por não ser vantajoso manter mais do que um file descriptor activo para efectuar leitura do mesmo ficheiro em zonas diferentes, pois o overhead acabava por cair sobre o próprio IO que é muito penalizante para o funcionamento em shared memory. Desta forma, verificou-se que, estando a correr ao nível da mesma máquina, não se tem vantagens em paralelizar as leituras de IO. No entanto conseguese ganhar alguma vantagem na paralelização das escritas para a memória durante as leituras dos dados, na medida que a criação dos documents não tem a restrição de seguir a ordem de leitura, tendo em conta a estrutura de dados adoptada. No caso dos subjects

consegue-se também algum paralelismo na medida que durante a leitura do ficheiro é possível estar a realizar as conversões a partir do buffer em paralelo. Analisou-se também a influência de se ter schedule aplicado no for interior de leitura dos vários subjects por linha, fazendo variar o valor de chunks associado a dynamic e guided. Neste caso verificou-se que guided se tornava mais eficiente, na medida que a master thread tem sempre bastante trabalho com a leitura do ficheiro e depois as escritas na memória eram efectuadas em paralelo. No entanto, o peso que tinha o controlo dos vários fork e joins constantes num ciclo muito interior, demonstrou não trazer vantagem, mesmo com o balanceamento de carga e manipulação dos valores de chunk. De acordo com os resultados apresentados e tendo em conta as restrições de ordem não se conseguiu paralelizar mais esta parte do código.

3.2. Computing Averages

Calculamos as médias atribuindo os gabinetes às threads disponiveis, isto levanta problemas de *load balancing* quando, temos menos gabinetes que threads disponiveis e quando nos passos do algoritmo principal cada gabinete pode ter um total de documentos atribuidos diferentes o que leva a divisão de trabalho diferente entre threads.

De modo a tentar resolver este problema de gestão de trabalho tentámos alterar a maneira como calculamos as médias.

Alterando a ordem dos fors em cascata que usamos, tentamos dividir os documentos em vez dos gabinetes pelas threads mas isto trouxe problemas de sincronização e atrasos na execução devido a termos várias threads a tentarem alterar o mesmo valor do gabinete.

Tentámos a paralelização dos for's interiores da solução original mas esta solução trouxe problemas na quantidade de overhead adicional devido à constante criação de threads.

Tentamos também efectuar a paralelização mantendo o ciclo for exterior e adicionando um ciclo for interior que se traduz numa situação de neasted parallelism. Porque havia necessidade de sincronização dos dados em termos de acessos a memória pela necessidade de todos os processos associados ao ciclo que percorre todos os documentos, verificou-se que o overhead de comunicação e de controlo se tornava mais pesado do que a vantagem que se tirava do paralelismo. Experimentou-se mesmo correr o parallel for exterior com shedulle dynamic e gided com chunks de 1, 2 e 4 em que se conseguia apenas pequenas optimizações na execução para testes maiores e com uma cobertura de paralelização do código de mais de 90Para analisar a revelância ao nível de comunicação e controlo experimentou-se criar tasks para a tarefa primitiva da soma dos valores em vez do for. Uma vez mais demonstrou não trazer vantagem para além de não se ter conseguido evitar os problemas de concorrência de dados que resultavam em resultados errados.

No final a solução inicial foi a escolhida porque mostrou ter a melhor performance entre todas as alternativas testadas.

3.3. Computing Distances

Ao calcular as distâncias separamos os documentos entre as threads disponiveis usando um simples *pragma omp for* e colocando as variaveis auxiliares na lista *private* esta provou ser a solução mais simples e rápida.

Esta implementação garante uma distribuição de trabalho entre as threads pois todos os documentos têm a mesma quantidade de trabalho, e como os documentos são independentes entre eles não existe zona crítica.

Uma alteração adicional que nos ajudou a obter melhores tempos foi o uso de uma variavel auxiliar na lista private, esta variável era do tipo Cabinet* e foi declarada static volatile, pensamos que isto ajudou na execução diminuindo o número de invalidações de cache quando as threads queriam aceder ao vector principar de gabinetes.

Conseguiu-se também aumentar o paralelismo ao colocar o document a analisar como variável local em vez de se estar a passar o vector de documents. Desta forma document era recebido por cada tarefa como firstprivate e permitiu o acesso concorrente às várias posições do vector, que são preenchidas independentemente.

Com estas várias modificações do código em relação à versão de série conseguiu-se uma cobertura de paralelismo do código de mais de 90

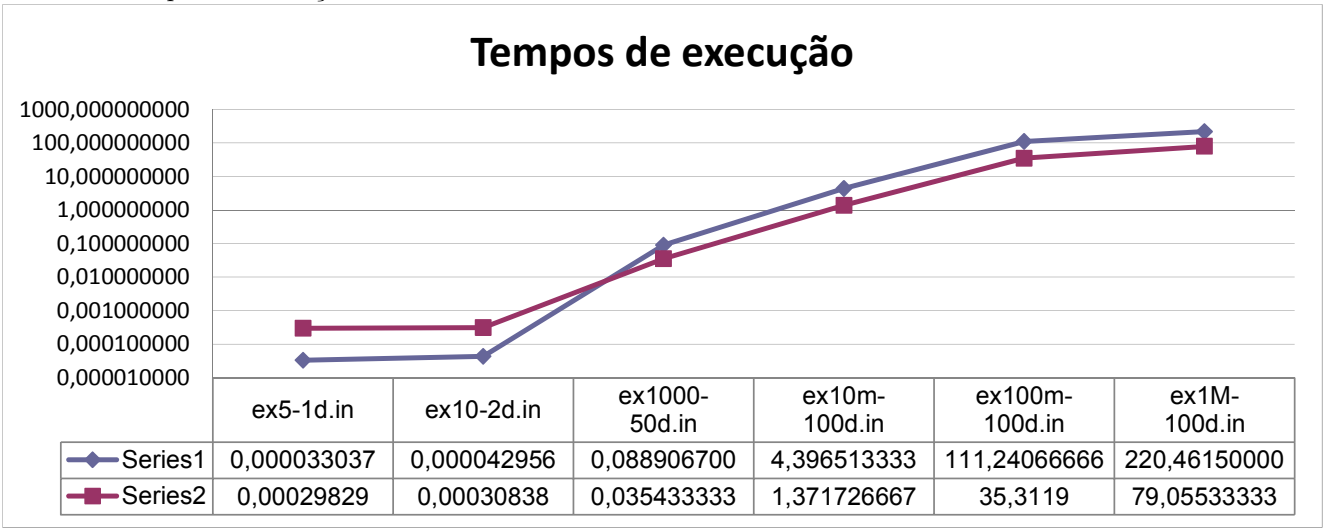
4. Resultados Obtidos

Todos os tempos de execução foram medidos usando máquinas dos laboratórios da RNL, os tempos de execução para cada teste são as médias de 6 execuções. Para ajudar com avaliação criá-mos 2 testes novos:

- ex100m-100d.in um teste com os cem mil primeiros documentos do teste ex1M-100d.in e mantendo o mesmo número de Gabinetes e Sujeitos

- ex10m-100d.in um teste com os dez mil primeiros documentos do teste ex1M-100d.in e mantendo o mesmo número de Gabinetes e Sujeitos

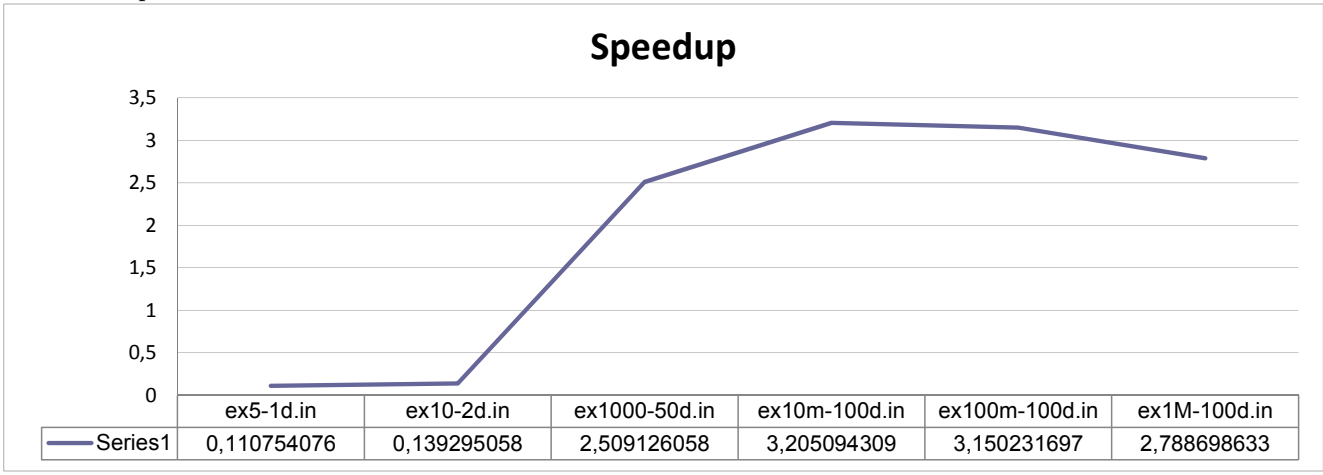
Primeiro mostramos um gráfico com uma escala logaritmica dos tempos de execução obtidos:



	ex5-1d.in	ex10-2d.in	ex1000-50d.in	ex10m-100d.in	ex100m-100d.in	ex1M-100d.in
Series1	0,000033037	0,000042956	0,088906700	4,396513333	111,24066666	220,46150000
Series2	0,00029829	0,00030838	0,035433333	1,371726667	35,3119	79,05533333

Por esta tabela podemos ver que os testes iniciais têm na verdade um atraso em relação aos de série devido ao overhead da criação de threads, mas após 3 teste começamos a observar melhoramentos nos tempos de execução.

No gráfico seguinte mostramos o Speedup calculado usando os tempos acima obtidos:



	ex5-1d.in	ex10-2d.in	ex1000-50d.in	ex10m-100d.in	ex100m-100d.in	ex1M-100d.in
Series1	0,110754076	0,139295058	2,509126058	3,205094309	3,150231697	2,788698633

Excluindo os 2 teste inicais obtemos speedups entre os 2,4 e 3.5 o que demonstra uma considerável eficiência na paralelização do algoritmo.