UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO
PROGRAMAÇÃO CONCORRENTE – SSC0143

LOYS HENRIQUE SACCOMANO GIBERTONI NUSP 8532377
SADY SELL NETO NUSP 8532418

RELATÓRIO 2 – OPENMP E OPENMPI

São Carlos
Novembro/2015

Introdução

O ser humano é uma criatura visual: tem muito mais facilidade em memorizar imagens e cenas gráficas em geral do que textos ou números. É difícil encontrar quem não guarde suas recordações como fotografias. Além disso, como se diz popularmente: “Uma imagem vale mais que mil palavras!”. Portanto, uma das áreas importantes da computação é a de computação gráfica e processamento de imagens.

Computacionalmente falando, imagens nada mais são do que matrizes, e, como se espera, operações em matrizes são geralmente simples, independentes e repetitivas. Tal cenário é o cenário perfeito para explorar o paralelismo inerente deste tipo representação sensorial.

Uma das operações embaraçosamente paralelas que pode ser feita é o estêncil de suavização ou smooth. Por definição, em estatística e processamento de imagens, realizar uma suavização sobre um conjunto de dados é criar uma função aproximada que tenta capturar padrões importantes nos dados, deixando de fora ruídos ou outras estruturas de fina escala ou fenômenos rápidos. Na suavização, os pontos de dados de um sinal são modificados de maneira que pontos individuais (provavelmente por causa de ruído) sejam reduzidos, e pontos que são menores que seus vizinhos sejam aumentados, gerando um sinal mais suave. A suavização pode ser usada de duas maneiras importantes que podem ajudar na análise de dados, sendo capaz de extrair mais informação dos dados, dado que a perda tolerada pela suavização seja razoável, e sendo capaz de providenciar análises que são tanto flexíveis quanto robustas.

Mais detalhadamente, o filtro consiste em substituir o valor de cada pixel pela média do valor dos seus vizinhos, dentro de uma janela de tamanho definido, incluindo a si próprio. A menos de sincronização para impedir que a alteração de um determinado valor de pixel influencie no cálculo dos demais, a aplicação desse filtro se dá de perfeitamente bem em paralelo por possuir uma independência inata de dados.

O paralelismo se torna ainda mais interessante quando se tem vários nós de processamento. A imagem pode ser então dividida em blocos, e ter a suavização feita por bloco, depois reagrupar os grupos suavizados. Neste caso, o paralelismo dá-se a nível de processo: são criados vários processos, um para cada bloco, divididos entre os nós, responsáveis por suavizar cada bloco. Por meio do modelo de passagem de mensagens, torna-se possível fazer essa dispersão (*scatter*) para os nós processarem, e também a reunião (*gather*) dos dados suavizados. Tal modelo se torna particularmente vantajoso para imagens grandes: com a divisão de tarefas, cada nó terá um bloco potencialmente

pequeno para processar; uma rede de comunicação eficiente também é desejável, para que o overhead criado para a comunicação não supere os cálculos que serão realizados. Este é o papel da tecnologia OpenMPI empregada no trabalho: fornecer uma interface para passagem de mensagens.

Porém, ainda assim, isso seria explorar pouco do paralelismo inerente aos problemas de processamento de imagens. Afinal de contas, dessa forma apenas cria-se a divisão de tarefas em subtarefas menores, porém ainda é possível aproveitar o fato de o problema consistir em operações simples, (razoavelmente) independentes e repetitivas. Em outras palavras, por mais que tenha-se empregado vários nós para resolver o problema, o que já ajuda a aumentar consideravelmente a eficiência da solução. No entanto, pensando isoladamente em cada nó, esse desempenho pode ser ainda melhor, se esse "paralelismo inerente" for explorado. Usando um modelo de criação de threads para a aplicação do filtro em cada nó, obter-se-á uma eficiência maior em cada nó, melhorando a solução localmente. Aliando tal fato à divisão do problema em subproblemas menores, a melhoria em cada subproblema implica na melhoria global. Para esse papel, foi empregada a ferramenta OpenMP.

O algoritmo que aplica o estêncil de suavização sobre uma imagem foi implementado em C++ utiliza uma janela de 5x5 pixels e pode ser usado para suavizar imagens em formato .ppm ou .pgm de forma sequencial ou paralela, de acordo com a escolha do usuário.

Resultados obtidos

A tabela abaixo (precedida por sua legenda) mostra o tempo de execução amostral dos algoritmos:

Legenda da tabela:

- S_i: tempo de execução da iésima amostra sequencial em segundos;
- $\overline{S}$: média dos tempos de execução das amostras sequenciais em segundos;
- σ_S: desvio padrão dos tempos de execução das amostras sequenciais em segundos ;
- P_i: tempo de execução da iésima amostra paralela em segundos;
- $\overline{P}$: média dos tempos de execução das amostras paralelas em segundos;
- σ_p: desvio padrão dos tempos de execução das amostras paralelas em segundos;
- R: speedup.

Tempo / Arquivo	flower_duo .pgm	italian_valley .pgm	large_elevation .pgm	tower_bridge .pgm
S_1	2,98364	2,94813	9,86047	9,71378
S_2	2,99607	3,03026	9,64774	7,64976
S_3	3,91852	3,03242	9,59506	7,56306
S_4	3,90954	2,95069	9,63196	7,44941
S_5	3,79223	2,98903	9,61638	8,63334
S_6	2,95434	2,96409	12,7669	7,40644
S_7	3,53312	3,01666	9,60617	7,47557
S_8	2,99272	2,99214	9,60249	7,53385
S_9	3,82387	2,97163	9,77429	7,42659
S_{10}	2,9908	2,98252	9,60053	7,46756
$\overline{S}$	3,389485	2,987757	9,970199	7,831936
σ_S	0,440589	0,03064	0,986613	0,754937
P_1	1,00062	1,16329	2,21103	1,74603
P_2	0,846907	1,18894	1,73436	1,81784
P_3	1,66945	1,03506	1,7456	1,55614
P_4	0,943828	1,28428	2,19077	1,80909
P_5	0,81582	0,815266	1,75201	1,77535
P_6	0,846897	0,943999	2,37769	1,48012
P_7	1,03258	0,869898	1,71457	1,4347
P_8	1,07242	1,1838	1,75514	1,58468
P_9	0,847368	0,845297	1,76574	1,57936
P_{10}	0,846669	3,14145	1,76933	1,98265
$\overline{P}$	0,992256	1,247128	1,901624	1,676596
σ_P	0,254871	0,685679	0,252343	0,174889
R	3,415938024	2,395709983	5,24299178	4,671331674

Tabela 1 - tempos de execução do algoritmo para imagens grayscale

Tempo / Arquivo	flower_duo .ppm	italian_valley .ppm	large_elevation .ppm	tower_bridge .ppm
S_1	5,93103	4,90892	16,0017	12,2323
S_2	4,90536	4,91859	15,9594	15,7032
S_3	4,92404	5,04309	15,9238	12,2409
S_4	5,16606	4,89341	15,9652	12,2382
S_5	4,91175	4,89373	15,962	12,2329
S_6	4,96671	4,9629	15,9612	13,611
S_7	4,94645	4,89118	15,9208	12,3864
S_8	4,9182	4,89338	15,9275	12,2319
S_9	4,91178	6,49798	15,9259	12,2544
S_{10}	4,90771	4,92475	15,9821	12,2349
$\overline{S}$	5,048909	5,082793	15,95296	12,73661
σ_S	0,319714	0,499453	0,027549	1,126895
P_1	1,41068	1,39482	2,88433	2,3571
P_2	1,41551	1,39883	2,88479	2,38554
P_3	1,3883	1,38521	2,84278	2,40582
P_4	1,39636	1,22336	2,84696	2,38771
P_5	1,35218	1,51153	2,86141	2,38042
P_6	1,43278	1,3546	2,95704	2,38062
P_7	1,41853	1,52559	2,92695	2,38977
P_8	1,35812	1,21811	2,91457	2,41971
P_9	1,1971	1,4886	2,88964	2,78369
P_{10}	1,57832	1,33997	3,09649	2,36487
$\overline{P}$	1,394788	1,384062	2,910496	2,425525
σ_P	0,093487	0,107351	0,074431	0,127115
R	3,619839718	3,672373781	5,481182589	5,251073479

Tabela 2 - tempos de execução do algoritmo para imagens rgb

Esses resultados foram obtidos executando o programa no cluster Cosmos do LASDPC. As versões sequenciais foram rodadas no *node01* (10.1.1.10), e as versões paralelas em todos os nós, incluindo o frontend, com exceção do *node02* (10.1.1.20) e do *node06* (10.1.1.60).

Soluções

Estruturas de dados e arquivos

A solução adotada pelo grupo para a implementação do algoritmo em C++ foi criar uma classe que contém todos os métodos necessários para executar a suavização tanto de forma sequencial, quanto paralela (usando OpenMP + OpenMPI).

O foco do código desenvolvido pelo grupo é a organização e o fácil entendimento, e não a eficiência, o que tem impacto no tempo de execução, mas não no speedup entre a versão sequencial e paralela. Esse foi o motivo da escolha do grupo pela linguagem C++.

Como não se optou por uma biblioteca já implementada de computação gráfica, como OpenCV, com o intuito de não aumentar muito o overhead no que tange à leitura das imagens, o próprio grupo implementou a leitura de imagens Netpbm, isto é, imagens no formato .pgm e .ppm. Nenhum outro formato, como por exemplo, os formatos comprimidos (.jpeg, .png, entre outros) são aceitos. A leitura armazena o conteúdo inteiro da imagem com um vetor unidimensional de bytes, que na linguagem C++ podem ser implementados como `unsigned char`.

O vetor é unidimensional: ele armazena os dados lidos da imagem de maneira sequencial e na mesma ordem que o algoritmo os lê da imagem. De uma maneira mais tangível, suponha um arquivo .pgm com os valores $v_1, v_2, v_3, \ldots$ (onde v_i representa o i-ésimo valor de cor de cinza da imagem) e um arquivo .ppm com os valores $(r_1, g_1, b_1), (r_2, g_2, b_2), (r_3, g_3, b_3), \ldots$ (onde r_i representa o i-ésimo valor de vermelho da imagem, g_i o i-ésimo valor de verde e b_i, o i-ésimo valor de azul). Esses dados serão armazenados no vetor como $v_1, v_2, v_3, \ldots$ e $r_1, g_1, b_1, r_2, g_2, b_2, r_3, g_3, b_3, \ldots$, respectivamente. Note também que, as imagens por si só são bidimensionais, mas o vetor que as armazena é unidimensional.

Tal multidimensionalidade levou a criação dos métodos que mapeiam posição e acesso, bidimensional ou tridimensional. Como exemplo, tem-se que o método bidimensional aceita dois parâmetros, que seriam os supostos dois índices necessários para o acesso do vetor, e retorna uma posição correspondente à posição equivalente do vetor unidimensional. Também há o processo inverso: um método que recebe uma posição absoluta, e retorna como uma tupla os índices equivalentes em duas ou três dimensões.

Decomposição do problema

De acordo com o livro "Introduction to Parallel Computing", segunda edição, a técnica de decomposição utilizada foi a decomposição de dados.

Primeiramente é fácil perceber, por eliminação, que a única decomposição plausível para este problemas é a de dados. Observe: a decomposição recursiva não se aplica, pois não se trata de um método / função / procedimento recursivo; também não pode ser decomposição exploratória, uma vez que o método não consiste na problemática de dados vários caminhos, encontrar qual deles levaria à solução, conforme é usado na filosofia de decomposição exploratória; tampouco é decomposição especulativa, dado que o programa não consiste em calcular duas (ou mais) possibilidades, para depois utilizar uma delas.

No entanto, a decomposição de dados se encaixa perfeitamente no problema. A divisão que foi feita, foi para dividir o conteúdo da imagem entre vários nós e threads de processamento. Em outras palavras, foram os dados de processamento (que são obtidos por meio de entrada) que foram particionados, configurando o exato cenário de decomposição de dados.

Algoritmo

Para a execução do algoritmo, tanto em versão sequencial quanto paralela, o construtor público aceita como parâmetro um nome de arquivo, o arquivo .pgm ou .ppm do qual serão lidos os dados da imagem. A leitura é feita de maneira genérica para os dados de cabeçalho: tipo de imagem, largura, altura e valor máximo de cor, uma vez que aquele sempre armazena estes em modo texto e nesta ordem para quaisquer tipos de arquivos Netpbm. O tipo lido é muito importante, pois sinaliza melhor do que a extensão sobre como o conteúdo está armazenado. Então, um método apropriado para o tipo lido é chamado para configurar a dimensão da imagem, isto é, o número de cores (1 para imagens .pgm, 3 para imagens .ppm; a escolha do nome dimensão está no fato de esse valor representar o tamanho da terceira dimensão / profundidade de um vetor tridimensional, caso a implementação usasse arranjo multidimensional) ler o restante dos dados, os quais tratam do conteúdo da imagem propriamente dita.

Os métodos chave deste trabalho são `smooth_sequential`, `smooth_parallel` e `smooth`. O método `smooth_sequential` apenas chama o método auxiliar privado `smooth`, responsável pela tarefa de suavização. Já o método `smooth_parallel`

gerencia toda a parte relativa ao modelo de passagens de mensagens (usando a ferramenta OpenMPI): a divisão de tarefas entre os nós, a dispersão de dados pelos nós, a suavização de cada nó (feita pelo mesmo método auxiliar `smooth` usado pelo método `smooth_sequential`), a reunião de dados suavizados e composição da nova imagem. Ambos os métodos calculam o tempo de execução do filtro, sendo que é possível imprimir esse tempo, conforme será discutido mais abaixo.

No aspecto da divisão de tarefas entre os nós, no modelo de passagem de mensagens, o mestre trabalha também: o trabalho é dividido entre todos os nós, incluindo o nó mestre, que também será encarregado de calcular a sobra da imagem que não poderá ser passada pela dispersão dos dados.

Além disso, note que há uma pequena perda quando se divide a imagem, uma vez que cada nó enxergará sua porção a ser calculada como uma imagem independente. Perceba que as bordas divisão eram contínuas antes de haver a divisão, e a janela conteria mais pixels; porém com a divisão, uma borda não “enxerga” o resto da janela contida em outra parte da imagem, que estará sendo processada por outro nó. Contudo, como o algoritmo conta o número de vizinhos na janela 5x5 centrada em um determinado pixel ao invés de preencher a janela com pixels espúrios, a suavização é mais fiel ao conteúdo da imagem (uma vez que não se introduz pixels estrangeiros), sendo que tal perda se torna realmente irrelevante.

O método que de fato realiza a suavização é o método `smooth`. Este método é responsável por criar uma imagem resultante, usando o construtor privado, que aceita parâmetros relativos a largura, altura, valor máximo de cor, e dimensão, e cria uma imagem com tais especificações, com vetor de capacidade suficiente para comportar uma imagem desse porte, porém vazio; percorrer cada pixel da matriz; percorrer seus vizinhos válidos pertencentes à janela de 5x5 pixels centrado nele; somar o valor deles para cada cor, contar o número de vizinhos envolvidos neste processo; calcular a média; e atribuir a média na posição correspondente na imagem resultante.

O laço que percorre os pixels da imagem pode ser paralelizado, conforme especificado pela diretiva `pragma omp parallel for`, caso em que, serão criadas várias *threads*, cada uma percorrendo uma parte da imagem. O número de *threads* e a divisão de trabalho entre elas é gerenciada pela própria ferramenta OpenMP, a fim de obter um desempenho ótimo. Todavia, nem sempre deseja-se a paralelização do laço; no caso sequencial, percorrer a imagem com tecnologia paralela seria contraditório. Para resolver estre problema, a diretiva conta com a cláusula `if(parallel)`. O parâmetro `parallel` é um parâmetro booleano que indica se o laço deve ou não ser feito em

paralelo (terá o valor `false` quando for chamado pelo método `solve_sequential`, e o valor `true` quando for chamado pelo método `solve_parallel`).

Além disso, o método de execução paralela também chama uma outra versão sobrecarregada do método `smooth`, que tem a mesma função que o método descrito até então, exceto que ele recebe dois parâmetros: `first` e `last`, e aplica somente o filtro para o intervalo de linhas [`first`, `last`) na imagem, e o faz apenas de modo paralelo, uma vez a que a versão sequencial nunca chama este método.

Por fim, os métodos incialmente descritos, `smooth_sequential` e `smooth_parallel`, possuem duas *flags* (com valor padrão `false`): `will_save` e `echo_time`, as quais indicam para o método se ele deve salvar a imagem resultante também (*flag* `will_save`) e se ele deve imprimir o tempo gasto para aplicação do filtro (*flag* `echo_time`). De qualquer forma, será retornado um ponteiro para imagem suavizada.

Para explicações mais detalhadas sobre o algoritmo, veja os comentários contidos no código fonte.

Link do github

O código é público e está hospedado no github. Para cloná-lo, basta clonar a url:
https://github.com/SadySellNeto/Programacao_Concorrente_2015_T2-Grupo01a.git (https)
git@github.com:SadySellNeto/Programacao_Concorrente_2015_T2-Grupo01a.git (ssh)

Dificuldades

O grupo passou por diversas dificuldades. A começar, uma das dificuldades mais básicas encontradas pelo grupo foi o fato de este trabalho necessitar de duas tecnologias até então novas (OpenMP e OpenMPI) para atingir sua plenitude. Foi preciso realizar vários testes isolados com cada uma delas para entender seus funcionamentos e avaliar seus desempenhos em diferentes ocasiões. Outro problema relativo a tais tecnologias é que, apenas conhecê-las não foi o suficiente; também foi necessário estudar como integrá-las.

Conhecendo as tecnologias, o próximo problema enfrentado foi decidir a melhor forma de decomposição dos dados. Por exemplo, conhecendo todas as diretivas e cláusulas da ferramenta OpenMP, decidir qual laço ou qual estrutura de programa paralelizar foi um problema razoável. Após realizar diferentes testes, obteve-se eficiência

máxima ao paralelizar apenas o laço que percorre a imagem para aplicar a suavização para aquele pixel; houve uma queda de eficiência ao se tentar paralelizar o laço do cálculo do filtro em si ou a combinação dos dois. Os problemas não se limitam apenas a esse. O grupo teve um pequeno problema em adequar o arranjo multidimensional em um vetor unidimensional; no entanto, esta uma foi uma dificuldade pífia para não dizer nula.

O grupo também enfrentou problemas para decidir a melhor divisão de tarefas para cada nó na parte de OpenMPI. Eventualmente optou-se por dividir em blocos de linhas, uma vez que os dados de uma linha encontram-se sequenciais no vetor. Qualquer outra configuração ia requerer um acesso intercalado à estrutura de dados, o que causaria uma queda sensível na eficiência do algoritmo. Infelizmente, aliado a esse problema, o grupo enfrentou o problema de decidir o que seria feito com o trecho da imagem que não foi possível enviar via dispersão de dados (*scatter*), eventualmente deixando o excesso de carga para o mestre; problema este que gerou outros problemas, principalmente de mapeamento de memória.

Por fim, um problema em menor escala, porém potencialmente inibidor do trabalho inteiro foram problemas de acesso ao cluster. O grupo teve se conectar várias vezes a vários nós via o comando `ssh` para que o comando `mpirun` conseguisse acessá-los. Além disso, à pedido do professor, o grupo não utilizou o nó de endereço 10.1.1.20, tampouco o nó de endereço 10.1.1.60, pois não respondia à conexão *ssh*, nem mesmo pelo nome de “node06”, e também não pelo nome de “ceres”.

Metodologia de execução dos experimentos

Os experimentos foram realizados da seguinte maneira: foram baixadas as imagens para a o cluster; como o grupo não possuía permissão de escrita nos diretório `/home/grupo01a/img/` e seus subdiretórios, resolveu-se criar um outro diretório, `/imagens/` no diretório `/home/grupo01a`, com a seguinte estrutura:

```
/
|    home/
|    |    grupo01a/
|    |    |    imagens/
|    |    |    |    gray/
|    |    |    |    |    flower_duo.pgm
|    |    |    |    |    italian_valley.pgm
|    |    |    |    |    large_elevation.pgm
```

```
|   |   |   |   |   tower_bridge.pgm
|   |   |   |   rgb/
|   |   |   |   |   flower_duo.ppm
|   |   |   |   |   large_elevation.ppm
|   |   |   |   |   italian_valley.ppm
|   |   |   |   |   tower_bridge.ppm
```

A partir daí, executava-se o programa de maneira intercalada, primeiro entre execução, depois entre arquivos (por exemplo, `flower_duo.pgm` sequencial, `flower_duo.pgm` paralelo, `italian_valley.pgm` sequencial, `italian_valley.pgm` paralelo, e assim por diante). Depois, repetiu-se o experimento 10 vezes, amostrando cada um dos tempos e calculando a média e o desvio padrão deles.

Referências

Documentação cabeçalho string: http://www.cplusplus.com/reference/string/
Documentação cabeçalho vector: http://www.cplusplus.com/reference/vector/
Documentação cabeçalho list: http://www.cplusplus.com/reference/list/
Documentação cabeçalho chrono: http://www.cplusplus.com/reference/chrono/
Documentação cabeçalho stdexcept: http://www.cplusplus.com/reference/stdexcept/
Documentação cabeçalho thread: http://www.cplusplus.com/reference/thread/
Documentação cabeçalho istream: http://www.cplusplus.com/reference/istream/
Documentação cabeçalho ostream: http://www.cplusplus.com/reference/ostream/
Documentação cabeçalho sstream: http://www.cplusplus.com/reference/sstream/
Documentação cabeçalho fstream: http://www.cplusplus.com/reference/fstream/
Documentação cabeçalho iostream: http://www.cplusplus.com/reference/iostream/
Dúvidas gerais de programação: http://stackoverflow.com/