Computação Paralela 2018/19 – Projecto 2

Eduardo R. B. Marques, DCC/FCUP

1 Introdução

1.1 Objectivo

Pretende-se que faça uso do OpenMP para a paralelisação de:

1. Três algoritmos dados para multiplicação de matrizes, que se diferenciam por diferentes padrões de acesso à memória.
2. Dois algoritmos que empregam a tradicional eliminação de Gauss com "pivot" e "back-substitution" para **(a)** resolução de sistemas de equações lineares e **(b)** cálculo da matriz inversa.

1.2 Entrega e apresentação

Envie o seu trabalho por e-mail para `edrdo@dcc.fc.up.pt` até **17 / 12 / 2018**. Na mensagem anexe o seu relatório em formato PDF e o seu código num arquivo ZIP (execute `make clean` previamente). A data de apresentação do trabalho será posteriormente anunciada.

1.3 Código base

Para a realização do trabalho deve obter um arquivo ZIP de código base disponível na página da disciplina. Veja o documento `README.txt` para detalhes sobre a estrutura de código, sua compilação e uso.

1.4 Uso da máquina `sibila2`

Será disponibilizado acesso à máquina `sibila2.dcc.fc.up.pt` para poder avaliar a correcção e escalabilidade do seu código. Para aliviar a carga sobre a máquina, use tanto quanto possível os computadores de laboratório para desenvolvimento e validação preliminar, e apenas a `sibila2` para executar código já robusto e tirar resultados para análise final de desempenho dos seus programas.

Para copiar ficheiros de máquina para máquina utilize o utilitário `scp`. Tenha o cuidado de preservar a sua área de trabalho de acesso indevido por outros utilizadores; execute `chmod go-rwx /home/mylogin` para prevenir esse acesso.

1.5 Avaliação

A avaliação terá em conta a clareza, correção e paralelisação adequada do seu código, e a exposição do seu trabalho no relatório e apresentação.

1.6 Referências

- "Parallel Programming in C with MPI and OpenMP", M. J. Quinn, §11.2, §12.3, §12.4.
- "Parallel Programming for Multicore and Cluster Systems", T. Rauber e G. Runger, §4.6.2, §8.1.

2 Multiplicação de matrizes

2.1 Algoritmo tradicional

O pseudo-código abaixo exprime o algoritmo tradicional para obter `C` = `A` × `B` onde `A`, `B`, e `C` são matrizes com dimensões `m` × `n`, `n` × `p` e `m` × `p`, respectivamente.

```
// assume-se que C está previamente inicialisada com zeros
for i = 0, 1, ... m - 1
    for j = 0, 1, ... p - 1
        for k = 0, 1, ... n - 1
            C[i, j] += A[i, k] * B[k, j]
```

Implemente e paralelise este algoritmo em `mm/mm1.c`.

2.2 Algoritmo com inversão de índices

Observe que no algoritmo original podemos obter o mesmo resultado invertendo a ordem dos dois ciclos mais internos, isto é:

```
// assume-se que C está previamente inicialisada com zeros
for i = 0, 1, ..., m - 1
    for k = 0, 1, ..., n - 1
        for j = 0, 1, ..., p - 1
            C[i, j] += A[i, k] * B[k, j]
```

Para o ciclo mais interno desta variação do algoritmo, os acessos a `B` e `C` em iterações sucessivas são a posições contíguas e `A[i,k]` é constante. Será que isso poderá acelerar significativamente a execução do algoritmo?

Implemente e paralelise este algoritmo em `mm/mm2.c`.

2.3 Algoritmo com "tiling"

Por último, considera-se o seguinte algoritmo que emprega a técnica de "tiling":

```
// assume-se que C está previamente inicialisada com zeros
for ii = 0, T, 2 * T, ...,  m - m % T
    for kk = 0, T, 2 * T, ..., n - n % T
        for jj = 0, T, 2 * T, ..., p - p % T
            for i = ii, ii+1, ... , min(ii + T, m) - 1
                for k = kk, kk+1, ..., min(kk + T, n) - 1
                    for j = jj, jj+1, ... , min(jj + T, p) - 1
                        C[i,j] += A[i, k] * B[k, j]
```

Os três ciclos mais internos (em `i`, `j` e `k`) operando em sub-blocos (de `A`, `B` e `C`) de dimensão `T` × `T`, onde `T` é um parâmetro escolhido por forma a tentar que os dados acedidos estejam em memória "cache" com alta probabilidade. Implemente e paralelise este algoritmo em `mm/mm3.c`. Na estrutura de código dada, é sugerido que altere o parâmetro `T` compilando o programa usando "`make T=<valor>`". **Nota:** ao pseudo-código `for v = 0, T, 2 * T ..., lim - lim % T` corresponde em C simplesmente `for (v = 0; v < lim; v += T)` .

2.4 Validação

Faça uso dos utilitários `grm.bin` e `gim.bin` para gerar matrizes de teste, e do utilitário `cm.bin` para comparar duas matrizes.

2.5 Análise de desempenho

Analise o comportamento dos três algoritmos de multiplicação de matrizes, variando:

- o número de threads de 1 a 16 em múltiplos de 2;
- dois/três valores distintos para a dimensão n de matrizes quadradas, (por ex. 1024, 2048) por forma a que a execução dos algoritmos com 1 thread não exceda 2 minutos – não se esqueça entretanto de validar a correcção dos programas para matrizes que não sejam quadradas!;
- Para `mm3.c`, diferentes valores para o tamanho de T (ex., 16, 32, 64, ...) que permitam discernir o valor óptimo para a máquina `sibila2`.

Apresente no relatório uma tabela dos tempos médios de 5 execuções para cada configuração (incluindo p/os vários valores de `T` no caso de `mm3.c`), e respectivo "speed-up", bem como uma apreciação geral dos resultados.

3 Resolução de sistemas de equações lineares

3.1 Código sequencial

É dado em `sles/sles_seq.c` o algoritmo tradicional de resolução de sistemas de equações lineares $Ax = b$ para matrizes quadradas. Como usual, o algoritmo emprega eliminação de Gauss com "pivot" seguida de "back substitution".

3.2 Análise de dependências e paralelisação

Pretende-se que:

1. Identifique no código sequencial as dependências de dados, em particular quais delas são "loop-carried", usando a terminologia e notação dada nas aulas teóricas. Apresente esta análise no relatório.
2. Implemente em `sles/sles_par.c` uma paralelização do algoritmo sequencial. Pretende-se que o algoritmo paralelo siga os mesmos passos lógicos do algoritmo sequencial, isto é, que seja uma paralelização efectiva do código original). **Não** deverá considerar formulações alternativas que o descaracterizem ou outros algoritmos. Sugere-se que paralelize o algoritmo de forma gradual.

3. Descreva no relatório os traços gerais da sua aproximação de paralelização e em particular a resolução de dependências "loop-carried" que afectavam à partida a paralelisação. Se não conseguir paralelisar certos passos do algoritmo, explique a dificuldade que encontrou em fazê-lo.

3.3 Validação

Pode gerar matrizes de teste usando o utilitário `gles.bin`.

O script `sles.sh` permite validar se o resultado do programa `sles_par.bin` está correcto.

3.4 Cálculo da matriz inversa

O algoritmo de eliminação de Gauss-Jordan calcula a inversa de uma matriz quadrada `A` resolvendo o sistema de equações $\mathtt{A}\,\mathtt{X} = \mathtt{I}$ onde `I` é a matriz identidade.

Escreva em `inv/inv.c` a implementação do algoritmo, generalizando/adaptando o código que tenha feito em `sles_par.c`.

As matrizes de teste poderão ser geradas usando de novo o programa `gsles.bin`, e a correcção do resultado validada com um dos programas de multiplicação de matrizes (afinal de contas deveremos ter $\mathtt{A} \times \mathtt{A}^{-1} = I$!).

3.5 Análise de desempenho

Faça uma análise do desempenho dos programas `sles_par.bin` e `minv.bin` em contornos similares aos pedidos para os algoritmos de multiplicação de matrizes.