Turma: 06N	Semestre: 2° Sem 2016	Nota:
Prof. Mário Olímpio de Menezes		

Instruções: A prova deve ser feita sem consulta. O entendimento faz parte da prova. As respostas devem ser assinaladas a CANETA. **Desligue o seu celular durante a realização da prova!**

Esta prova tem 11 questões. O total de pontos é 10.

Questão 1 (1 pt(s))

Dado um vetor de 16 elementos que precisa ser processado por 4 threads (T0, T1, T2, T3). Assinale a alternativa correta com relação ao esquema de divisão do trabalho e os escalonamentos (schedules).

- ○ Utilizando-se `schedule(static,2)`, $v[13]$ será processado por T3 na última rodada, $v[3]$ será processado por T2 na primeira rodada e $v[6]$ será processado por T1 na terceira rodada.
- ○ Utilizando-se `schedule(static,1)`, $v[13]$ será processado por T1 na última rodada, $v[3]$ será processado por T3 na primeira rodada e $v[6]$ será processado por T2 na segunda rodada.
- ○ Utilizando-se `schedule(static,4)`, $v[13]$ será processado por T1 na última rodada, $v[3]$ será processado por T1 na primeira rodada, $v[6]$ será processado por T1 na terceira rodada.
- ● Utilizando-se `schedule(static,2)`, $v[13]$ será processado por T3 na última rodada, $v[3]$ será processado por T1 na segunda rodada e $v[6]$ será processado por T0 na segunda rodada.

Questão 2 (½ pt(s))

Selecione a alternativa que apresenta definição e conceito corretos com relação a OpenMP.

- ○ OpenMP é um paradigma para o desenvolvimento de programação paralela em sistemas de memória distribuída.
- ● OpenMP é uma API para o desenvolvimento de programação paralela em sistemas de memória compartilhada.
- ○ OpenMP é um middleware para o desenvolvimento de programação paralela em sistemas de memória compartilhada.
- ○ OpenMP é uma metodologia para o desenvolvimento de programação paralela em sistemas de memória, otimizada para memória cache.

Questão 3 (1 pt(s))

O OpenMP utiliza _____ do compilador para o pré-processamento do código fonte. Uma linha de código típica para iniciar um trecho de código paralelo com OpenMP é _____. Selecione a alternativa que oferece as opções corretas para preencher esta frase.

- ○ bibliotecas; `#pragma omp parallel num_threads(thread_count)`
- ○ diretivas; `#def omp_set_num_threads(thread_count)`
- ● diretivas; `#pragma omp parallel num_threads(thread_count)`
- ○ bibliotecas; `#pragma omp pthread_create(thread_count)`

Questão 4 (½ pt(s))

Considere o seguinte trecho de código e responda:

```
fibo[1] = 1;
#pragma omp parallel for num_threads(2)
  for (i = 2; i < n; i++)
    fibo[i] = fibo[i-1] + fibo[i-2];
```

- (●) A execução deste código ocorrerá sem problemas com quaisquer números de threads.
- (●) A execução deste código poderá gerar um resultado incorreto por conta da dependência entre as iterações do laço.
- ( ) Para que este código execute corretamente, é preciso adicionar uma cláusula `schedule`, para dividir corretamente o trabalho entre as threads.
- ( ) Uma medida que pode ser utilizada neste código para que ele execute sem problemas é a utilização da cláusula `private(fibo)`.

Questão 5 (1 pt(s))

Uma das maneiras de se determinar o número Pi é a partir da série

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

O termo de "soma" pode ser implementado pelo seguinte trecho de código:

```
double sum = 0.0;
PRAGMA OpenMP
for (k = 0; k < n; k++) {
  if (k % 2 == 0)
     factor = 1.0;
  else
     factor = -1.0;
  sum += factor/(2*k + 1);
}
```

Qual das alternativas abaixo é a cláusula OpenMP correta para ser utilizada no código dado.

- (●) `# pragma omp parallel for num_threads(thread_count) reduction(+:sum)`
- ( ) `# pragma omp parallel for num_threads(2)`
- ( ) `# pragma omp parallel for num_threads(thread_count) reduction(+:sum) private(factor)`
- ( ) `# pragma omp parallel schedule(static, 8) private(factor)`

Questão 6 (1 pt(s))

Sobre a cláusula reduction no OpenMP

- ( ) É utilizada para se reduzir o tamanho do código, geralmente entre 20 e 30%.
- ( ) É utilizada para se conseguir um modelo de exclusão mútua, reduzindo-se o risco de resultados inconsistentes.
- (●) É utilizada para se reduzir (diminuir) o valor de uma variável, aplicando-se sempre um operador soma aos resultados.
- ( ) É utilizada para combinar múltiplas cópias locais de uma variável em diferentes threads, em uma única cópia na thread master.

Questão 7 (1 pt(s))

Para que as operações de envio e recebimento de mensagens funcionem no MPI, alguns campos das funções MPI_Send e MPI_Recv devem ser casados, isto é, devem ter valores correspondentes. Selecione a alternativa que indica os campos que obrigatoriamente devem ter valores casados.

- ( ) Tipo de dado enviado/recebido, emissor/receptor, rótulo da msg, communicator.
- ( ) Função send/recv e rótulo da msg.
- ( ) Communicator, rótulo da msg, função de broadcast, tipo de dado.
- (●) Rótulo da msg, função de redução, communicator, emissor/receptor.

8

P2 ProgPar — FCI – Mackenzie — Prof. Mário Menezes

Questão 8 (1 pt(s))
Em um programa feito com MPI, sempre utilizamos uma chamada _____ Depois se queremos saber quantos processos fazem parte do grupo de comunicação, devemos utilizar _____

- MPI_Init(NULL, NULL); MPI_get_num_threads(MPI_COMM_WORLD, &tamk);
- MPI_Init(NULL, NULL); MPI_Comm_size(MPI_COMM_WORLD, &tamk);
- MPI_Open(NULL, NULL); MPI_NUM_THREADS(MPI_COMM_WORLD, &tamk);
- MPI_Start(Communicator); MPI_Comm_size(MPI_WORLD, &tamk);

Questão 9 (1 pt(s))
Considere as seguintes afirmações:

I. O modelo de programação do MPI é o SPMD, já que o mesmo programa vai rodar nas diversas máquinas do cluster.

II. MPI é um paradigma de programação paralela destinado a sistemas de memória compartilhada, onde é possível fazer a troca de mensagens por buffers compartilhados entre os processos.

III. MPI deve ser utilizado sempre que se deseja construir sistemas de processamento em placas gráficas (GPUs) porque ele utiliza as mais modernas tecnologias de troca de mensagens.

IV. Para distinguir qual parte do código deve ser executada pelos processos trabalhadores, uma simples cláusula `if` é suficiente. O `rank` do processo no Communicator identifica se é master (0) ou slave ($\neq 0$).

V. No MPI, o conceito de Comunicadores é o de um grupo de processos que trocam mensagens entre si para a realização de determinada tarefa.

Assinale a alternativa correta:

- As afirmativas II e I estão corretas.
- As afirmativas III e V estão erradas.
- As afirmativas I e V estão corretas, mas a alternativa IV está errada.
- As afirmativas IV e V estão corretas, juntamente com a afirmativa I que também está correta.

Questão 10 (1 pt(s))
Um programador escreve uma função para um sistema que rodará em um cluster com MPI, conforme mostrado abaixo.

```
void rr_v(
 double     local_a[],
 int        local_n,
 int        local_m,
 int        n,
 int        my_rank,
 MPI_Comm   comm){
 double* a = NULL;
 int i;
 if (my_rank == 0) {
    a = malloc(n*sizeof(double));
    for (i = 0; i < n; i++)
        a[i] = rand()%12;
 }
 MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_m, MPI_DOUBLE, 0, comm);
 free(a);
}
```

Considere que `local_a`, `my_rank` e `comm` foram corretamente declaradas.

- A função deve ter sido chamada como `rr_v(local_a,10,200,200,my_rank,comm)`.
- A função deve ter sido chamada como `rr_v(local_a, 200, 10, 10, my_rank, comm)`.
- A função deve ter sido chamada como `rr_v(local_a, 10, 10, 200, my_rank, comm)`.
- A função deve ter sido chamada como `rr_v(local_a, 200, 200, 10, my_rank, comm)`.

```
    if (my_first_i % 2 == 0) /* my first i is even */
        factor = 1.0;
    else /* my first i is odd */
        factor = -1.0;
    <TRECHO DE CODIGO>
    return NULL;
}  /* Thread sum */
```

Assinale a alternativa correta com o trecho de código que deve ser colocado no local indicado para que os resultados da execução sejam eficientes (com o mínimo de serialização) e corretos.

●
```
for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
    pthread_mutex_lock(&mutex1);
    sum += factor/(2*i+1);
    pthread_mutex_unlock(&mutex1);
}
```

○
```
pthread_mutex_lock(&mutex1);
for (i = my_first_i; i < my_last_i; i++, factor = -factor) {
    sum += factor/(2*i+1);
}
pthread_mutex_unlock(&mutex1);
```

○
```
for (i = my_first_i; i < my_last_i; i++, factor = -factor)
    my_sum += factor/(2*i+1);
pthread_mutex_lock(&mutex1);
sum += my_sum;
pthread_mutex_unlock(&mutex1);
```

○
```
for (i = my_first_i; i < my_last_i; i++) {
    pthread_mutex_lock(&mutex1);
    factor = -factor;
    pthread_mutex_unlock(&mutex1);
    sum += factor/(2*i+1);
}
```

V = [0, 1, 2 ... 13, 14, 15]

3. (1½ pt(s)) Considere um vetor **v** de 16 elementos que precisa ser processado por 4 threads (T0, T1, T2, T3). Realize o escalonamento deste trabalho utilizando o `schedule(static,2)`.

T0 → 0 1 8 9
T1 → 2 3 10 11
T2 → 4 5 12 13
T3 → 6 7 14 15

4. (1 pt(s)) Uma das maneiras de se prover Coerência de Cache é através do uso de um mecanismo de "bisbilhotagem". Este mecanismo, também chamado de Protocolo *Snoopy* de Coerência de Cache apresenta as seguintes características (asinale as alternativas corretas):

(a) (X) O controlador de cache fica escutando o bus compartilhado.

(b) ( ) Uma tabela com todas as entradas do Cache é mantida e controlada para todos os processadores e seus caches.

(c) ( ) Quando uma transação é relevante, isto é, se afeta algum bloco que está naquele cache, o controlador toma uma ação para assegurar a coerência.

(d) (X) Quando o controlador escuta alguma operação que modifica uma variável que está com ele, a tabela central é atualizada para manter o valor coerente.

(e) (X) Uma das ações possíveis do controlador quando ele detecta uma transação relevante é invalidar a entrada do cache.

5. (1 pt(s)) A cláusula `reduction` no OpenMP

○ É utilizada para se reduzir o tamanho do código, geralmente entre 20 e 30%.

- ○ É utilizada para se conseguir um modelo de exclusão mútua, reduzindo-se o risco de resultados inconsistentes.
- ● É utilizada para combinar múltiplas cópias locais de uma variável em diferentes threads, em uma única cópia na thread master.
- ○ É utilizada para se reduzir (diminuir) o valor de uma variável, aplicando-se sempre um operador soma aos resultados.

6. (1 pt(s)) Assinale **V**erdadeiro ou **F**also nas alternativas abaixo:
   (a) (F) O MPI é uma especificação (*Message Passing Interface*) para ser utilizada exclusivamente em computadores de memória compartilhada.
   (b) (V) A função MPI_Scatter é do tipo comunicação coletiva, e envia somente os pedaços de dados necessários para cada um dos processos do comunicador.
   (c) (F) Todos os processos do comunicador devem sempre chamar a função MPI_Gather para coletar os dados enviados pelos outros processos.
   (d) (V) Quando utilizamos o MPI, estamos utilizando o paradigma SPMD (Single-Program Multiple-Data).
   (e) (F) A funço MPI_Bcast é do tipo comunicação ponto-a-ponto, e espalha os dados pertencentes a um único processo a vários processos do comunicador.
   (f) (V) Na comunicação coletiva, todos os processos no comunicador precisam chamar a mesma função coletiva.
   (g) (V) Em uma chamada de comunicação coletiva, cada processo pode preencher o dest_process com o seu destinatário.
   (h) (V) Em uma chamada de comunicação coletiva, as mensagens são casadas somente na base do comunicador e da ordem na qual elas são chamadas.
   (i) (F) Um processo pode responder uma chamada para MPI_Reduce utilizando a chamda MPI_Recv pois ele receberá somente sua parte dos dados
   (j) (V) Em uma grande tarefa no MPI só é possível utilizar o MPI_COMM_WORLD para enviar mensagens entre os processos, não sendo possível definir grupos menores de processos.

7. (1 pt(s)) Assinale **V**erdadeiro ou **F**also nas seguintes afirmativas:
   (a) (V) Quanto maior o grau de concorrência menor a granularidade das tarefas.
   (b) (V) O comprimento do Caminho Crítico representa o tempo máximo que se obterá para a execução do algoritmo paralelo.
   (c) (V) O Grau de Concorrência é o número de tarefas que podem executar em paralelo em um determinado momento.
   (d) (F) O Grau de Concorrência é obtido ao se analisar o grafo de dependência das tarefas e é mantido constante durante a execução.
   (e) (V) O Caminho Crítico no gráfico de dependências é o caminho mais longo ponderado através do grafo.
   (f) (V) Em um grafo de dependência de tarefas, um caminho crítico mais curto favorece um mais alto grau de concorrência.

8. (2 pt(s)) Mapear tarefas a processos é crítico para o desempenho paralelo. Considere a multiplicação de uma matriz densa A (15x5) por um vetor (5x1). Cada operação de leitura de uma linha da matriz ou do vetor (carga dos dados) representa uma unidade de trabalho; cada operação aritmética (multiplicação, soma, etc), incluindo o armazenamento do resultado, representa uma unidade de trabalho da tarefa; a apresentação de cada elemento do resultado representa uma unidade de trabalho.
   (a) Construa o Grafo Acíclico Direcionado (DAG) de dependência de tarefas, com granularidade 3 elementos de Y (vetor resultado) por tarefa, considerando uma (ou mais) tarefa(s) a leitura dos dados e uma tarefa a apresentação dos resultados.
   (b) Qual é o comprimento de caminho crítico para este grafo?

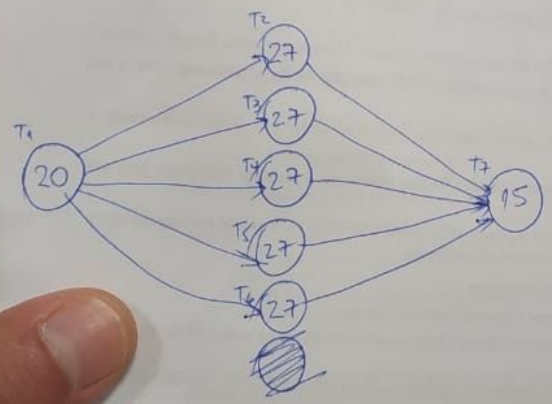
a) T_1: leitura dos dados { matriz A: 15 linhas: 15 ut; vetor B: 5 linhas: 5 ut } > 20 ut

T_2: multiplicação 3 linhas: $y[0] = a_{11} \times b_1 + a_{12} \times b_2 + a_{13} \times b_3 + a_{14} \times b_4 + a_{15} \times b_5$ (1 2 3 4 5 6 7 8 9 operações)

3 linhas $\Rightarrow 3 \times 9 = 27$ ut

$T_3 \dots T_6$: 27 ut

T_7: apresentação: 15 elementos: 15 ut



b) Comprimento caminho crítico

$20 + 27 + 15 = 62$

Semestre: 2º Sem 2018 | Nota:

Prof. Mário Olímpio de Menezes

Instruções: A prova deve ser feita sem consulta. O entendimento faz parte da prova. As respostas devem ser assinaladas a CANETA. **Desligue o seu celular durante a realização da prova!**.

Esta prova tem 11 questões. O total de pontos é 10.

1. (1 pt(s)) Considere as seguintes afirmações:

   I. O modelo de programação do MPI é o SPMD, já que o mesmo programa vai rodar nas diversas máquinas do cluster.

   II. MPI é um paradigma de programação paralela destinado a sistemas de memória compartilhada, onde é possível fazer a troca de mensagens por buffers compartilhados entre os processos.

   III. MPI deve ser utilizado sempre que se deseja construir sistemas de processamento em placas gráficas (GPUs) porque ele utiliza as mais modernas tecnologias de troca de mensagens.

   IV. Para distinguir qual parte do código deve ser executada pelos processos trabalhadores, uma simples cláusula `if` é suficiente. O `rank` do processo no Communicator identifica se é master (0) ou slave ($\neq 0$).

   V. No MPI, o conceito de Comunicadores é o de um grupo de processos que trocam mensagens entre si para a realização de determinada tarefa.

   Assinale a alternativa correta:

   - ○ As afirmativas II e I estão corretas.
   - ○ As afirmativas III e V estão erradas.
   - ● As afirmativas I e V estão corretas, mas a alternativa IV está errada.
   - ○ As afirmativas IV e V estão corretas, juntamente com a afirmativa I que também está correta.

2. (1 pt(s)) Considere o seguinte trecho de código e responda:

```
fibo[0] = fibo[1] = 1;
#pragma omp parallel for num_threads(2)
  for (i = 2, i < n; i++)
     fibo[i] = fibo[i-1] + fibo[i-2];
```

   - ○ A execução deste código ocorrerá sem problemas com quaisquer números de threads.
   - ● A execução deste código poderá gerar um resultado incorreto por conta da dependência entre as iterações, isto é, `fibo[i]` depende do valor de `fibo[i-1]` e `fibo[i-2]`.
   - ○ Para que este código execute corretamente, é preciso adicionar uma cláusula `private(i)`, para dividir corretamente o trabalho entre as threads.
   - ○ Uma medida que pode ser utilizada neste código para que ele execute sem problemas é a utilização da cláusula `private(fibo)`.

3. (½ pt(s)) A cláusula `reduction` no OpenMP
   - ○ É utilizada para se reduzir o tamanho do código, geralmente entre 20 e 30%.
   - ○ É utilizada para se conseguir um modelo de exclusão mútua, reduzindo-se o risco de resultados inconsistentes.
   - ● É utilizada para combinar múltiplas cópias locais de uma variável em diferentes threads, em uma única cópia na thread master.
   - ○ É utilizada para se reduzir (diminuir) o valor de uma variável, aplicando-se sempre um operador soma aos resultados.

4. (1 pt(s)) Assinale **V**erdadeiro ou **F**also nas seguintes afirmativas:
   (a) (F) O comprimento do Caminho Crítico representa o tempo máximo que se obterá para a execução do algoritmo paralelo.
   (b) (V) O Grau de Concorrência é o número de tarefas que podem executar em paralelo em um determinado momento.
   (c) (V) Quanto maior o grau de concorrência menor a granularidade das tarefas.
   (d) (V) O Caminho Crítico no gráfico de dependências é o caminho mais longo ponderado através do grafo.
   (e) (F) O Grau de Concorrência é obtido ao se analisar o grafo de dependência das tarefas e é mantido constante durante a execução.
   (f) (V) Em um grafo de dependência de tarefas, um caminho crítico mais curto favorece um mais alto grau de concorrência.

5. (1 pt(s)) Mapear tarefas a processos é crítico para o desempenho paralelo. Explique como deve ser feito este mapeamento utilizando os grafos de dependência de tarefas e de interações de tarefas.

Ao mapear tarefas e seus processo podemos traçar o grafo de interação dessas tarefas, onde o grau dessas interações representa as dependências entre essas tarefas, como por exemplo a multiplicação de uma matriz densa por um vetor

Uma das maneiras de se determinar o número Pi é a partir da série

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}$$

As questões 6 e 7 abordam implementações para esta soma de série.

6. (1 pt(s)) A versão serial da soma da série é:

```
double factor = 1.0;
double sum = 0.0;
for (i = 0; i < n; i++, factor = -factor){
     sum += factor/(2*i + 1);
}
pi = 4.0*sum;
```

Uma das maneiras de paralelizar este problema é dividir as iterações do loop `for` entre threads e fazer `sum` uma variável compartilhada. Esta divisão, para facilitar, será feita de modo exato, ou seja, cada thread receberá a mesma quantidade de trabalho. Uma versão do código da soma a ser executado pelas threads é mostrado a seguir:

```
void* Thread_sum(void* rank) {
    long my_rank = (long) rank;
    double factor;
    long long i;
    long long my_n = n/thread_count;
    long long my_first_i = my_n * my_rank;
    long long my_last_i = my_first_i + my_n;
```

```
    if (my_first_i % 2 == 0) /* my_first_i is even */
        factor = 1.0;
    else /* my_first_i is odd */
        factor = -1.0;
    for (i = my_first_i; i < my_last_i; i++, factor = -factor)
        sum += factor/(2*i+1);
    return NULL;
}  /* Thread_sum */
```

Utilizando este código, um aluno obteve os seguintes resultados:

	n			
	10^5	10^6	10^7	10^8
π	3.14159265	3.14159265	3.14159265	3.14159265
1 Thread	3.14158265	3.14159165	3.14159255	3.14159264
2 Threads	3.14168972	3.14099285	3.10734193	3.14292942

Ajude o aluno a entender os resultados com 2 threads, indicando **os problemas** no código bem como **as soluções necessárias**.

.....

.....

.....

.....

.....

7. (1 pt(s)) Considere o trecho abaixo, e responda:

```
double sum = 0.0;
ESCREVA AQUI O PRAGMA OpenMP
for (k = 0; k < n; k++) {
  if (k % 2 == 0)
     factor = 1.0;
  else
     factor = -1.0;
  sum += factor/(2*k + 1);
}
```

Escreva o pragma correto que deve ser colocado no local indicado no código para que ele funcione corretamente.

#pragma omp parallel for num_threads(Thread_count) reduction (+: sum) private (factor)

8. (1 pt(s)) Considere um vetor **v** de 16 elementos que precisa ser processado por 4 threads (T0, T1, T2, T3). Assinale a alternativa correta com relação ao esquema de divisão do trabalho no OpenMP utilizando os escalonamentos (schedules).

- ○ Utilizando-se `schedule(static,2)`, $v[13]$ será processado por T3 na última rodada, $v[3]$ será processado por T2 na primeira rodada e $v[6]$ será processado por T1 na terceira rodada.
- ○ Utilizando-se `schedule(static,4)`, $v[13]$ será processado por T1 na última rodada, $v[3]$ será processado por T1 na primeira rodada, $v[6]$ será processado por T1 na terceira rodada.
- ○ Utilizando-se `schedule(static,2)`, $v[13]$ será processado por T3 na última rodada, $v[3]$ será processado por T1 na segunda rodada e $v[6]$ será processado por T0 na segunda rodada.
- ● Utilizando-se `schedule(static,1)`, $v[13]$ será processado por T1 na última rodada, $v[3]$ será processado por T3 na primeira rodada e $v[6]$ será processado por T2 na segunda rodada.

9. (½ pt(s)) Para que as operações de envio e recebimento de mensagens funcionem no MPI, alguns campos das funções MPI_Send e MPI_Recv devem ser casados, isto é, devem ter valores correspondentes. Selecione a alternativa que indica os campos que obrigatoriamente devem ter valores casados.

- ○ Função send/recv e rótulo da msg.
- ○ Communicator, rótulo da msg, função de broadcast, tipo de dado.
- ● Tipo de dado enviado/recebido, emissor/receptor, rótulo da msg, communicator.
- ○ Rótulo da msg, função de redução, communicator, emissor/receptor.

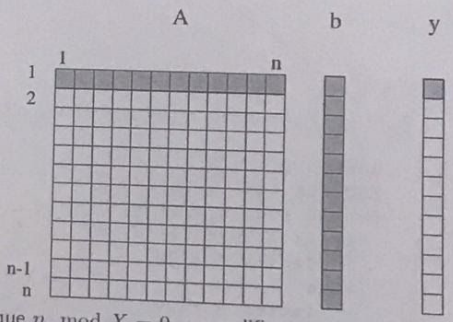
P2 CompPar

FCI – Mackenzie

Prof. Mário Menezes

10. (1 pt(s)) Assinale **V**erdadeiro ou **F**also nas alternativas abaixo:
    (a) (F) A funço MPI_Bcast é do tipo comunicação ponto-a-ponto, e espalha os dados pertencentes a um único processo a vários processos do comunicador.
    (b) (V) Na comunicação coletiva, todos os processos no comunicador precisam chamar a mesma função coletiva.
    (c) (F) Em uma chamada de comunicação coletiva, cada processo pode preencher o `dest_process` com o seu destinatário.
    (d) (V) Em uma chamada de comunicação coletiva, as mensagens são casadas somente na base do comunicador e da ordem na qual elas são chamadas.
    (e) (F) O MPI é uma especificação (*Message Passing Interface*) para ser utilizada exclusivamente em computadores de memória compartilhada.
    (f) (V) A função MPI_Scatter é do tipo comunicação coletiva, e envia somente os pedaços de dados necessários para cada um dos processos do comunicador.
    (g) (F) Todos os processos do comunicador devem sempre chamar a função MPI_Gather para coletar os dados enviados pelos outros processos.
    (h) (V) Quando utilizamos o MPI, estamos utilizando o paradigma SPMD (Single-Program Multiple-Data).
    (i) (F) Um processo pode responder uma chamada para MPI_Reduce utilizando a chamda MPI_Recv pois ele receberá somente sua parte dos dados
    (j) (V) Em uma grande tarefa no MPI só é possível utilizar o MPI_COMM_WORLD para enviar mensagens entre os processos, não sendo possível definir grupos menores de processos.

11. (1 pt(s)) Considere a multiplicação de uma matriz densa por um vetor, conforme ilustrado na figura ao lado, e responda.



Se tivermos um número X de tarefas (Tasks) e considerando que $n \mod X = 0$, exemplifique como se pode obter granularidades *fine-grain* e *coarse-grain* nesta multiplicação.

FINE GRAIN SÃO grandes nºs DE TAREFA E COARSE GRAIN SÃO pequeno nºs de tarefas. Multiplicando a matriz densa pelo vetor, quando cada tarefa representar um elemento individual em y podemos obter um FINE-GRAIN E quando cada tarefa computa vários elementos em y podemos obter um COARSE-GRAIN

UNIVERSIDADE PRESBITERIANA MACKENZIE
– Faculdade de Computação e Informática –

Ciência da Computação
Programação Paralela
Prova Parcial 1 – 20150928

Nome do Aluno: GABARITOVSKI		Matrícula: 1111110-1
Turma:	Semestre: 2º Sem 2015	Nota: 10.0
Assinatura: Esse é o cara!		

Instruções: A prova deve ser feita sem consulta. O entendimento faz parte da prova. As respostas podem ser a lápis. Seja claro e preciso em suas respostas. **Desligue o seu celular durante a realização da prova!**

Esta prova tem 10 questões. O total de pontos é 10.

Questão 1 (1½ pt(s))
Um dos problemas típicos onde se demanda computação de alto desempenho é a previsão do tempo. Um problema deste tipo necessita tipicamente 10^{12} operações para o cálculo de todas as células. Suponha que ao rodar o problema em um cluster de 100 nós de computação (1 processador de 1 core por nó), uma execução sequencial (sem nenhum paralelismo) tenha demorado aproximadamente 10 dias. Indique se as afirmações abaixo são verdadeiras ou falsas, justificando as respostas:

(a) Se o problema tiver uma fração paralelizável de 80%, pode-se conseguir um speedup maior do que 4.

Solução:

V
$$S(p) = \frac{P}{1+(p-1)f} \qquad S(100) \approx \frac{100}{20,8}$$
$$S(100) = \frac{100}{1+(100-1)\times 0,2} \qquad S(100) \approx 4,807$$

(b) Se a fração paralelizável deste problema for de 95%, mesmo assim, o speedup máximo não passará de 20.

Solução:
V $S(100) = \frac{100}{1+(100-1)\times 0,05} = \frac{100}{5,95} \approx 16,8$

(c) Se o cluster tivesse 200 nós, a execução sequencial seria mais rápida.

Solução:
F – A execução sequencial não usará mais do que um nó, portanto, 200 nós não farão diferença.

Questão 2 (1 pt(s))
Temos duas maneiras básicas de implementar paralelismo: de tarefas e de dados. Defina, sucintamente, estas duas maneiras de paralelismo.

Solução:

Paralelismo de Tarefas: Divide o problema em várias tarefas que são distribuídas entre os núcleos

Paralelismo de Dados: Divide os dados do problema entre os núcleos; cada núcleo executa tarefas similares na [illegible] dos dados.

que cada um obteve os seguintes resultados de soma dos trechos que eram responsáveis: 8, 19, 7, 15, 7, 13, 12, 14. Cada core recebeu 123 elementos para somar.

(a) Quantas operações (adição) no máximo realizará o nó mestre para computar a soma global?

Solução:
Para somar os 123 elementos iniciais, foram realizadas 122 operações. Para somar os resultados intermediários recebidos dos outros cores, mais 7 operações, assim, no total, o nó mestre realizará **129** operações

(b) E qual o valor mínimo de operações ele poderia realizar, se um algoritmo mais eficiente (balanceado) fosse utilizado.

Solução:
Para somar os 123 elementos inicis, foram realizadas 122 operações. Com um algoritmo balanceado, mais 3 operações seriam necessárias, resultando em **125** operações.

Questão 4 (1 pt(s))
O trecho de código em C abaixo implementa uma versão do problema do produtor-consumidor. Quando o buffer está vazio, o consumidor deve esperar que um item seja produzido, e o produtor deve analisar sua produção. Dadas as duas funções da biblioteca pthreads, indique em qual posição cada uma deve ser colocada para que o código funcione corretamente:

(a) (**3**) pthread_cond_signal(&c);

(b) (**6**) pthread_cond_wait(&nb,&c);

```
// Área de memória compartilhada
Buffer b;   // buffer de armazenamento temporário
int nb_itens = 0;  // contador de itens no buffer
pthread_mutex_t mb;   // proteção do buffer e do contador
pthread_cond_t c;     // sincronização produtor/consumidor

void Produtor() {
  Item it;
  for( ; ; ) {
    it = ProduzItem();
    pthread_mutex_lock(&mb);
(1)
    ArmazenaBuffer(b, it);
(2)
    nb_itens++;
(3)
    pthread_mutex_unlock(&mb);
(4)
  }
}

void Consumidor() {
  Item it;
  for( ; ; ) {
    pthread_mutex_lock(&mb);
(5)
    while( nb_itens == 0)
(6)
      it = LeBuffer();
(7)
    nb_itens--;
(8)
    pthread_mutex_unlock(&mb);
(9)
  }
}
```

Questão 5 (1 pt(s))
O modelo de computador de von Neumann estabeleceu um novo paradigma na computação da então ao propor que dados e instruções ficassem na mesma memória. Desta forma, a CPU era responsável por buscar a próxima instrução a ser executada e também os dados necesários para a operação. Muitas melhorias tem sido propostas à arquitetura básica de von Neumann. Dentre elas podemos citar:

- ○ Emprego de discos de estado sólido (SSD) para melhorar o desempenho
- √ **Adição dos caches para armazenar dados e instruções mais utilizados**
- ○ Adição de placas de rede para comunicação externa
- ○ Criação de clusters de computadores para processamento paralelo
- ○ Utilização do modelo de computação em nuvem

Questão 6 (½ pt(s))
Marque **(V)** ou **F** nas afirmações abaixo

(a) **(V)** Arquiteturas do tipo SIMD realizam operações em vetores/matrizes muito bem.

(b) **(F)** As GPUs (Unidades Gráficas de Processamento) são baseados no estilo de paralelismo MIMD

(c) **(V)** Nas arquiteturas do tipo SIMD, a utilização recursos e o desempenho sofrem se a computação não for totalmente paralelizada.

Revisão Computação Paralela
Perguntas da Prova 1:

1) Um dos problemas típicos onde se demanda computação de alto desempenho é a previsão do tempo. Um problema deste tipo necessita tipicamente 1012 operações para o cálculo de todas as células. Suponha que ao rodar o problema em um cluster de 100 nós de computação (1 processador de 1 core por nó), uma execução sequencial (sem nenhum paralelismo) tenha demorado aproximadamente 10 dias. Responda:

a) Se o problema tiver uma fração paralelizável de 80%, pode-se conseguir um speedup maior do que 4?

Fórmulas: (Speedup)
n/(1+(n-1)f). Onde "n" é a quantidade de "nós" e "f" é a fração NÃO paralelizável do problema.
OU
1/[(1-p)+(p/n)]. Onde "n" é a quantidade de "nós" e "p" é a fração PARALELIZÁVEL do problema.

R: Speedup de 4,8. Então, sim é possível.

b) Se a fração paralelizável deste problema for de 95%, qual será o speedup máximo?

R: O Speedup máximo é de 16,8.

c) Se o cluster tivesse 200 nós, a execução sequencial seria mais rápida?

R: Não, pois se a execução é sequencial, não é possível paralelizar.

d) A paralelização deste problema para este cluster deverá utilizar qual modelo de comunicação?

R:

2) Temos duas maneiras básicas de implementar paralelismo: de tarefas e de dados. Defina, sucintamente, estas duas maneiras de paralelismo:

R:
Paralelismo de Tarefas: Divide o problema em várias tarefas que são distribuídas entre os núcleos.

Paralelismo de Dados: Divide os dados do problema entre os núcleos, cada núcleo executa tarefas similares na produção (ou no tratamento, ou na execução) dos dados.

3) Um computador com 8 cores precisa fazer diversas operações de soma. Cada core recebeu 123 elementos para somar.

a) Quantas operações (adição) no máximo realizará o nó mestre para computar a soma global?

R: Para somar os 123 elementos iniciais foram realizadas 122 operações. Para somar os resultados intermediários recebidos dos

outros cores, será necessário mais 7 operações. Assim, no total, o nó mestre realizará 129 operações.

b) E qual o valor mínimo de operações ele poderá realizar, se um algoritmo mais eficiente (balanceado) fosse utilizado?

R: Para somar os 123 elementos iniciais foram realizadas 122 operações. Com um algoritmo balanceado, mais 3 operações seriam necessárias (Já que paralelamente os outro núcleos estão tratando das outras operações), resultando em 125 operações.

4) Onde colocar os trechos de código para um semáforo (Exclusão Mútua). Respostas, começo no espaço **3** do código e o final no espaço **6** do código.

5) O modelo de computador de Von Neumann estabeleceu um novo paradigma na computação de então, ao propor que dados e instruções ficassem na mesma memória. Desta forma, a CPU era responsável por buscar a próxima instrução a ser executada e também os dados necessários para a operação. Muitas melhorias têm sido propostas à arquitetura básica de Von Neumann. Dentre elas podemos citar:

R: Adição dos caches para armazenar dados e instruções mais utilizados.

6)
(V) Arquiteturas do tipo SIMD realizam operações em vetores/matrizes muito bem.

(F) As GPUs (Unidades Gráficas de Processamento) são baseadas no estilo de paralelismo MIMD.

(V) Nas arquiteturas do tipo SIMD, a utilização recursos e o desempenho sofrem se a computação não trivialmente paralelizada.

Revisão Computação Paralela
Perguntas da Prova 1 (ou Sub?):

1) Concorrência. O que é?
2) Explicar sobre Paralelismo, pra que serve? Técnicas mais rápidas? Processamento mais não sei o que...
3) NUMA com Cluster
4) Técnicas de Paralelismo: Pipeline, Execução de processos Multi-Process...
5) Mutex??? Tinha código, qual o código certo para Mutex??? (Mutex = Exclusão Mútua)
6) T0 T1 T2 T3 com schedule(static, ...)
7) Memória Cache
7) V ou F
8) V ou F (MPI)
9) V ou F (Granularidade, Grau de Concorrência e Canais Críticos)
10) Tarefas. Instrução de Tarefas e Tarefas Independentes.
11) Matriz Densa. Fine-grain e Coarse-grain

Revisão Computação Paralela
Perguntas da Prova 2:

1) Divisão de trabalho e escalonamentos (schedules). schedule(static, 2) separa o vetor de 2 em 2 entre as Threads.

2) OpenMP é uma API para o desenvolvimento de programação paralela em sistemas de memória compartilhada.

3) O OpenMP utiliza diretivas do compilador para o pré-processamento do código fonte. Uma linha de código típica para iniciar um trecho de código paralelo com OpenMP é #pragma omp parallel num_threads(thread_count).

4) A execução do código Fibonacci poderá gerar um resultado incorreto por conta da dependência entre as iterações do laço.

5) #pragma omp parallel for num num_threads(thread_count) reduction(+:sum) private (factor).
Porque sum precisa ser atualizado sempre globalmente, e factor precisa ser private para que diversas Threads não alterem seu valor em outras Threads.

6) A cláusula Reduction no Open MP é utilizada para combinar múltiplas cópias locais de uma variável em diferentes threads, em uma única cópia na Thread Master.

7) Para que as operações de envio e recebimento de mensagens funcionem no MPI, alguns campos das funções MPI_Send e MPI_Recv

devem ser casados, isto é, devem ter valores correspondentes. Os campos que obrigatoriamente devem ter valores casados são:
- Tipo de Dado Enviado/Recebido
- Emissor/Receptor
- Rótulo da Mensagem
- Comunicador

8) Para iniciar um programa feito com MPI, sempre utilizamos uma chamada MPI_Init(NULL, NULL). Depois, se queremos saber quantos processos fazem parte do grupo de comunicação, devemos utilizar MPI_Comm_size(MPI_COMM_WORLD & tamk).

9)
(V) I. O modelo de programação do MPI é o SPMD, já que o mesmo programa vai rodar nas diversas máquinas do Cluster.

(F) II. MPI é um paradigma de programação paralela destinado a sistemas de memória comparitlhada, onde é possível fazer a troca de mensagens por buffers [?] compartilhados entre os processos. [O MPI na verdade é uma API destinada a Sistemas de Memória Distribuída]

(F) III. MPI deve ser utilizado sempre que se deseja construir sistemas de processamento em placas gráficas (GPUs) porque ele utiliza as mais modernas tecnologias de troca de mensagens.

(V) IV. Para distinguir qual parte do código deve ser executada pelos processos trabalhadores, uma simples cláusula if é o suficiente. O rank do processo no Communicador identifica se é master (0) ou slave ($\neq$ 0).

(V) V. No MPI, o conceito de Comunicadores é o de um grupo de processos que trocam mensagens entre si para a realização de determinada tarefa.

As respostas I, IV e V estão corretas

10) A função deve ter sido chamada como rr_v(local_a, 10, 10, 200, my_rank, comm)

11) O LIMITE do vetor é 5000, e o número de threads é 8: 5000/8 ≈ 625. Logo, cada thread processará, **no total**, ≈ 625 elementos para operar.

Outra prova:

1) Igual Questão 9.
2) igual Questão 4.
3) Igual Questão 6.

4)
(F) O comprimento do Caminho Crítico representa o tempo máximo que se obterá para a execução do algoritmo paralelo.
(V) O Grau de Concorrência é o número de tarefas que podem executar em paralelo em um determinado momento.
(V) Quanto maior o Grau de Concorrência, menor a Granularidade das tarefas.

(V) O Caminho Crítico no gráfico de dependências é o caminho mais longo ponderado através do grafo.
(F) O Grau de Concorrência é obtido ao se analisar o grafo de dependência das tarefas e é mantido constante durante a execução.
(V) Em um grafo de dependência de tarefas, um caminho crítico mais curto favorece um mais alto grau de concorrência.

5) Mapear tarefas a processos é crítico para o desempenho paralelo. Explique como deve ser feito este mapeamento utilizando os grafos de dependência de tarefas e de interações de tarefas. (NÃO RESPONDEMOS)

6) Parecido com a Questão 5. (REVISAR)

7) Igual a Questão 5.
8) Igual a Questão 1
9) Igual a Questão 7

10)
(F) A função MPI_Bcast é do tipo comunicação ponto-a-ponto, e espalha os dados pertencentes a um único processo a vários (todos) processos do comunicador. (MPI_Bcast é do tipo comunicação coletiva. uma comunicação do tipo ponto-a-ponto seria a MPI_Send e a MPI_Recv)

(V) Na comunicação coletiva, todos os processos no comunicador precisam chamar a mesma função coletiva.

(F) Em uma chamada de comunicação coletiva, cada processo pode preencher o dest_process com seu destinatário. (Isso seria comunicação ponto-a-ponto)

(V) Em uma chamada de comunicação coletiva, as mensagens são casadas somente na base do comunicador e da ordem na qual elas são chamadas.

(F) O MPI é uma especificação (Message Passing Interface) para ser utilizada exclusivamente em computadores de memória compartilhada. (Memória Distribuída é o certo).

(V) A função MPI_Scatter é do tipo comunicação coletiva, e envia somente os pedaços de dados necessários para cada um dos processos do comunicador.

(F) Todos os processos do Comunicador devem sempre chamar a função MPI_Gather para coletar os dados enviados pelos outros processos. [MPI_Gather serve para "recolher" os pedaços de dados, que foram "espalhados" através da função MPI_Scatter, enviados aos outros processos]

(V) Quando utilizamos o MPI, estamos utilizando o paradigma SPMD (Single Program Multiple-Data).

(F) Um processo pode responder uma chamada para MPI_Reduce utilizando a chamada MPI_Recv pois ele receberá somente sua parte dos dados. (MPI_Reduce e MPI_Recv não tem nada a ver um com o outro... MPI_Send e MPI_Recv sim)

(F) Em uma grande tarefa no MPI só é possível utilizar o MPI_COMM_WORLD para enviar mensagens entre os processos, não sendo possível definir grupos menores de processos.

11) Fine-grain são grandes números de tarefas e coarse-grain são pequenos números de tarefas.
Multiplicando a Matriz Densa pelo Vetor, quando cada tarefa computar um elemento individual em Y, podemos obter um fine-grain. Já quando multiplicando a Matriz Densa pelo Vetor, quando cada tarefa computa vários elementos em Y, podemos obter um coarse-grain.

Revisão Computação Paralela
Lista de Exercícios:

1: Explique quais são as razões básicas para a utilização de processamento paralelo.

R:
(1)
Um aproveitamento melhor do Hardware e a diminuição do tempo de execução, aumentando o desempenho.
(2)
Resolver problemas maiores, uso efetivo de recursos de máquina, custo efetivo de energia.
(3)
- O tempo para encontrar uma solução tende a ser mais rápido, além de dar a possibilidade de resolução de problemas maiores em um tempo aceitável.
- Tendo em mente que a arquitetura e hardwares modernos são construídos pensando no processamento paralelo, utilizá-lo significa aproveitar de maneira mais eficiente os recursos disponíveis na máquina.
- É uma utilização mais eficiente de recursos como energia, tempo , processamento, etc.
- Pode ajudar no contorno de limitações de memória.

2: Quando analisamos a linha histórica da evolução do número de transistores nos chips (CPUs), uma das coisas notáveis é o gargalo no desempenho entre Memória e Processador. Enquanto a memória já atingiu níveis de desempenho altíssimos, com as memórias DDR6, a CPU ainda está nas famílias básicas dos processadores. Esta discrepância de desempenho é denominada “Gap de Performance Processador-Memória”, e cresce aproximadamente 50% ao ano. Analise esta frase, se está correta ou não, e caso não esteja, reescreva-a corretamente.

R:

(1)

Quando analisamos a linha histórica da evolução do número de transistores nos chips (CPUs), uma das coisas notáveis é o gargalo no desempenho entre Memória e Processador. Enquanto os processadores já atingiram níveis de desempenho altíssimos, as memórias têm sofrido atraso por causa da comunicação enre as fronteiras dos chips . Esta discrepância de desempenho é denominada “Gap de Performance Processador-Memória”, e cresce aproximadamente 50% ao ano.

(2)

A frase está incorreta. Há sim um gargalo entre os desempenhos de Memória e Processador, e há sim um desempenho altíssimo presente nos tipos de memória mais recentes, mas a hierarquia de memória nunca mudou, isto é, memórias mais rápidas sempre serão mais custosas. Há memórias mais rápidas disponíveis, mas em menores quantidades. O gargalo existe entre a Memória e o Processador é a quantidade de memória disponível para ser usada pelo processador de

forma que isto não gere um gargalo no tráfego de dados. Isto é, o desafio é encontrar formas mais eficientes de organização de memória para garantir uma maior eficiência de uso da mesma, pois a memória mais rápida ainda é escassa que um processamento mais rápido.

3. Explique o que é um Sistema denominado NUMA (Non-Uniform Memory Access), ou Acesso Não Uniforme à Memória, descrevendo suas principais características em comparação a um sistema de acesso uniforme à memória.

R:
- Um sistema SMP consiste na ideia de múltiplos processadores consumindo uma mesma memória, e sendo administrados por um mesmo sistema operacional. Isto, em teoria, é o ideal para o processamento de dados concorrentes, mas não bloqueantes, isto é, dados que podem ser compartilhados entre processos, mas não necessariamente são codependentes. Uma limitação desta solução, por um lado, é quando a quantidade de processadores aumenta, podendo ocasionar em um congestionamento no tráfego de dados.
- Para solucionar este problema, NUMA trabalha com o conceito de memory bus, adicionando uma camada a mais para as já conhecidas hierarquias de memória, cache e principal. Desta forma, um grupo de processadores pode consumir memória de um mesmo bus, quando os dados não estiverem presentes em uma memória cache, ao invés de terem que ir direto na memória principal. Os bus são locais, por grupos de processadores, e não necessariamente compartilhados.
- Essa solução pode resultar em uma complexidade a mais, já que implica em memórias estão sendo replicadas em bus diferentes,

memórias que podem ser alteradas em outros locais, problemas de sincronização, etc.

4. Liste as técnicas de paralelismo em nível de instrução.

R:
- Pipelining: A ideia é que, múltiplas instruções possam ser empilhadas, em estados diferentes de execução, otimizando a utilização do processador. Isto é, para um conjunto qualquer de instruções, em estados diferentes, o processador consegue avançar os estados de forma independente, quando os processos são independentes.
- Superscalar
- Out-of-order: A ideia é de que este tipo de técnica pode ser incorporado por outros tipos de técnicas de paralelismo em nível de instrução, e consiste no ato de executar blocos de instrução fora de ordem, otimizando o uso do processador. Instruções dentro de processos que não são dependentes e etc.
- Register remaining: Uma técnica que consiste eliminar falsos positivos em processos dependentes.
- Speculative Execution: Quando o processo é executado de forma especulativa, isto é, há uma boa chance daquele processo ser executado no futuro, então sua execução é adiantada e seu resultado é armazenado. Caso o mesmo não venha acontecer, o resultado é só ignorado.
- Branch Prediction

5. Uma das técnicas para se conseguir paralelismo é a adição de mais processadores a um computador. Indique quais são as arquitetura possíveis quando se tem mais de um processador em um computador.

R: Arquiteturas como MIMD (Multiple Instruction, Multiple Data) e MISD(Multiple Instruction, Single Data), onde a primeira é a mais presente nos sistemas contemporâneos e a segunda não é utilizada.

6. Um dos conceitos importantes quando estudamos Computação Paralela é o de Concorrência. Explique o que é Concorrência e quais são os requisitos para consegui-la.

R: Concorrência é o conceito de múltiplas tarefas sendo executadas em um computador. Tarefas são concorrentes entre si se, e somente se, podem ser executadas simultaneamente, ou seja, quando não há dependências entre as tarefas.

7. Considere o sistema apresentado na figura 1 e responda quais são os valores corretos de A e B para que este sistema seja um sistema de memória distribuída, tipo cluster. Justifique sua resposta.

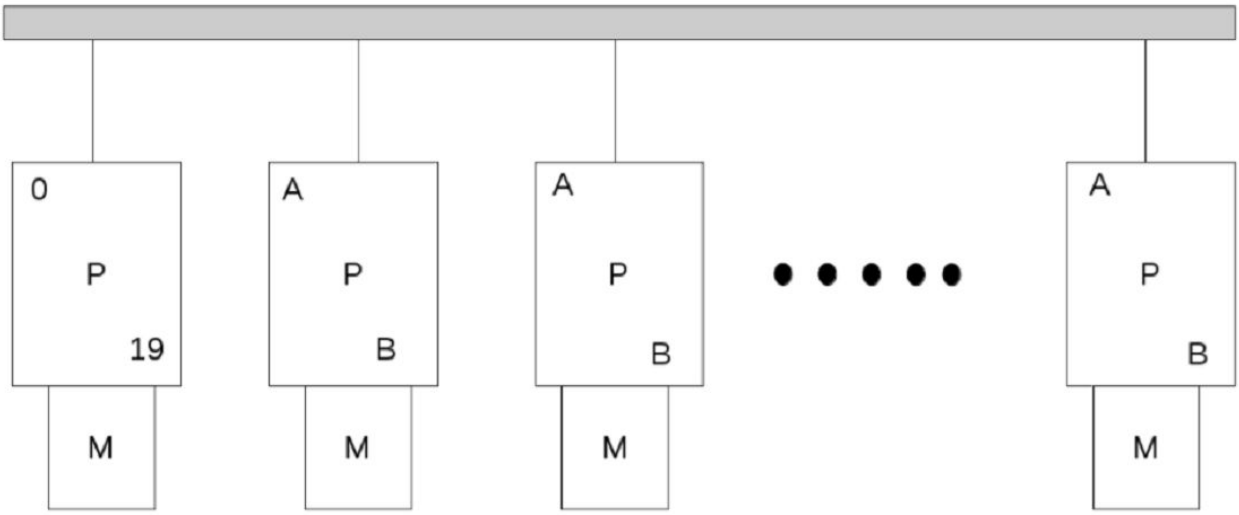


Figura 1: Sistema de Memória distribuída

R: Em um sistema de Memória Distribuída os valores de A e B não são alterados, ou seja A=0 e B=19, para todos dentro do Cluster. Já, se fosse um sistema de Memória Compartilhada, os valores poderiam ser diferentes uns dos outros.

8. Um computador que tem apenas uma unidade de execução consegue executar tarefas concorrentes em paralelo, já que a única restrição é que elas sejam concorrentes? Justifique:

R: Sim, pois executar tarefas concorrentes não implica na execução dela em múltiplos processadores. Você pode ter, por exemplo, uma estrutura de pipeline para satisfazer a concorrência destas tarefas

9. Quais são os modelos de comunicação possíveis de serem utilizados em sistemas de memória compartilhada? E em um sistema de memória distribuída? Discorra sobre este tópico (comunicação entre tarefas).

R: Em um sistema de memória compartilhada as comunicações podem ser feitas diretamente entre múltiplos processadores e um barramento de memória cache única, no caso de UMA, ou entre múltiplos processadores e múltiplos barramentos de memória cache locais. Nos dois casos, os processadores se comunicam diretamente com um sistema de memória.
Em um sistema de memória distribuída, há diversas formas de comunicação entre processadores. Há comunicação pela rede, limitada pela largura da banda e latência, há hardware específico baseado no controle de tráfego de dados, barramentos adicionais entre sistemas

especializados no gerenciamento deste tráfego. Basicamente, há um hardware específico servindo como ponte de dados, realizando a comunicação entre os nós do sistema.

10. O emprego de Memória Cache em sistemas de memória compartilhada auxilia na redução da latência no acesso à memória, já que a informação é trazida para mais perto de cada processador. Entretanto esta tecnologia também traz consigo algumas dificuldades. Discorra sobre estas dificuldades, explicando o que acontece:

R: Um dos maiores problemas com a utilização da memória Cache em sistemas distribuídos é a coerência de Cache, isto é, a replicação de um dado específico em um bloco de memória e em uma parte da memória Cache. No momento que um processo modifica este dado na memória Cache o dado original, presente na memória principal, se torna desatualizado.

11. Uma das maneiras de se prover Coerência de Cache é através do uso de um mecanismo des “bisbilhotagem”. Este mecanismo, também chamado de Protocolo Snoopy de Coerência de Cache. Explique como ele funciona, apresentado um exemplo e a sequência de ações que acompanha:

R: Quando um dado específico é compartilhado em diversos caches e um processador modifica este dado, a mudança tem que ser propagada através de todos os outros caches. Uma forma de propagar essa mudança é através da técnica de bisbilhotagem, que consiste na ideia de um cache controller monitorar os estados de um dado compartilhado, observando sua mudança de estados. Cada linha de

uma memória cache possui um dos seguintes estados: dirty, valid, invalid e shared e um valor, que pode ser escrito ou lido. Todo novo bloco inserido na memória cache entra com o estado valid. Em uma falha de leitura no cache local, a solicitação de leitura é transmitida no barramento. Todos os cache controllers realizam o monitoramento do barramento. Se alguém tiver armazenado esse endereço no seu cache local, e o seu estado for dirty, ele realizará a alteração do estado para valid e enviará uma cópia para o nó solicitante. O estado valid implica que a linha de cache está atualizada. Em uma falha de gravação local, a detecção de barramento pelos cache controllers garantem que quaisquer cópias em outros caches tenham seu estado alterado para invalid. O estado invalid implica que uma cópia desse dado costumava existir no cache, mas não está atualizada. A vantagem deste método é que é mais rápida que outros mecanismos de coerência de cache, mas só quando há uma largura de banda suficiente para processar o grande número de requisições de atualização vistos por todos os processadores. A desvantagem é em relação a sua escalabilidade, pois o acesso frequente na memória cache resultado dos mecanismos de sincronização podem resultar em condições de corrida, aumentando assim o tempo de acesso destes dados e o uso de memória subsequente. Cada requisição precisa ser transmitida para todos os nós do sistema, todos os cache locais, o que significa que o barramento interno precisa comportar a quantidade de nós, o que pode ser um problema em arquiteturas NUMA.

12. Considere o trecho de código abaixo utilizado por um programador para permitir que dois processos troquem informações por uma variável compartilhada. Analise o código e indique se está correto. Caso não esteja correto, indique o que deveria ser feito para corrigi-lo.

```
#i n cl u d e <s t d i o . h>
```

```
#include<unistd.h>/*containsforkprototype*/

intcounter=0;
voidfunctionC(int);

intmain(void){
        intpid,i;
        intvetor[10];
        pid=fork();
        if(pid==0){
                printf("Acessandoovetornoprocessofilho\n");
                for(i=0;i<10;i++)functionC(vetor[i]);
                printf("Counter:%d\n",counter);
        }
        else{
                printf("Inicializandoovetornoprocessopai\n");
                for(i=0;i<10;i++)vetor[i]=i+3;
        }
}

voidfunctionC(intj)
{
counter+=j;
}
```

R:

13. Um programador realiza alguns testes de desempenho em um código de multiplicação de matrizes e percebe que há um aumento muito grande no número de “cache miss” ao utilizar o trecho abaixo.

```
for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                a[j][i]=i*j;
```

Qual é o problema que este programador está enfrentando? Qual a solução?

R: Deveria ser a [i] [j]= i * j ; ao invés de a [j] [i]= i * j ;

14. Um dos problemas típicos onde se demanda computação de alto desempenho é a previsão do tempo. Um problema deste tipo necessita tipicamente 1012 operações para o cálculo de todas as células. Suponha que ao rodar o problema em um cluster de 100 nós de computação (1 processador de 1 core por nó), uma execução sequencial (sem nenhum paralelismo) tenha demorado aproximadamente 10 dias. Responda:

(a) Se o problema tiver uma fração paralelizável de 80%, pode-se conseguir um speedup maior do que 4?

Fórmulas: (Speedup)
n/(1+(n-1)f). Onde "n" é a quantidade de "nós" e "f" é a fração NÃO paralelizável do problema.
OU
1/[(1-p)+(p/n)]. Onde "n" é a quantidade de "nós" e "p" é a fração PARALELIZÁVEL do problema.

R: Speedup de 4,8. Então, sim é possível.

(b) Se a fração paralelizável deste problema for de 95%, qual será o speedup máximo?

R: O Speedup máximo é de 16,8.

(c) Se o cluster tivesse 200 nós, a execução sequencial seria mais rápida?

R: Não, pois se a execução é sequencial, não é possível paralelizar.

(d) A paralelização deste problema para este cluster deverá utilizar qual modelo de comunicação?

R: Um modelo de comunicação compatível com Sistemas de Memória Distribuída. Pode ser, Comunicação por rede ou utilizando um Hardware específico por exemplo.

15. A ideia básica utilizada para se criar programas paralelos é o particionamento do trabalho a ser realizado entre os cores. Duas abordagens são amplamente utilizadas para fazer este particionamento. Considere a seguinte situação: um professor P leciona para uma sala de 100 alunos e, ao final da disciplina, aplica um exame final que consiste de 5 questões. Este professor tem 4 assistentes (PA, PB, PC e PD). Explique, exemplificando com este problema apresentado, as duas abordagens de paralelização tradicionalmente utilizadas. Dica: os cores que executarão o trabalho são o professor P e seus assistentes, PA, PB, PC e PD, ou seja, 5 cores.

R: Há duas aproximações possíveis para este problema: paralelismo de tarefas e paralelismo de dados. Supondo que a coleção de professor e assistentes representam processadores, as questões de uma prova representam tarefas e a quantidade de provas em si, representam os dados, nós podemos realizar as seguintes atribuições: Cada processador pode ficar encarregado de realizar uma tarefa, ou seja, cada professor / assistente estaria encarregado de corrigir uma das cinco questões em todas as provas, fazendo referência ao paralelismo

de tarefas, ou cada professor / assistente poderia ficar encarregado de corrigir uma fração do número total de provas, representando o paralelismo de dados. Ou seja, no paralelismo de tarefas, cada um corrige 100 questões, mas somente uma das questões por prova, nesta situação, há a necessidade de sincronizar os dados com as outras questões corrigidas e pode gerar uma certa dependência, se feito de forma incorreta, afinal, um professor / assistente não pode corrigir mais de uma questão por vez e uma prova (um dado) não pode ser corrigido por mais de um professor assistente. No segundo caso, cada professor / assistente receberia vinte provas para corrigir, e estaria encarregado de corrigir todas as cinco questões. A segunda solução pode ser ruim caso um dos assistentes não esteja perfeitamente apto para corrigir todas as questões, mas o processo de sincronização de dados é bem mais simples, basta juntar o montante no final.

16. Quais são os passos típicos para se construir um algoritmo paralelo? Comente cada um.

R: (1) Slide:
Passos Típicos para se construir um algoritmo paralelo:
● identificar quais pedaços do trabalho podem ser feitas concorrentemente.
● particionar o trabalho concorrente em processadores independentes
● distribuir a entrada do programa, a saída e os dados intermediários
● coordenar os acessos aos dados compartilhados: evitar conflitos
● garantir ordem correta do trabalho utilizando sincronização
Por que “típicos”? Alguns dos passos podem ser omitidos
● se os dados estão em memória compartilhada, não é necessário distribuí-los

● se for utilizar MPI, pode não haver dados compartilhados
● o mapeamento do trabalho nos processadores pode ser feito estaticamente pelo programador ou dinamicamente pelo runtime.

(2) Tentativa de responder antes de ver o slide:
Uma maneira é dividir o problema em subproblemas menores do mesmo tipo e cada processador resolve um subproblema executando um algoritmo sequencial. Isso trata-se da Decomposição das Tarefas, que cada núcleo processará paralelamente. Mapear tarefas a processos é crítico para o desempenho paralelo, e isso pode ser feito através da utilização dos grafos de dependência de tarefas e de interações de tarefas. Onde no primeiro primeiro verificamos quais tarefas devem aguardar que outra tarefa seja finalizada antes, se existe deendência de dados, e no segundo verificamos quais tarefas possuem interações entre si, por exemplo, uma tarefa deseja acessar um recurso, que outra está usando. Duas analogias seriam, Ao lavar roupas, tenho roupas coloridas e brancas, não posso secar a roupa se ainda não a lavei, logo, secar roupa depende de lavar roupa. Ainda neste exemplo, eu possuo apenas uma máquina de lavar, logo a tarefa de lavar roupas coloridas, e lavar roupas brancas não podem ser executadas ao mesmo tempo, pois interagem com o mesmo recurso, a máquina de lavar, mas podem ser colocadas para secar ao mesmo tempo, neste momento ocorrendo paralelismo.

17. Na decomposição das tarefas, podemos representar a dependência entre as tarefas com um Grafo Acíclico Direcionado (DAG). Neste grafo

o que representam os nós e a arestas? O que representa o caminho mais longo neste grafo? E o seu comprimento?

R: Os nós representam as tarefas e as arestas representam as dependências. O caminho mais longo neste gráfico é considerado o Caminho Crítico, e o comprimento do caminho crítico equivale ao tempo mínimo que uma determinada execução pode demorar para finalizar (limite inferior para o tempo de execução paralelo.)

18. Na decomposição das tarefas podemos ter diversos tamanhos que geram diferentes granularidades. Explique estes conceitos.

R: Existes dois conceitos relacionados a Granularidade, o Fine-Grain e o Coarse-Grain. O Fine-Grain representa um grande número de tarefas, possívelmente menores, sendo executadas em paralelo, já no Coarse Grain, temos um número menor de tarefas, porém estas tarefas são possívelmente maiores. Por isso "grãos finos" e "grãos grossos".

19. O que é o grau de concorrência? Como podemos medir, ou seja, quais as métricas empregadas?

R: O Grau de Concorrência é o número de tarefas que podem executar em paralelo, e pode mudar durante a execução do programa. O Grau de Concorrência possui 2 métricas: o grau máximo de concorrência, que é o número máximo de tarefas concorrentes em qualquer ponto da execução, e o grau médio de concorrência, que é o número médio de tarefas que podem ser executadas em paralelo. Grau de Concorrência possui um relacionamento inver com a Granularidade das Tarefas.

20. Como as tarefas geralmente trocam dados (comunicação), esta interação também é representada por um grafo. Explique como é este grafo.

R: Este é o Grafo de Interação de Tarefas, e ele serve para mostrar quais tarefas comunicam-se entre si, onde os nós são as tarefas, e cada aresta é uma interação (troca de dados).

21. Como é o mapeamento de tarefas a processos?

R: Mapear tarefas a processos é crítico para o desempenho paralelo.

- Em que bases se deve escolher o mapeamento?
- utilizando grafos de dependência de tarefas
    - escalone tarefas independentes a processos separados
    - ociosidade mínima
    - balanceamento de carga ótimo
- utilizando grafos de interações de tarefas
    - queremos que os processos tenham uma interação mínima um com o outro
    - comunicação mínima

Um bom mapeamento deve minimizar o tempo de execução paralelo através de:

- Mapeamento de tarefas independentes a diferentes processos
- Atribuindo tarefas no caminho crítico a processos ASAP
- Minimização da interação entre processos
    - mapeia tarefas com interações densas ao mesmo processo.

Dificuldade: critérios frequentemente conflitam uns com os outros.

22. Na decomposição recursiva, uma característica desejável do problema é que as tarefas tenham comunicação mínima ou nula. Por quê?

R:Na decomposição Recursiva, buscamos a granularidade mínima, para que deste modo a dependência entre as sub-tarefas seja mínima ou nula, aumentando assim o grau de concorrência entre elas, aumentando o desempenho.

23. Quais os tipos de decomposição baseada em dados? Explique sucintamente cada um:

R: R:Na Decomposição baseada em Dados, primeiro identificamos os dados nos quais as computações são realizadas, depois particionamos os dados entre as várias tarefas. Para tanto, os dados podem ser particionados de várias maneiras

Dados de entrada: Quando a saída não é conhecida e a mesma é computada como uma função da entrada, particiona-se os dados de entrada entre as diversas tarefas, então cada uma realiza a computação na sua parte dos dados. O processamento subsequente combina os resultados parciais das tarefas anteriores
Dados de saída: Se cada elemento da saída pode ser calculado independentemente, particiona-se os dados de saída entre as tarefas. Então, cada tarefa realiza a computação para suas respectivas saídas saídas. Exemplo: multiplicação de matriz densa por vetor.
Dados de entrada + saída: Não tem no Slide.

Dados intermediários: Se a computação é uma sequência de transformações (de dados de entrada em dados de saída), pode-se decompor as tarefas baseado em dados necessários nas etapas intermediárias. Exemplo, enquanto na Decomposição de dados de saída, você separa uma tarefa para cada cálculo da saída, na decomposição de dados intermediários, você pode particionar inclusive estes cálculos em mais tarefas, por exemplo, na multiplicação de matrizes, vc multiplica os valores de uma linha da primeira matriz com os valores de uma coluna da outra, e depois disso você soma para obter o resultado, particionar os dados intermediários, é definir a tarefa de multiplicar a um processo, e a de somar a outro.

EXTRA: Decomposição Recursiva:

A Decomposição Recursiva utiliza a abordagem de Dividir e Conquistar

1. decomponha um problema em um conjunto de sub-problemas
2. recursivamente decomponha cada sub-problema

3. pare a decomposição quando a granularidade mínima desejada é alcançada

EXTRA: Cálculo do Grau Médio de Concorrência:

Total de trabalho / Comprimento do Caminho Crítico

Extra: Grafo de interação de tarefas vs grafo de dependência de tarefas:

- grafo de interação de tarefas representam dependência de dados
- grafo de dependência de tarefas representam dependências de controle

EXTRA: Qual a diferença entre Dependência e concorrência?

R: Dependência é quando um processo depende de algum recurso que outro processo possui ou está utilizando. Enquanto Concorrência, é quando dois processos estão sendo executados simultaneamente sem disputar os mesmos recursos.

EXTRA: Calcular tempo de computação no Grafo de Interação.

- Hipóteses:
  - cada nó leva uma unidade de tempo para processar
  - cada interação (aresta) causa um custo adicional de uma unidade de tempo.
- Se o nó 0 é uma tarefa: comunicação = 3; computação = 4

EXTRA: O que é Mutex?

Utilizado para proteger as estruturas de dados compartilhadas em modificações concorrentes e implementar seções críticas e monitores. Pode possuir dois estados, "unlocked" e "locked".

EXTRA: Paralelismo e Concorrência:

Paralelismo = Concorrência + Hardware “paralelo”

❑ Concorrente não é a mesma que paralelo! Por quê?

❑ Execução paralela

- ❍ Tarefas concorrentes podem de verdade executar ao mesmo tempo
- ❍ Múltiplos recursos (processamento) devem ser disponíveis

❑ Paralelismo = concorrência + hardware “paralelo” ❍ Ambos são necessários

- ❍ Encontrar oportunidades de execução concorrente ❍ Desenvolver a aplicação para executar em paralelo
- ❍ Rodar a aplicação em um hardware paralelo

❑ Estes elementos precisam ser suportados por recursos de hardware

- ❍ Processadores, cores, ... (execução de instruções)
- ❍ Memória, DMA, redes, ... (outras operações associadas)
- ❍ Todos os aspectos da arquitetura de um computador oferece oportunidades para a execução em hardware paralelo

❑ Concorrência é uma condição necessária para paralelismo

EXTRA: Taxonomia de Flynn:

Distingue arquiteturas multiprocessador em duas dimensões independentes

- ❍ Instrução e Dados
- ❍ Cada dimensão pode ter somente um estado: Single ou Multiple

❑ SISD: Single Instruction, Single Data

- ❍ Máquina Serial (não paralela)

❑ SIMD: Single Instruction, Multiple Data

- ❍ Arrays de processadores e máquinas vetoriais

❑ MISD: Multiple Instruction, Single Data (estranho)

- MIMD: Multiple Instruction, Multiple Data
  - Sistemas de computação paralela mais comuns

EXTRA: Coerência de Cache:

- Caches privados de cada processador criam um problema
  - Cópias de uma variável pode estar presente em múltiplos caches
  - Uma escrita por um processador pode não se tornar visível aos outros
    - Eles continuarão acessando o valor obsoleto em seus caches

É um problema mais sério em arquiteturas NUMA (Non-Uniform Memory Acces) do que nas arquiteturas UMA (Uniform Memory Acces)

- UMA:
  - Memória (não o cache) uniformemente equidistante
  - Leva o mesmo tempo (geralmente) para acessar
- NUMA:
  - Memória local e memória remota
  - Acesso à memória local é mais rápido
  - Acesso à memória remota é mais lento
  - Acesso não é uniforme
  - Desempenho dependerá da localidade dos dados

EXTRA: Protocolo Snoopy:

Quando um dado específico é compartilhado em diversos caches e um processador modifica este dado, a mudança tem que ser propagada através de todos os outros caches. Uma forma de propagar essa mudança é através da técnica de bisbilhotagem, que consiste na ideia

de um cache controller monitorar os estados de um dado compartilhado, observando sua mudança de estados. Cada linha de uma memória cache possui um dos seguintes estados: dirty, valid, invalid e shared e um valor, que pode ser escrito ou lido. Todo novo bloco inserido na memória cache entra com o estado valid. Em uma falha de leitura no cache local, a solicitação de leitura é transmitida no barramento. Todos os cache controllers realizam o monitoramento do barramento. Se alguém tiver armazenado esse endereço no seu cache local, e o seu estado for dirty, ele realizará a alteração do estado para valid e enviará uma cópia para o nó solicitante. O estado valid implica que a linha de cache está atualizada. Em uma falha de gravação local, a detecção de barramento pelos cache controllers garantem que quaisquer cópias em outros caches tenham seu estado alterado para invalid. O estado invalid implica que uma cópia desse dado costumava existir no cache, mas não está atualizada. A vantagem deste método é que é mais rápida que outros mecanismos de coerência de cache, mas só quando há uma largura de banda suficiente para processar o grande número de requisições de atualização vistos por todos os processadores. A desvantagem é em relação a sua escalabilidade, pois o acesso frequente na memória cache resultado dos mecanismos de sincronização podem resultar em condições de corrida, aumentando assim o tempo de acesso destes dados e o uso de memória subsequente. Cada requisição precisa ser transmitida para todos os nós do sistema, todos os cache locais, o que significa que o barramento interno precisa comportar a quantidade de nós, o que pode ser um problema em arquiteturas NUMA.

https://www.youtube.com/watch?v=uLikXssApZk&t=171s

EXTRA: Communicator:

As we have seen when learning about collective routines, MPI allows you to talk to all processes in a communicator at once to do things like distribute data from one process to many processes using MPI_Scatter or perform a data reduction using MPI_Reduce. However, up to now, we have only used the default communicator, MPI_COMM_WORLD.

For simple applications, it's not unusual to do everything using MPI_COMM_WORLD, but for more complex use cases, it might be helpful to have more communicators. An example might be if you wanted to perform calculations on a subset of the processes in a grid. For instance, all processes in each row might want to sum a value. This brings us to the first and most common function used to create new communicators:

```
MPI_Comm_split(
        MPI_Comm comm,
        int color,
        int key,
        MPI_Comm* newcomm)
```

EXTRA: Advantages of Shared Memory Architectures

❑ Compatibility with SMP hardware

❑ Ease of programming when communication patterns are complex or vary dynamically during execution

❑ Ability to develop applications using familiar SMP model, attention only on performance critical accesses

❑ Lower communication overhead, better use of BW for small items, due to implicit communication and

memory mapping to implement protection in hardware, rather than through I/O system

❑ HW-controlled caching to reduce remote communication by caching of all data, both shared and private

EXTRA: Advantages of Distributed Memory Architectures

❑ The hardware can be simpler (especially versus NUMA) and is more scalable

❑ Communication is explicit and simpler to understand

❑ Explicit communication focuses attention on costly aspect of parallel computation

❑ Synchronization is naturally associated with sending messages, reducing the possibility for errors introduced by incorrect synchronization

❑ Easier to use sender-initiated communication, which may have some advantages in performance

EXTRA: Metodologia Foster

1 - Particionando: Divida a Computação a ser realizada e os dados a serem operados em tarefas menores. O foco deve ser me executar tarefas que podem ser executadas em paralelo.

2 - Comunicação: Determinar que comunicação precisa ser executada entre as tarefas identificadas nos passos anteriores.

3 - Aglomeração ou Agregação: Combine as tarefas e comunicações identificadas nos passos anteriores em tarefas maiores. Por exemplo, se a tarefa A deve ser executada antes que a tarefa B possa ser executada, faz sentido agregá-las em uma única tarefa composta.

4 - Mapeamento: Atribua a tarefa composta identificada no passo anterior aos processos/threads. Isso deve ser feito de modo a minimizar a comunicação, e cada processo/thread recebe aproximadamente o mesmo volume de trabalho.

EXTRA: Ler Slides 04 e 05

EXTRA: Link da Michelle:
http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/