UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
Professor: Julio Melo

Trabalho Prático 1 – Processos, Memória compartilhada e Sincronização

Como vimos em aula, podemos realizar comunicação entre processos usando arquivos ou sinais, que, embora menos flexíveis podem ser usados para resolver problemas de comunicação mais simples.

No entanto, a utilização de sinais ou arquivos para comunicação dos processos possui diversas desvantagens que podem ocasionar perda de desempenho em um programa paralelo. Para resolver de forma mais genérica esse problema, podemos usar **memória compartilhada**, no entanto o uso desse recurso implica que algumas precauções devem ser tomadas.

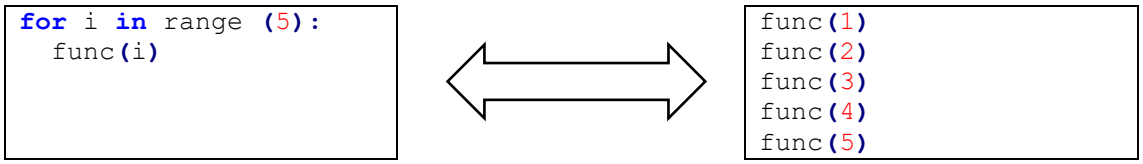
O principal problema com o uso de memória compartilhada está relacionado com a necessidade, em alguns casos, de **sincronização** do acesso à memória pelos processos que a compartilham. A sincronização entre processos é feita através de **semáforos ou *mutexes*** (que são um caso especial dos semáforos). Estes elementos são usados para **bloquear** o acesso à memória compartilhada, enquanto um processo a usa.

O trecho no código em que um processo usa a memória compartilhada é chamado de **região crítica**, trechos de código que escrevem em memória compartilhada sempre são considerados regiões críticas, no entanto trechos que fazem leitura da memória não são, obrigatoriamente, considerados críticos, a menos que o programa dependa de alguma ordem específica de execução entre os processos.

Exemplo prático de programação paralela

Loop *unrolling*[1] é uma técnica usada para acelerar automaticamente a execução de programas em determinados ambientes. Como o nome diz, esse processo funciona "desenrolando" um ou mais loops, de forma que eles possam ser executados em paralelo.

A ideia principal é bastante simples: Dado um loop com um número fixo e finito de instruções, é possível converter esse loop em várias instruções sequenciais.



Exemplo 1, exemplo de looo à esquerda, loop desenrolado à direira.

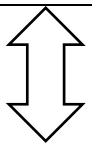
A ideia por trás desta técnica é que, se as várias instruções resultantes do desenrolar do loop puderem ser executadas em paralelo, o programa terá um ganho significativo em relação ao loop convencional.

Na prática, podemos usar essa técnica substituindo o loop clássico por uma função que realize o processamento em paralelo. Dessa forma implemente o loop unrolling de acordo com as especificações abaixo:

a) Defina uma função *unroll* que recebe uma matriz(LxN) de valores, **args**; uma função, **func**, que **deve** ser chamada em **paralelo;** uma lista de retornos, **results**, que deve ser usado para armazenar os retornos da função **func**, se existirem; um parâmetro, **method**, que é usado para indicar o tipo de implementação paralela que será usada. Cada linha, l, da matriz *args* representa uma chamada de *func*. Os parâmetros de cada chamada de *func* serão os elementos das linhas da matriz *args*, quando *func* retorna alguma coisa, esse retorno deve ser armazenado na lista *results*. O Parâmetro *method* indica se *unroll* irá implementar o paralelismo das chamadas de *func* usando **processos** com a função ***fork****(default)* ou usando ***threads***.

```
def func(var1, var2):
    print (str(var1) + ", " + str(var2))

#programa principal
#sem usar loop unrolling
for i in range(5):
    func(i, i+1)
```



```
def func(var1, var2):
    print (str(var1) + ", " + str(var2))

def unroll(args, func, method, results):
    ...

#programa principal
#usando loop unrolling
res = []
unroll([[0, 1],...,[4, 5]], func, 'proc',res)
unroll([[0, 1],...,[4, 5]], func, 'thre',res)
```

Exemplo 2, chamada da função unroll

b) Utilize a função *unroll* para implementar a soma e multiplicação de duas matrizes quaisquer. O programa principal deve imprimir as matrizes antes e o resultado da multiplicação/soma após o término do processo.

c) Apresente uma modelagem em redes de petri para a solução do item b.

Comparação entre implementação paralela e sequencial

Uma boa forma de verificar se a implementação paralela de um programa representa algum ganho é fazer uma comparação dos tempos que uma implementação sequencial demora para realizar a mesma operação. Assim, implemente os requisitos abaixo e plote os gráficos relativos às implementações sequencial, paralela com threads e paralela com processos.

a) Implemente um programa sequencial para calcular as mesmas operações em matrizes que foram realizadas pelo programa paralelo. Esse programa deve também calcular o tempo que leva para operar as matrizes.

b) Adicione, ao programa paralelo, uma lógica que permita calcular o tempo demorado para operar as matrizes. O tempo demorado entre o programa começar e chegar ao ponto de impressão do resultado.

c) Mostre, através de um gráfico, o ganho/perda entre as operações sequenciais e paralelas para matrizes com tamanhos crescentes: Ex: 1x1, 2x2, 3x3, 4x4, 5x5, 6x6, 8x8, 10x10, 20x20, 30x30, 40x40, 50x50, 75x75, 100x100, etc. (Dica: Utilize um gerador aleatório para os elementos das matrizes e enquanto incrementa linearmente o tamanho delas).

Sincronizando atividades paralelas

Um dos principais problemas da técnica de desenrolamento de loops é quando existe **dependência de dados** entre cada execução do loop. Nesse caso, a técnica pode convergir para uma execução igual à sequencial, pois os processos precisam esperar chamadas anteriores antes de executar.

```
aux = 0
for i in range(5):
  aux = aux + i;
```

Exemplo 3, exemplo de dependência de dados

A dependência de dados pode ser observada em problemas relativamente simples como o mostrado no Exemplo 3, nesse caso, para calcular o valor atual da variável aux, precisamos do valor anterior, o que implica que os 5 processos criados a partir do unrolling, teriam que esperar o cálculo dos processos anteriores antes de executar.

Uma solução ponto que corrigiria essa especificidade é a possibilidade de compartilhar a variável "aux" entre os processos, uma vez que o corpo do loop, não obrigatoriamente depende apenas o valor de aux. No entanto, a medida que mais variáveis são compartilhadas entre uma execução e outra do loop, mais sequencial o programa irá ser.

Mesmo existindo solução viável, identificar os pontos de dependência entre as execuções do loop não é uma tarefa trivial. Na maioria das vezes, é necessário executar o loop para que sejam identificados os pontos de dependência antes que alguma solução de execução em paralelo seja aplicada.

Outra solução viável é utilizar das características das operações realizadas, como associatividade, comultatividade, idempotência e outras para propor uma solução paralela mais fácil de ser automatizada. Um exemplo que mostra os usos dessas características é a multiplicação de N matrizes (supondo que possa ser feita).

a) Implemente um programa que gere e multiplique N matrizes utilizando a unroll. O programa deve ser o mais paralelo possível.
b) Modele, em redes de petri a multiplicação de 5 matrizes.

Entregáveis do Trabalho

O trabalho deve ser feito **em duplas**, e será equivalente a 50% da nota da unidade 1. A entrega será feita na data combinada com a turma contendo três itens principais: Um **relatório** contendo um resumo das soluções implementadas, os gráficos e as soluções relacionadas à modelagem; **um arquivo compactado** contendo **todos os programas** implementados; e **uma apresentação** (não precisa de slides, irei de mesa em mesa). A

linguagem usada na implementação é irrelevante, desde que seja acordado com o professor antes.