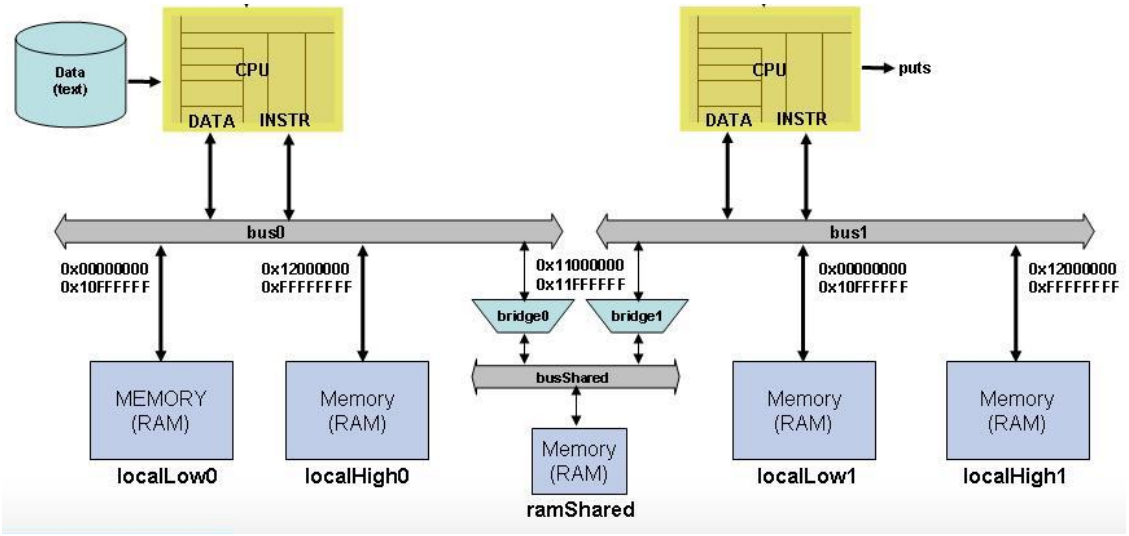
Trabalho OVP - Sort Shared Memory

Descrição geral:

Este trabalho tem por objetivo o desenvolvimento de aplicações para sistemas multiprocessados utilizando o paradigma de memória compartilhada. O aluno deve desenvolver uma aplicação escalável, ou seja, que possa ser executada em uma sistemas de diferentes tamanhos.

O aluno terá à disposição um script (*moduleGen.sh*) que gera sistemas de **N** processadores seguindo a arquitetura abaixo (que possui 2 processadores):



O objetivo é criar uma aplicação de ordenação (*sorting*) de vetores em um sistema multiprocessado com memória compartilhada. O sistema deve possuir **P** processadores, sendo que **P** deve ser uma potência de dois, como por exemplo: 2, 4, 8, 16, 32, etc. A aplicação deve ser baseada no paradigma de *divide and conquer,* ou seja, dividir o vetor para permitir que vários processadores contribuam com a ordenação. A fusão dos fragmentos ordenados deve ser efetuada seguindo a uma *Binary Tree Reduction.*

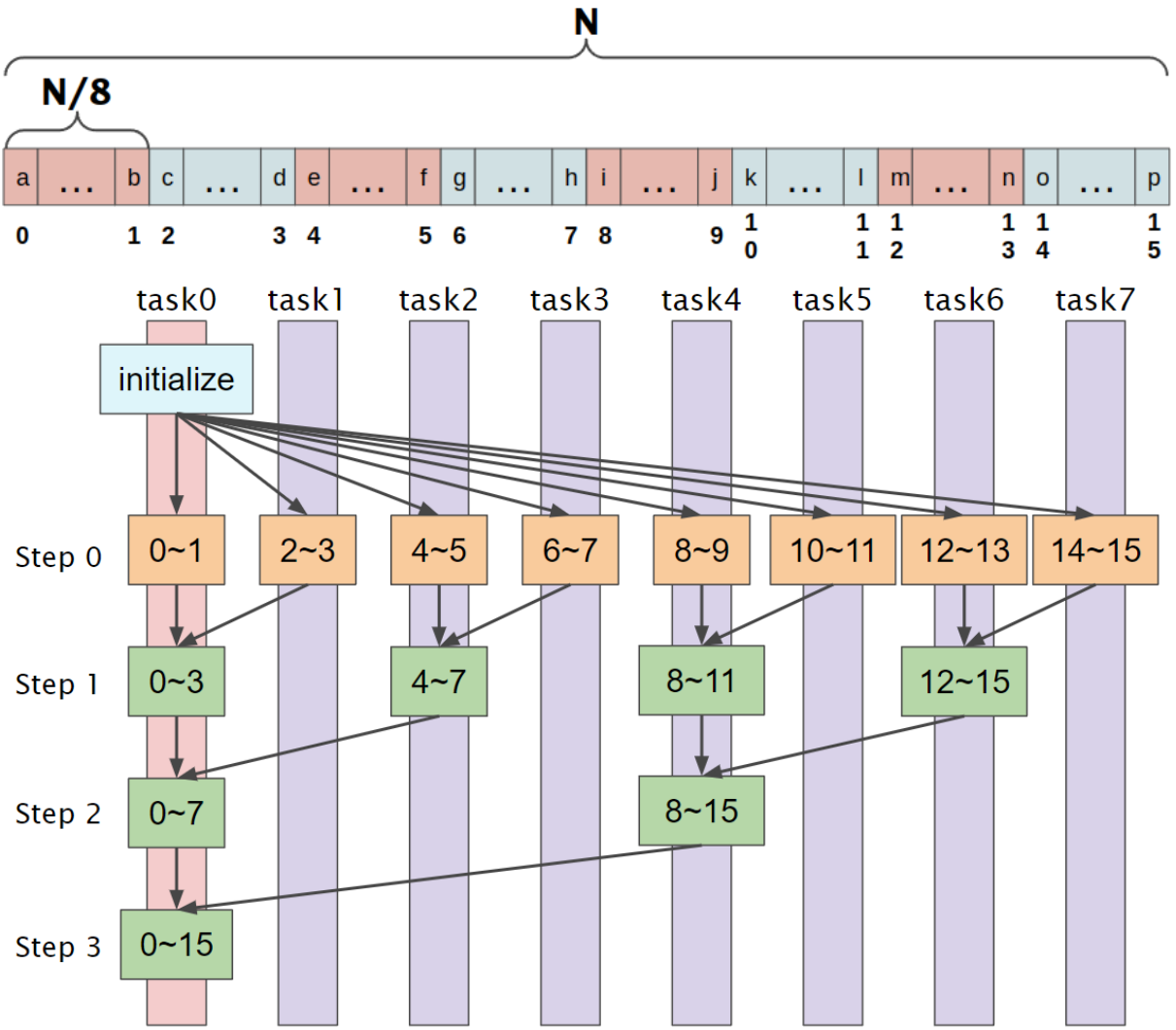
Dicas:

Vocês devem obter o .zip do trabalho que contém um esqueleto para a execução do trabalho.

- A aplicação é dividida em duas tarefas (**master.c** e **slave.c**). O master é responsável pela inicialização do vetor e da distribuição dos sub-vetores aos outros processadores. A tarefa slave é replicada em todos os outros processadores e aguarda os comandos provenientes do master para começar a execução.
- O *sorting* pode ser efetuado por qualquer algoritmo, entretanto, para facilitar, serão disponibilizados os algoritmos **QuickSort()**, para realizar o *sort* do fragmento do vetor atribuído ao processador, e o **MergeSort()** para fazer a fusão dos fragmentos.
- O trabalho já vem configurado com um sistema de 8 processadores. Para ampliar o sistema, basta entrar na pasta **module** e utilizar o script **moduleGen.sh** passando como referência o valor da quantidade de processadores para que um *.tcl* apropriado seja gerado automaticamente.
- Para executar o sistema (se ocorrer modificações) deve-se editar o script na pasta raiz (**example.sh**), adicionando uma tarefa para cada processador criado, por exemplo, um sistema onde N = 4 temos a seguinte situação:

```
19  # run the module using the harness
20  harness.exe --modulefile module/model.${IMPERAS_SHRSUF} \
21              --program twoProcessorShared/P0=application/master.${CROSS}.elf \
22              --program twoProcessorShared/P1=application/slave.${CROSS}.elf \
23              --program twoProcessorShared/P2=application/slave.${CROSS}.elf \
24              --program twoProcessorShared/P3=application/slave.${CROSS}.elf \
25              $* --imperasintercepts --parallelmax
```

As figuras abaixo exemplificam o funcionamento esperado da aplicação:



Requisitos:

1. Aplicação de *sorting* em sistema multiprocessado com memória compartilhada utilizando o paradigma de mestre-escravo e *Binary Tree Reduction*.
2. **A aplicação deve aceitar qualquer tamanho de vetor**. A aplicação **DEVE** ser capaz de tratar vetores de tamanho que **não** seja divisível pelo número de tarefas. *Por exemplo*, a aplicação deve ser capaz de ordenar um vetor de 103 posições em um sistema de 8 processadores.
3. Fazer um pequeno relatório comentando o código e providenciar um .zip contendo um cenário pronto para execução.