ENGENHARIA AEROESPACIAL E MECÂNICA
MECATRÔNICA
CE-265 PROCESSAMENTO PARALELO

Docente: Jairo Panetta
Discente: Lucas Kriesel Sperotto

Exercício 2 – RELATÓRIO

Paralelização por OpenMP do "Jogo da Vida" no Sistema "CROW"

Introdução:

Este relatório esta dividido em cinco partes distintas: procedimento, resultados e anexos de uma a três.

No tópico procedimento demostrarei o "passo a passo" para a realização do trabalho, bem como o raciocínio que me levou a deduzir a maneira com o qual paralelizei o código, por fim, serão apresentados os resultados de tempo e desempenho com sua interpretação.

No anexo um está exposto o arquivo de saída gerado para comprovar a correta instalação, No anexo dois está o trecho do código de ModVida.f90 paralelizado, e por ultimo, no anexo três está a saída gerada para comprovar a correta instalação, gerado pela execução com três threads.

Procedimento:

Utilizando-se do Unbuntu Linux 10.10 virtualizado pelo software VMware Workstation, efetuado o log-in no sistema "crow" tanto por "ssh" como por "sftp" (utilizando dois terminais).

Transferido dados já descompactados e compilado código, executado "XmitFunciona.sh" para garantir a correta instalação, o resultado se encontra no Anexo 1.

Para paralelizar o laço principal de "ModVida.f90", verificado e imaginado como se comportariam as variáveis que interagiam dentro do laço. Notado que as variáveis auxiliares dos laços, especialmente a do laço interno deveria ser privada para se evitar conflito com valores gerados pelos threads, obviamente a variável usada para guardar o numero de vizinhos vivos também teria que ser privada pelo mesmo motivo citado atrás.

Após a análise do código, paralelizado o laço principal do "ModVida.f90", criando três variáveis do tipo "FirtPrivate", sendo elas a "**i**" e "**j**" usadas nos laços e a "**vizviv**", utilizada para

guardar a contagem de vizinhos vivos a célula “alvo” necessária para impor a condição de “viva” ou “morta”. O “FirstPrivate” foi escolhido pelo motivo que ela permite iniciar a variável com o valor inicial. Mas percebo agora que não havia essa necessidade.

Ao todo efetuei três “baterias” de testes em tempos distintos e verifiquei uma diferença sutil no tempo de execução para números iguais de threads, creio que isso se deve as condições do sistema no ato de execução (memória, núcleos disponíveis, etc).

Tabelas de Resultados

Na tabela 1 está descrito os tempos de execução para diferentes tamanhos de tabuleiro com de um a oito threads.

Tabela 1 - Tempo para Diferentes Tamanhos de Tabuleiro e Números de Threads

Tamanho Tabuleiro	Tempo 1 Thread	Tempo 2 Thread	Tempo 3 Thread	Tempo 4 Thread	Tempo 5 Thread	Tempo 6 Thread	Tempo 7 Thread	Tempo 8 Thread
32	**0.00746**	**0.00382**	**0.00271**	**0.00205**	**0.00183**	**0.00158**	**0.00140**	**0.00140**
64	**0.06242**	**0.03141**	**0.02162**	**0.01598**	**0.01318**	**0.01122**	**0.00996**	**0.00857**
128	**0.51075**	**0.25610**	**0.17262**	**0.12883**	**0.10492**	**0.08887**	**0.07679**	**0.06557**
256	**4.13413**	**2.06827**	**1.38966**	**1.03632**	**0.84109**	**0.69774**	**0.60111**	**0.52139**
512	**33.30424**	**16.65677**	**11.12755**	**8.32923**	**6.70102**	**5.59869**	**4.81348**	**4.17308**
1024	**268.25574**	**134.10254**	**89.45499**	**66.88194**	**53.5069**	**44.63345**	**38.36774**	**33.41928**

O interessante em se executar até oito threads é que pude perceber o efeito da lei de Amdahl para o tabuleiro de tamanho “32” com mais de sete threads, onde o tempo de processamento se limita a parte sequencial do software. Na tabela 2 vemos os Speed-Up calculados com base nos tempos da tabela 1, agora vemos que mesmo aumentando o numero de threads não temos nenhum ganho para topologias de tabuleiro pequenas, temos um pequeno ganho para de 6 a 7 threads e nenhum ganho para de 7 a 8.

Tabela 2 - Speed-Up para Diferentes Tamanhos de Tabuleiro e Números de Threads

Tamanho Tabuleiro	Speed-Up 2 Thread	Speed-Up 3 Thread	Speed-Up 4 Thread	Speed-Up 5 Thread	Speed-Up 6 Thread	Speed-Up 7 Thread	Speed-Up 8 Thread
32	**1.952**	**2.752**	**3.628**	**4.076**	**4.721**	**5.328**	**5.328**
64	**1.987**	**2.887**	**3.906**	**4.735**	**5.563**	**6.267**	**7.283**
128	**1.994**	**2.958**	**3.964**	**4.867**	**5.747**	**6.659**	**7.789**
256	**1.998**	**2.974**	**3.989**	**4.915**	**5.925**	**6.877**	**7.929**
512	**1.999**	**2.992**	**3.998**	**4.970**	**5.948**	**6.918**	**7.980**
1024	**2.000**	**2.998**	**4.010**	**5.013**	**6.010**	**6.991**	**8.026**

Temos no gráfico 1, uma melhor visualização do comportamento do speed-up para os respectivos tabuleiros e números de threads, nele reconhecemos a baixa eficiência para os casos de tabuleiro menor com sete e oito threads.

Conforme aumentamos o problema o speed-up sempre aumenta, isso se deve ao fato da parcela paralelizada no software se torna maior (mais significativa em relação a parte sequencial). E obviamente se aumentando o numero de threads aumentamos o speed-up, mas para se ter certeza do quanto isso é vantajoso devemos calcular a eficiência para cada speed-up.

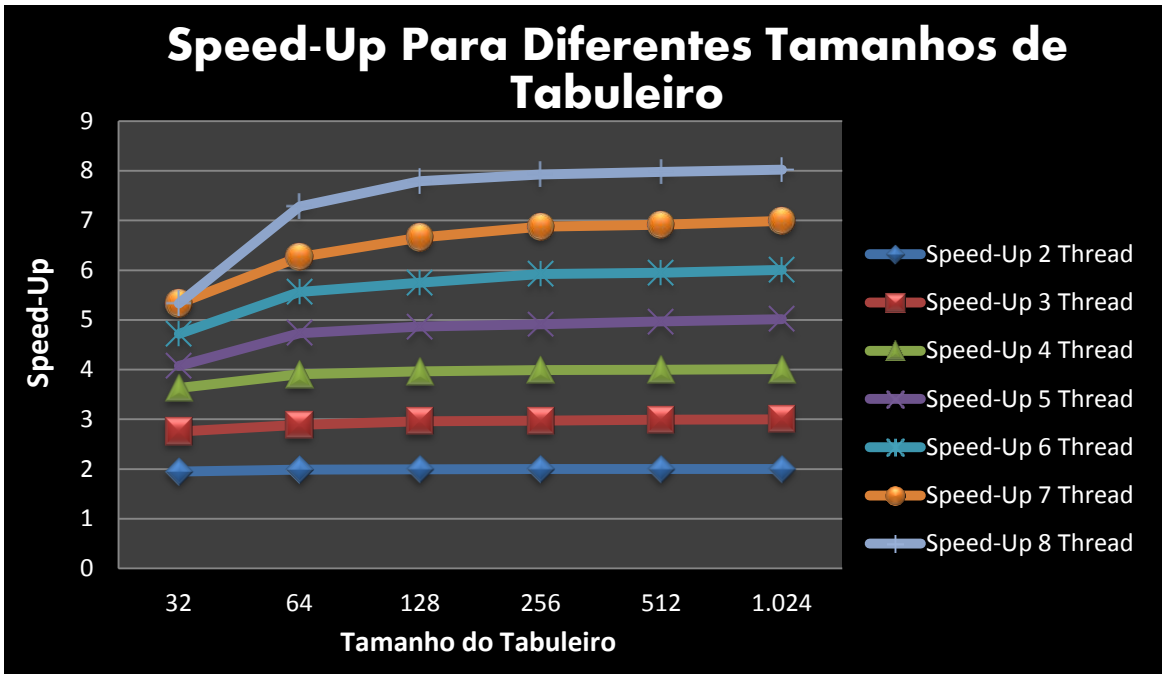

Gráfico 1 – Speed-Up Para Diferentes Tamanhos de Tabuleiro e Numero de Threads

Na Tabela 3 temos as eficiências calculadas. Essas eficiências foram calculadas dividindo o speed-up pelo numero de threads.

Tabela 3 – Eficiência Para Diferentes Tamanhos de Tabuleiro e Números de Threads

Tamanho Tabuleiro	Eficiência 2 Thread	Eficiência 3 Thread	Eficiência 4 Thread	Eficiência 5 Thread	Eficiência 6 Thread	Eficiência 7 Thread	Eficiência 8 Thread
32	**97,6%**	**91,7%**	**90,7%**	**81,5%**	**78,6%**	**76,1%**	**66,6%**
64	**99,3%**	**96,2%**	**97,6%**	**94,7%**	**92,7%**	**89,5%**	**91,0%**
128	**99,7%**	**98,6%**	**99,1%**	**97,3%**	**95,7%**	**95,1%**	**97,3%**
256	**99,9%**	**99,1%**	**99,7%**	**98,3%**	**98,7%**	**98,2%**	**99,1%**
512	**99,9%**	**99,7%**	**99,9%**	**99,4%**	**99,1%**	**98,8%**	**99,7%**
1024	**100%**	**99,9%**	**102%**	**102%**	**101%**	**99,8%**	**103%**

Vemos claramente pela tabela 3 quais as situações que propuseram maior desempenho na paralelização. Temos valores de eficiência baixos para os casos pequenos, com mais de cinco threads, isso confirma a lei de Amdahl e a impossibilidade de termos melhoras para certos casos. Para termos uma boa paralelização, temos que ter em vista o tamanho do problema para saber até onde podemos chegar sem desperdiçar poder de processamento.

Observamos também tanto eficiências maiores de 100% e speed-up acima do nominal, isso ou mostra que o software paralelizado teve um desempenho super-linear ou então é devido a erros nas medições dos tempos de execução. Devemos ter sempre uma margem de erro para fatores externos ao procedimento executado.

Por fim concluo que achei muito proveitoso essa atividade, principalmente pela possibilidade de aprender os comandos utilizados pelo sistema da “cray”, e por poder comparar e testar várias possibilidades (tentativas) de paralelização do mesmo código após a realização da tarefa.

Anexo 1 – Correta Instalação

```
***execucao com 1 threads***

  estado inicial dump das posicoes  1:  6
  ============================
  .X....
  ..X...
  XXX...
  ......
  ......
  ......
  ============================
  geracao 01 dump das posicoes  1:  6
  ============================
  ......
  X.X...
  .XX...
  .X....
  ......
  ......
  ============================
  geracao 02 dump das posicoes  1:  6
  ============================
  ......
  ..X...
  X.X...
  .XX...
  ......
  ......
  ============================
  geracao 03 dump das posicoes  1:  6
  ============================
  ......
  .X....
  ..XX..
  .XX...
  ......
  ......
  ============================
  geracao 04 dump das posicoes  1:  6
  ============================
  ......
  ..X...
  ...X..
  .XXX..
  ......
  ......
  ============================
  geracao 05 dump das posicoes  1:  6
  ============================
  ......
  ......
  .X.X..
  ..XX..
  ..X...
  ......
  ============================
  geracao 06 dump das posicoes  1:  6
  ============================
  ......
  ......
  ...X..
  .X.X..
  ..XX..
  ......
  ============================
  geracao 07 dump das posicoes  1:  6
  ============================
  ......
  ......
  ..X...
  ...XX.
  ..XX..
  ......
  ============================
  geracao 08 dump das posicoes  1:  6
  ============================
  ......
  ......
  ...X..
  ....X.
  ..XXX.
  ......
  ============================
  geracao 09 dump das posicoes  1:  6
  ============================
  ......
  ......
  ......
  ..X.X.
  ...XX.
  ...X..
  ============================
  geracao 10 dump das posicoes  1:  6
  ============================
  ......
  ......
  ......
  ....X.
  ..X.X.
  ...XX.
  ============================
  geracao 11 dump das posicoes  1:  6
  ============================
  ......
  ......
  ......
  ...X..
  ....XX
  ...XX.
  ============================
  geracao 12 dump das posicoes  1:  6
  ============================
  ......
  ......
  ......
  ....X.
  .....X
  ...XXX
  ============================
 Resultado correto
Application 106071 resources: utime ~0s,
stime ~0s

real    0m0.068s
user    0m0.008s
sys     0m0.004s
```

Anexo 2 – Código Paralelizado

```
module JogoDaVida
  implicit none
contains
  ! UmaVida: Executa uma iteracao do Jogo da Vida
  !          em tabuleiros de tamanho tam. Produz o tabuleiro
  !          de saida tabulOut a partir do tabuleiro de entrada
  !          tabulIn. Os tabuleiros tem tam-1 x tam-1 celulas
  !          internas vivas ou mortas. O tabuleiro eh orlado
  !          por celulas eternamente mortas.
  subroutine UmaVida (tam, tabulIn, tabulOut)
    integer, intent(in ) :: tam
    logical, intent(in ) :: tabulIn(0:tam+1,0:tam+1)
    logical, intent(out) :: tabulOut(0:tam+1,0:tam+1)

    integer :: i
    integer :: j
    integer :: vizviv

    ! percorre o tabuleiro determinando o proximo
    ! estado de cada célula

!$OMP PARALLEL DO FIRSTPRIVATE (i,j,vizviv)
    do j = 1, tam

       do i = 1, tam

          ! quantos vizinhos vivos

          vizviv = count( (/&
               tabulIn(i-1,j-1), tabulIn(i-1,j), tabulIn(i-1,j+1), &
               tabulIn(i  ,j-1),                 tabulIn(i  ,j+1), &
               tabulIn(i+1,j-1), tabulIn(i+1,j), tabulIn(i+1,j+1) /) )

          ! impoe regra do proximo estado

          if (tabulIn(i,j) .and. vizviv < 2) then
             tabulOut(i,j) = .false.
          else if (tabulIn(i,j) .and. vizviv > 3) then
             tabulOut(i,j) = .false.
          else if (.not. tabulIn(i,j) .and. vizviv == 3) then
             tabulOut(i,j) = .true.
          else
             tabulOut(i,j) = tabulIn(i,j)
          end if
       end do
    end do
!$OMP END PARALLEL DO

  end subroutine UmaVida
 ...
```

Anexo 3 – Correta Paralelização

```
***execucao com 3 threads***

 estado inicial dump das posicoes  1:  6
============================
.X....
..X...
XXX...
......
......
......
============================
geracao 01 dump das posicoes  1:  6
============================
......
X.X...
.XX...
.X....
......
......
============================
geracao 02 dump das posicoes  1:  6
============================
......
..X...
X.X...
.XX...
......
......
============================
geracao 03 dump das posicoes  1:  6
============================
......
.X....
..XX..
.XX...
......
......
============================
geracao 04 dump das posicoes  1:  6
============================
......
..X...
...X..
.XXX..
......
......
============================
geracao 05 dump das posicoes  1:  6
============================
......
......
.X.X..
..XX..
..X...
......
============================
geracao 06 dump das posicoes  1:  6
============================
......
......
...X..
.X.X..
..XX..
......
============================
geracao 07 dump das posicoes  1:  6
============================
......
......
..X...
...XX.
..XX..
......
============================
geracao 08 dump das posicoes  1:  6
============================
......
......
...X..
....X.
..XXX.
......
============================
geracao 09 dump das posicoes  1:  6
============================
......
......
......
..X.X.
...XX.
...X..
============================
geracao 10 dump das posicoes  1:  6
============================
......
......
......
....X.
..X.X.
...XX.
============================
geracao 11 dump das posicoes  1:  6
============================
......
......
......
...X..
....XX
...XX.
============================
geracao 12 dump das posicoes  1:  6
============================
......
......
......
....X.
.....X
...XXX
============================
 Resultado correto
Application 106358 resources: utime ~0s,
stime ~0s

real    0m0.072s
user    0m0.008s
sys     0m0.032s
```