Instituto de Informática

OpenMP:
Uma Introdução

Cláudio Geyer

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Informática
UFRGS

Fonte

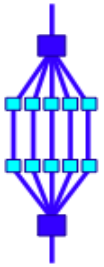
- Fontes
  - Slides inicialmente baseados em palestra de Ruud van der Pas
    - Janeiro de 2009
    - Na Nanyang Technological University, Singapore
  - Afiliação autor
    - Sun Microsystems

Informática UFRGS

Fonte

NANYANG TECHNOLOGICAL UNIVERSITY

Sun microsystems

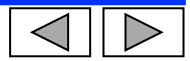


An Overview of OpenMP

Ruud van der Pas

Senior Staff Engineer
Technical Developer Tools
Sun Microsystems, Menlo Park, CA, USA

Nanyang Technological University
Singapore
Wednesday January 14, 2009



UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Informática UFRGS

Sumário

- OpenMP:
  - o que é? Para que serve? ...
  - Quando usar
  - Vantagens
  - Visão geral das principais funcionalidades
    - Paralelização de loops
  - Detalhamento dos principais recursos

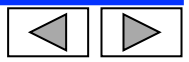
UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Referências

- Referências:
  - Curso introdutório a OpenMP (slides)
    - http://openmp.org/wp/2010/07/iwomp-2010-material-available/
    - Em particular:
      - a 4ª (última) parte é sobre escalabilidade e problemas de desempenho em certas aplicações OpenMP
      - http://www.compunity.org/training/tutorials/4%20OpenMP_and_Performance.pdf

Referências

- Referências:
  - Parallel Programming with GCC
    - Diego Novillo, Red Hat
    - Red Hat Summit, Nashville, May 2006
    - http://www.airs.com/dnovillo/Papers/rhs2006.pdf
  - Introduction to Parallel Programming in OpenMP
    - David Colignon
    - CÉCI - Consortium des Équipements de Calcul Intensif
      - http://www.ceci-hpc.be
    - http://www.cism.ucl.ac.be/Services/Formations/OpenMP_Intro.pdf



Informática UFRGS

Referências

- Referências:
  - Parallel Programming with MPI and OpenMP
    - Marc-André Hermanns, Jülich
    - Boas seções introdutórias
      - Sobre arquiteturas paralelas
      - Sobre programação paralela
    - http://www2.fz-juelich.de/jsc/files/docs/vortraege/MPI-OpenMP.pdf

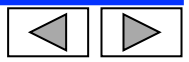
UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Referências

- Referências:
  - Especificação
    - OpenMP, The OpenMP API specification for parallel programming
    - http://openmp.org/
  - Artigos
    - Wikipedia (good summary)
      - http://en.wikipedia.org/wiki/Openmp
    - 32 OpenMP traps for C++ developers
      - http://software.intel.com/en-us/articles/32-openmp-traps-for-c-developers/

Referências

- Referências:
  - Artigos
    - Common Mistakes in OpenMP and How To Avoid Them
      - http://www.michaelsuess.net/.../suess_leopold_common_mistakes_06.pdf
    - IWOMP 2009, The 2009 International Workshop on OpenMP (Slides)
      - http://openmp.org/wp/2009/06/iwomp2009/
    - IWOMP 2010, The 2010 International Workshop on OpenMP (Slides)
      - http://openmp.org/wp/2010/07/iwomp-2010-material-available/



Referências

- Referências:
  - Artigos
    - Avoiding and Identifying False Sharing Among Threads
      - http://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing

Referências

- Referências:
  - Tutoriais
    - Parallel Programming with OpenMP: an Introduction,
      - A. Duran, BSC
      - http://www.prace-project.eu/hpc-training/training_pdfs/2641.pdf
    - A "Hands-on" Introduction to OpenMP,
      - SC08, Mattsonand Meadows, Intel
      - http://www.openmp.org/mp-documents/omp-hands-on-SC08.pdf
    - Cours OpenMP (en français !) de l'IDRIS
      - http://www.idris.fr/data/cours/parallel/openmp/

Referências

- Referências:
  - Tutoriais
    - Using OpenMP,
      - SC09, Hartman-Baker R., ORNL, NCCS
      - http://www.greatlakesconsortium.org/events/scaling/files/openmp09.pdf
    - OpenMP Tutorial,
      - Barney B., LLNL
      - https://computing.llnl.gov/tutorials/openMP/

Informática UFRGS

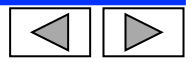
Organizações OpenMP

OpenMP™

http://www.openmp.org

cOMPunity

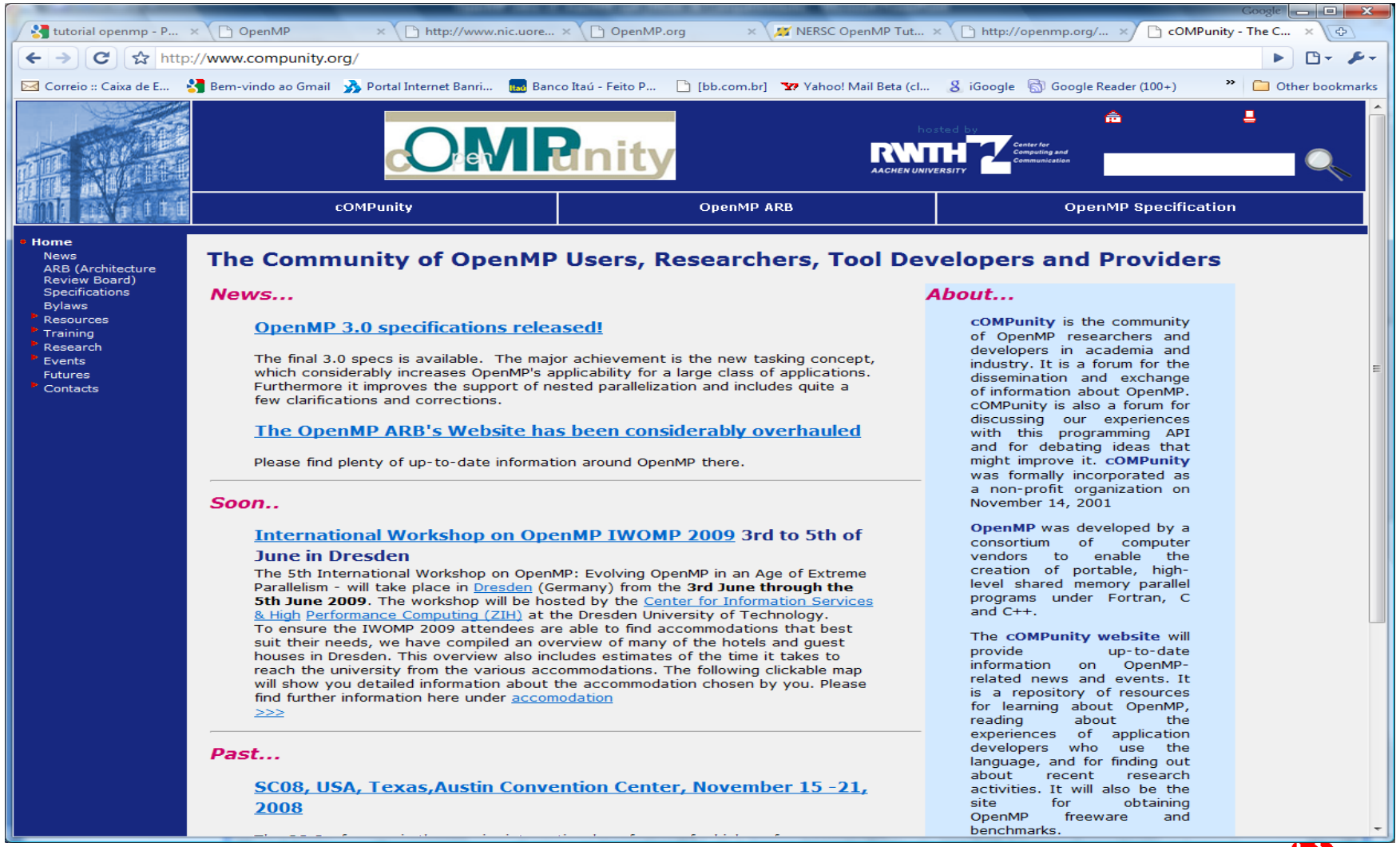
http://www.compunity.org



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

OpenMP: 1ª página

OpenMP Community: 1ª página



O que é OpenMP?

- Especificação para programação paralela em memória compartilhada
- Padrão “de fato”
- Mantida por ***OpenMP Architecture Review Board***
  - http://www.openmp.org
- Versão 3.0 produzida em maio de 2008
- Consiste de
  - Diretivas de compilação
  - Rotinas de execução
  - Variáveis de ambiente

Informática UFRGS

Quando usar OpenMP

- Comparando com compiladores paralelizantes sem diretivas
- O compilador não consegue obter o paralelismo desejado pelo programador
  - Porque não consegue "ver" o paralelismo
    - A análise de dependências não tem certeza se pode paralelizar
  - A granularidade das tarefas não é suficiente
    - O compilador não tem informações para paralelizar no mais alto nível
- A paralelização explícita via OpenMP pode resolver esses problemas

Vantagens de OpenMP

- Bom desempenho e escalabilidade
  - Se o programador fizer "a coisa certa"
- Padrão de fato e maduro
- Programa OpenMP é portável
  - Suportado por vários compiladores
  - IBM, Intel, ...
- Requer pouco esforço do programador
  - Comparando com Posix ou Java threads
- Permite paralelização incremental
  - Por partes (loops) do programa
- Programa sequencial (quase) = paralelo

Vantagens de OpenMP

- Programa sequencial (quase) = paralelo
  - Facilita manutenção
  - Facilita depuração
    - Por exemplo, em caso de erro (?) na versão paralela: executar a versão sequencial com mesma entrada

Informática UFRGS

OpenMP e Multicore

- OpenMP é especialmente indicado para processadores multicore
  - Modelos de memória e de threads podem ser mapeados de forma natural
  - Leve
  - Maduro
  - Muito usado e disponível

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Modelo de Memória de OpenMP

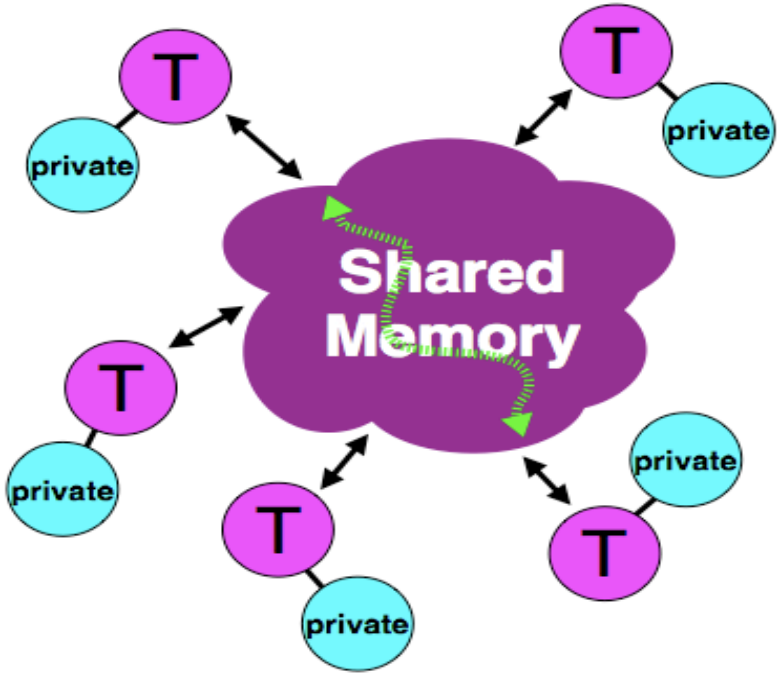
- Memória global compartilhada
  - Todas as tarefas (threads) têm acesso a essa memória
- Dados podem ser compartilhados ou privados
- Dados compartilhados podem ser acessados por todas as tarefas
- Dados privados somente podem ser acessados pela tarefa proprietária dos dados
- Transferência de dados é transparente ao programador
- Há mecanismos de sincronização implícitos

Modelo de Memória de OpenMP

NTU Talk
January 14
2009

10

The OpenMP Memory Model



- ✔ *All threads have access to the same, globally shared, memory*
- ✔ *Data can be shared or private*
- ✔ *Shared data is accessible by all threads*
- ✔ *Private data can only be accessed by the thread that owns it*
- ✔ *Data transfer is transparent to the programmer*
- ✔ *Synchronization takes place, but it is mostly implicit*

Atributos de Dados Compartilhados

- Em um programa OpenMP, os dados precisam receber um atributo ("labelled")
- Há dois tipos básicos
  - Shared
  - Private
- Shared
  - Só há uma instância do dado
  - Todas as tarefas podem ler e escrever nesses dados concorrentemente
    - Exceções se construtores são usados
  - Todas as alterações são visíveis a todas as tarefas
    - Mas não imediatamente a não ser se forçadas

Atributos de Dados Compartilhados

- Private
  - Cada tarefa tem uma cópia do dado
  - Nenhuma outra tarefa pode acessar o dado
  - Alterações são visíveis somente à tarefa proprietária do dado

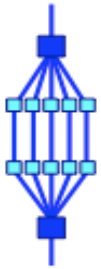
- Mais informações sobre a semântica de variáveis
  - Curso sobre OpenMP na ERAD 2010

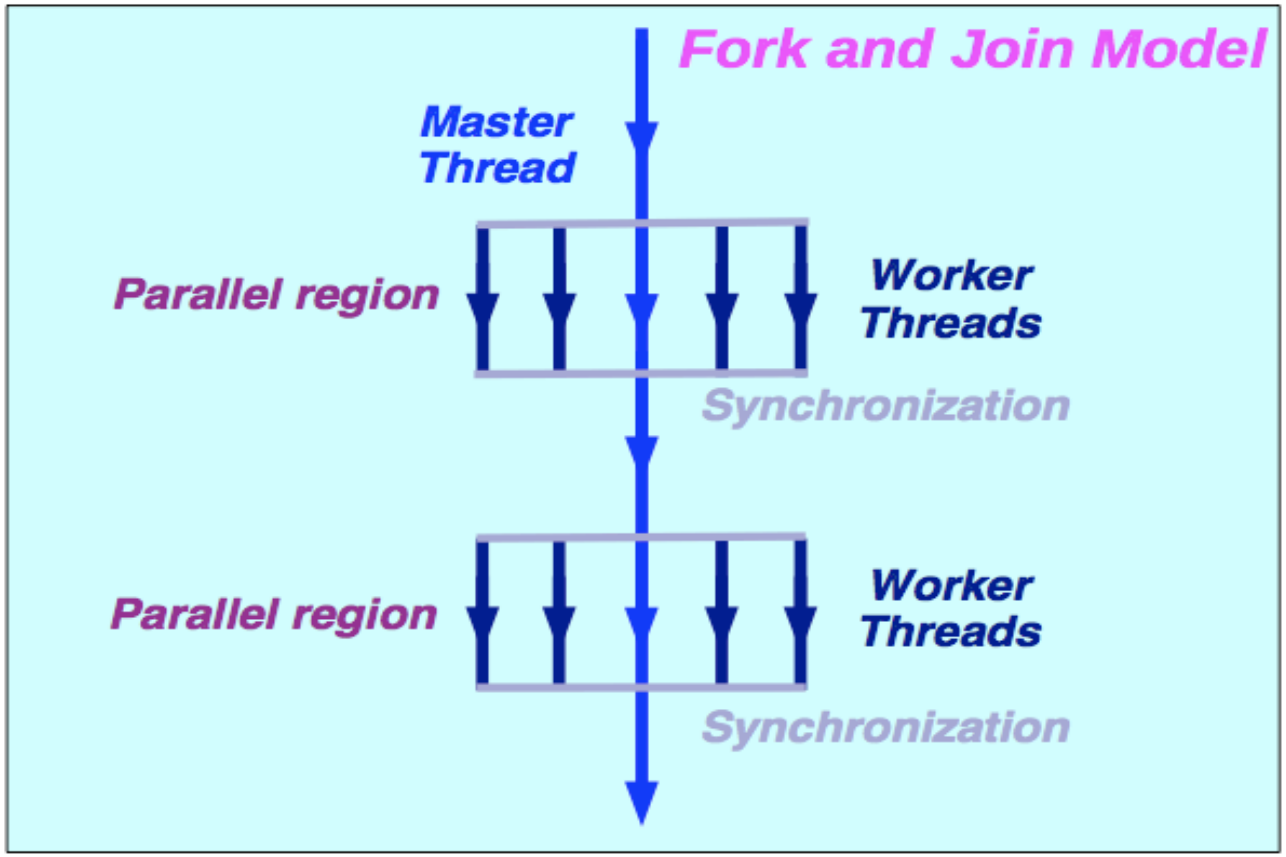
Modelo de Fluxos de Execução

- Modelo de fluxos de execução
  - Segue um modelo parbegin/parend repetido
  - Tipos de threads
    - Master: nas partes sequenciais
    - Workers: nas partes concorrentes
  - Região paralela
    - Parte concorrente
    - Com threads workers

Modelo de Execução OpenMP

The OpenMP Execution Model





Primeiro Exemplo OpenMP

- Loop sequencial com iterações independentes

```
for (int i=0; i<n; i++)
   c[i] = a[i] + b[i];
```

- Loop paralelizado usando diretiva OpenMP

```
#pragma omp parallel for
for (int i=0; i<n; i++)
   c[i] = a[i] + b[i];
```

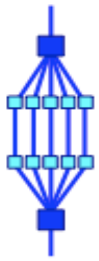
```
% cc –xopenmp source.c
% setenv OMP_NUM_THREADS 5
% a.out
```

Exemplo de execução de loop

- Exemplo de execução de loop
  - Próximo slide com figura
  - 5 threads (fluxos de execução)
    - Numeradas de 0 a 4
  - Código único nas threads
  - Vetor de 1000 elementos
  - Cada uma operando sobre um subconjunto distinto de partes do vetor
    - Vetor parcial com índices consecutivos
  - Por exemplo, thread 1 sobre índices 200 a 399

Execução Paralela do Exemplo

Example parallel execution



Thread 0	Thread 1	Thread 2	Thread 3	Thread 4
i=0-199	i=200-399	i=400-599	i=600-799	i=800-999
a[i]	a[i]	a[i]	a[i]	a[i]
+	+	+	+	+
b[i]	b[i]	b[i]	b[i]	b[i]
=	=	=	=	=
c[i]	c[i]	c[i]	c[i]	c[i]

Componentes da versão 2.5

- Diretivas
  - Região paralela
  - Worksharing
  - Synchronization
  - Atributos de dados compartilhados
    - private
    - firstprivate
    - lastprivate
    - shared
    - Reduction
  - Orphaning

Componentes da versão 2.5

- Ambiente de execução
  - Quantidade de tarefas
  - Id da tarefa
  - Ajuste dinâmico de tarefas
  - Paralelismo aninhado
  - Tempo de parede
  - Bloqueios
- Variáveis de ambiente
  - Quantidade de tarefas
  - Tipo de escalonamento
  - Ajuste dinâmico de tarefas
  - Paralelismo aninhado

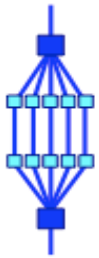
Exemplo Produto Vetorial (M*V)

- Exemplo mais elaborado
  - Produto vetorial: M*V
  - Visto como exemplo em algoritmos PRAM
  - Conjuntos de variáveis shared e private
  - Cada thread calcula um subconjunto de elementos do vetor resultado

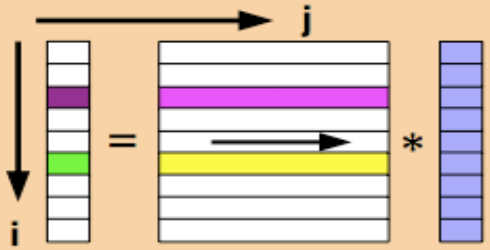
2º Exemplo: M*V

Example - Matrix times vector

Sun microsystems



```
#pragma omp parallel for default(none) \
            private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
   sum = 0.0;
   for (j=0; j<n; j++)
     sum += b[i][j]*c[j];
   a[i] = sum;
 }
```



TID = 0

```
for (i=0,1,2,3,4)
 i = 0
sum = Σ  b[i=0][j]*c[j]
   a[0] = sum
 i = 1
sum = Σ  b[i=1][j]*c[j]
   a[1] = sum
```

TID = 1

```
for (i=5,6,7,8,9)
 i = 5
sum = Σ  b[i=5][j]*c[j]
   a[5] = sum
 i = 6
sum = Σ  b[i=6][j]*c[j]
   a[6] = sum
```

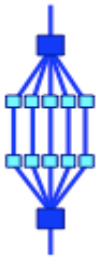
... etc ...

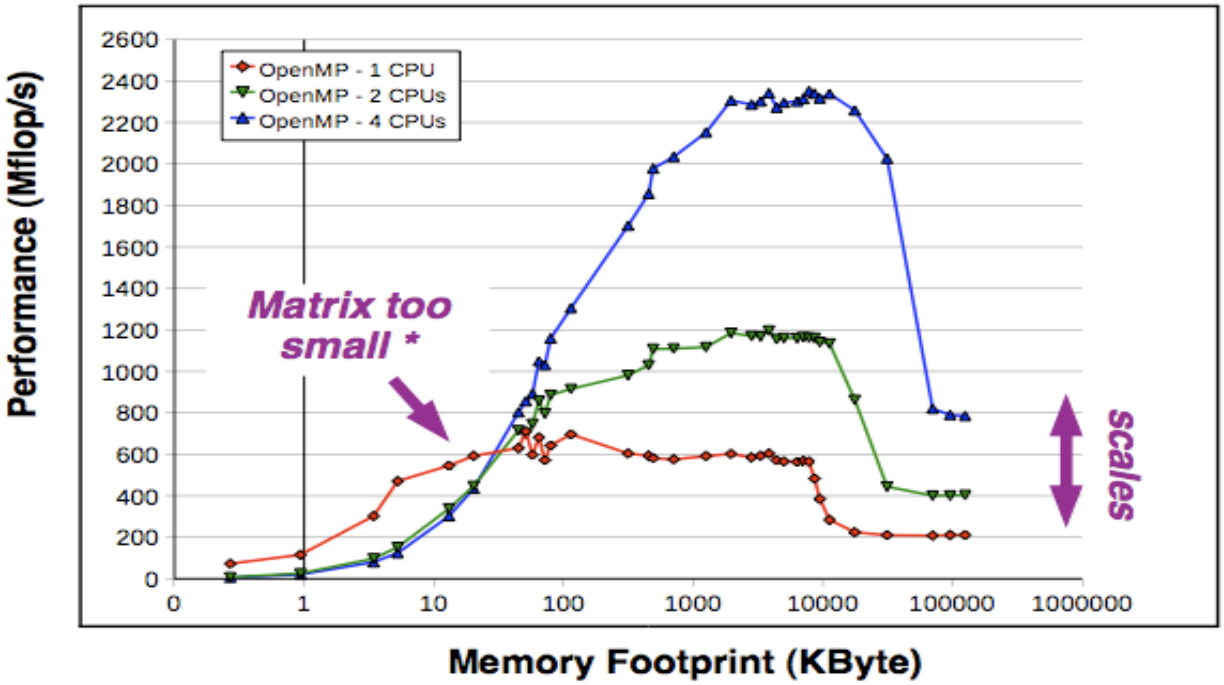
Exemplo de Avaliação de Desempenho

- Exemplo de avaliação de desempenho
  - Execução do mesmo problema com variação do número de cpus
  - Uma curva para cada quantidade de cpus
  - Eixo X:
    - Tamanho do problema
  - Eixo Y:
    - Tempo de execução (tempo paralelo)
  - Notar a perda de desempenho para entradas pequenas

Desempenho de OpenMP

OpenMP performance





*) With the IF-clause in OpenMP this performance degradation can be avoided

Outro Exemplo Sintético

- Exemplo de programa sintético
  - Slide com código adiante
  - Exemplo abstrato
  - Com várias blocos paralelizados
  - Uso de barreira

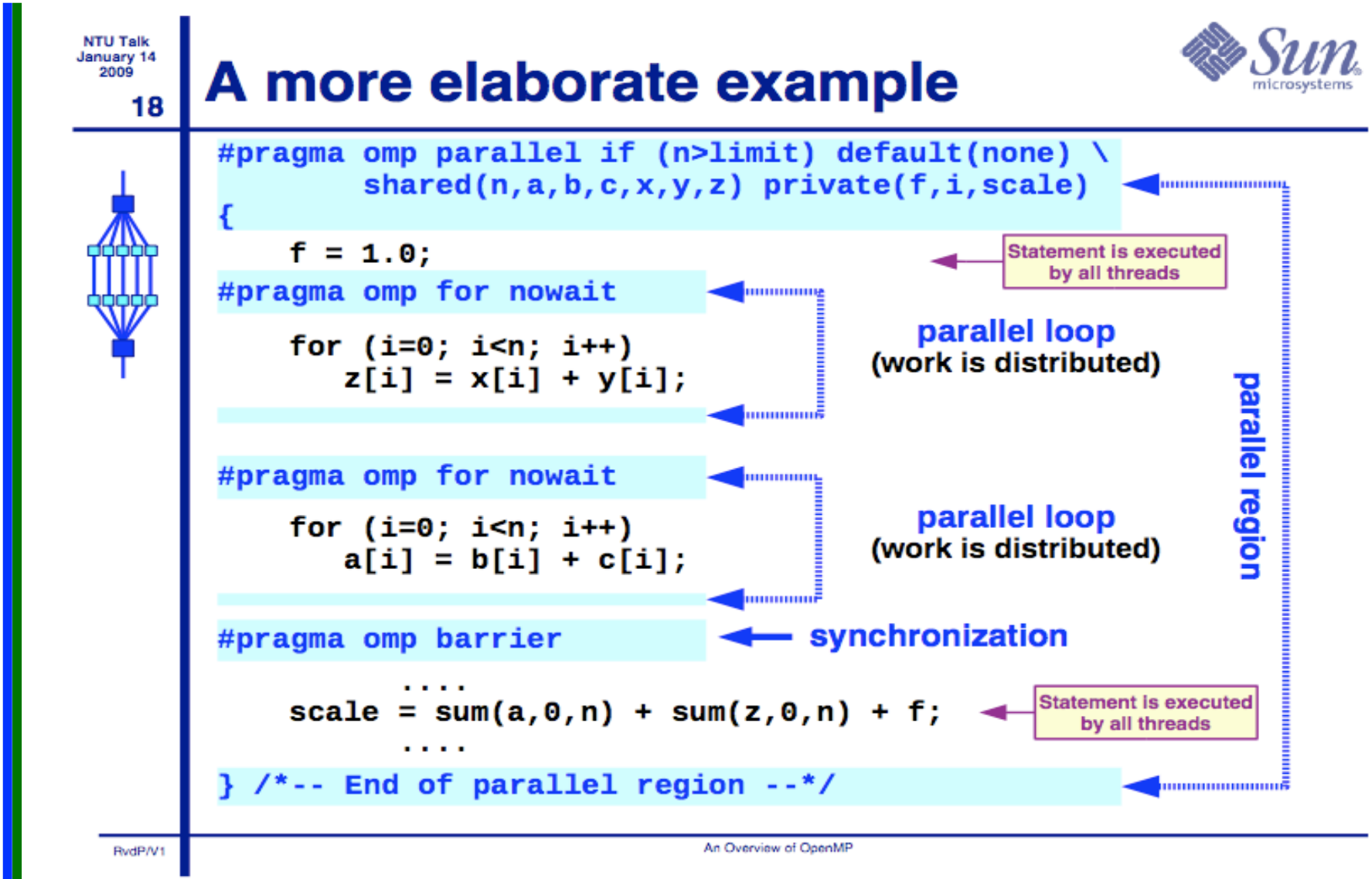
Outro Exemplo Sintético

- Descrição detalhada
  - 1º pragma
    - Define uma região paralela
    - Executada em paralelo se condição verdadeira
      - n > limit
  - Comando sem pragma
    - Executado por todas as threads
  - 2º pragma
    - Define um loop paralelo

Outro Exemplo Sintético

- Descrição detalhada
  - 3º pragma
    - Define um 2º loop paralelo
  - 4º pragma
    - Define uma barreira

Um Exemplo Sintético +



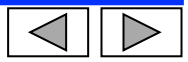
Informática UFRGS

OpenMP em mais Detalhes

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Termos e Comportamento

- Time OpenMP := Master + Workers
- Região Paralela
  - Bloco de código executado por todas as threads simultaneamente
  - Thread master sempre tem ID = 0
  - Ajuste de threads (se permitido) é realizado somente antes do início da execução da região
  - Regiões podem ser aninhadas mas esse recurso é dependente de implementação
  - Uma cláusula "if" pode ser usada como guarda
    - Se avaliada como "falsa", o código da região é executado sequencialmente



Termos e Comportamento

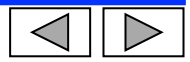
- Construção “work-sharing”
  - Divide a execução do código da região entre os membros do team

Informática UFRGS

Cláusula If

- Cláusula If
  - Sintaxe
    - if (expressão-escalar)
  - Somente executa em paralelo se expressão é avaliada em “true”
  - Caso contrário, executa sequencialmente

```
#pragma omp parallel if (n > threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```



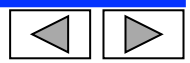
UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Informática UFRGS

Cláusula Shared

- Cláusula Shared
  - Sintaxe
    - shared (lista-de-variáveis)
  - Dados são acessíveis a todas as threads
  - Dados acessados no mesmo endereço

```
#pragma omp parallel if (n > threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```



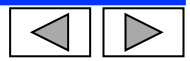
UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Informática UFRGS

Cláusula Private

- Cláusula Private
  - Sintaxe
    - private (lista-de-variáveis)
  - Todas as referências são locais
  - Valores indefinidos na entrada e saída da região
  - Não há associação com dado (variável) original

```
#pragma omp parallel if (n > threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

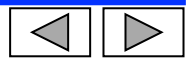
Informática UFRGS

Barreira /1

- Barreira
  - Supondo execução em paralelo dos 2 loops abaixo
  - Uma execução pode gerar algum erro (resultado inconsistente)?
  - Porque?

```
for (i=0; i < N; i++)
   a[i] = b[i] + c[i];
```

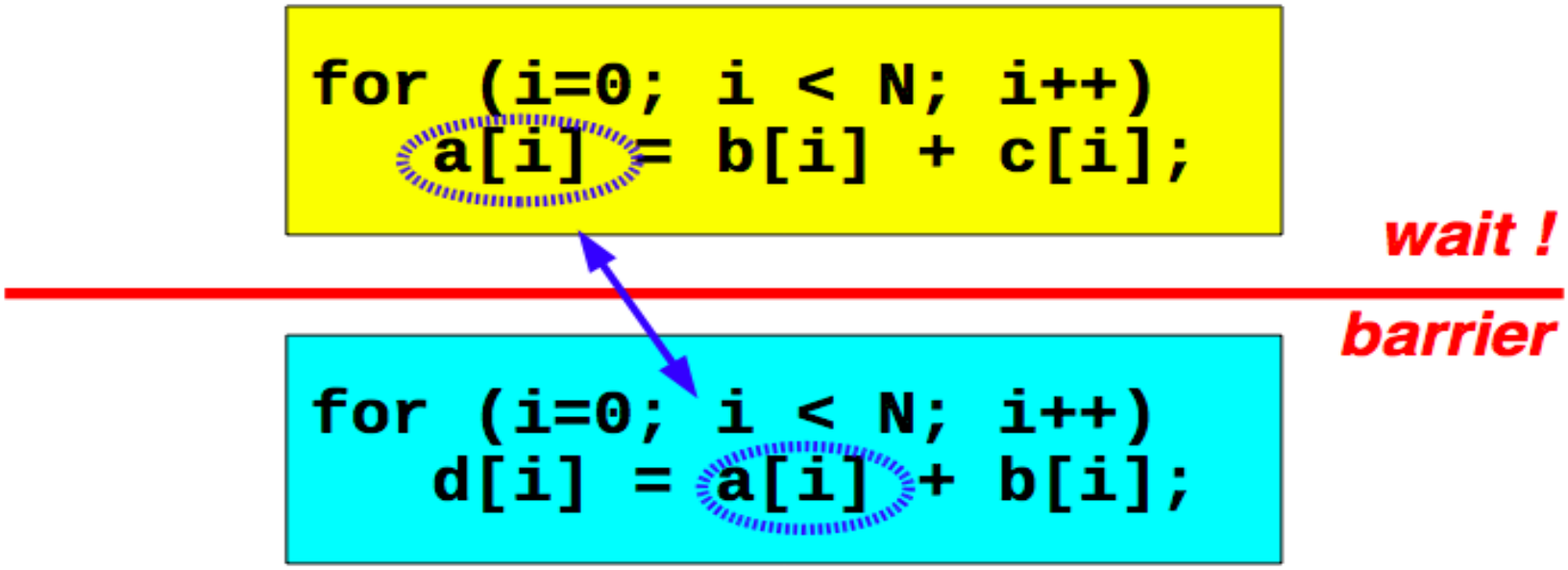
```
for (i=0; i < N; i++)
   d[i] = a[i] + b[i];
```



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

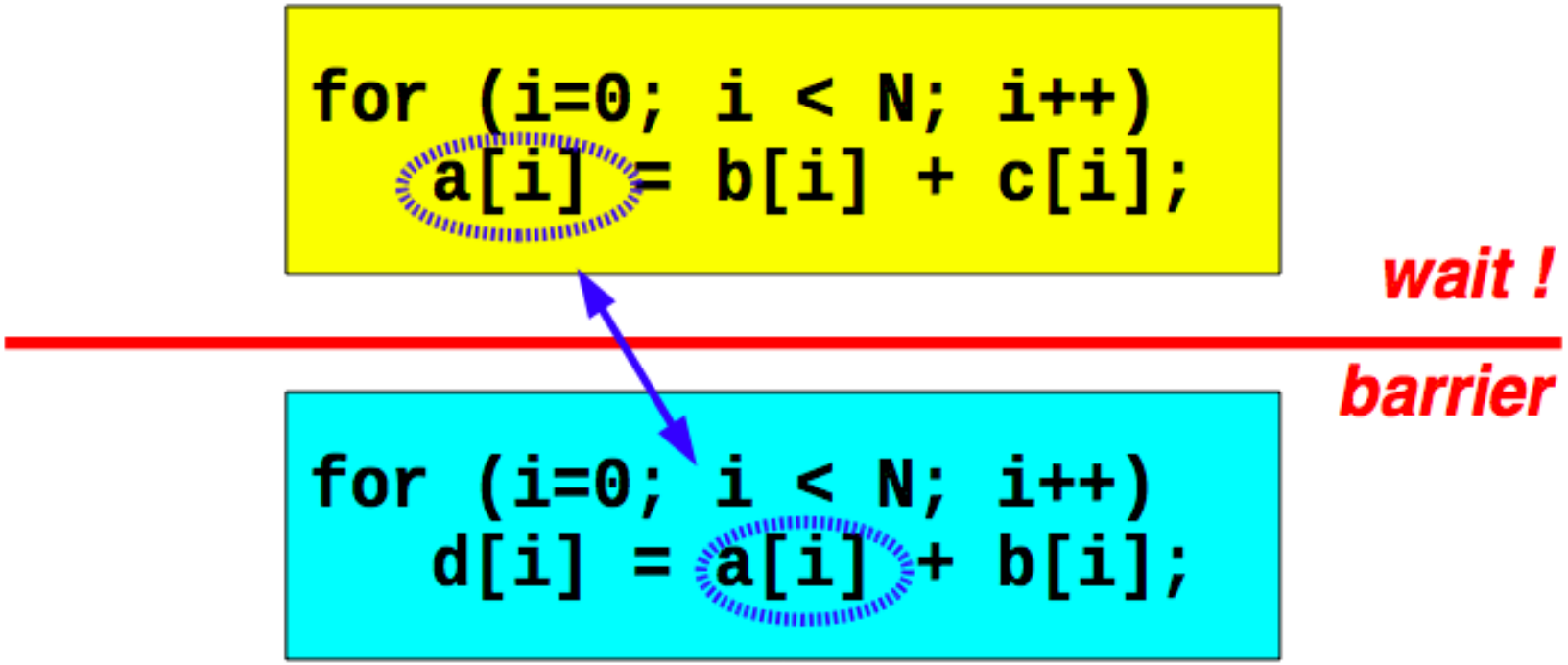
Barreira /2

- Barreira
  - É neccessário atualizar todo o *a[]* antes de usa-lo
  - Seria possível se as iterações de ambos os loops fossem mapeadas às mesmas threads



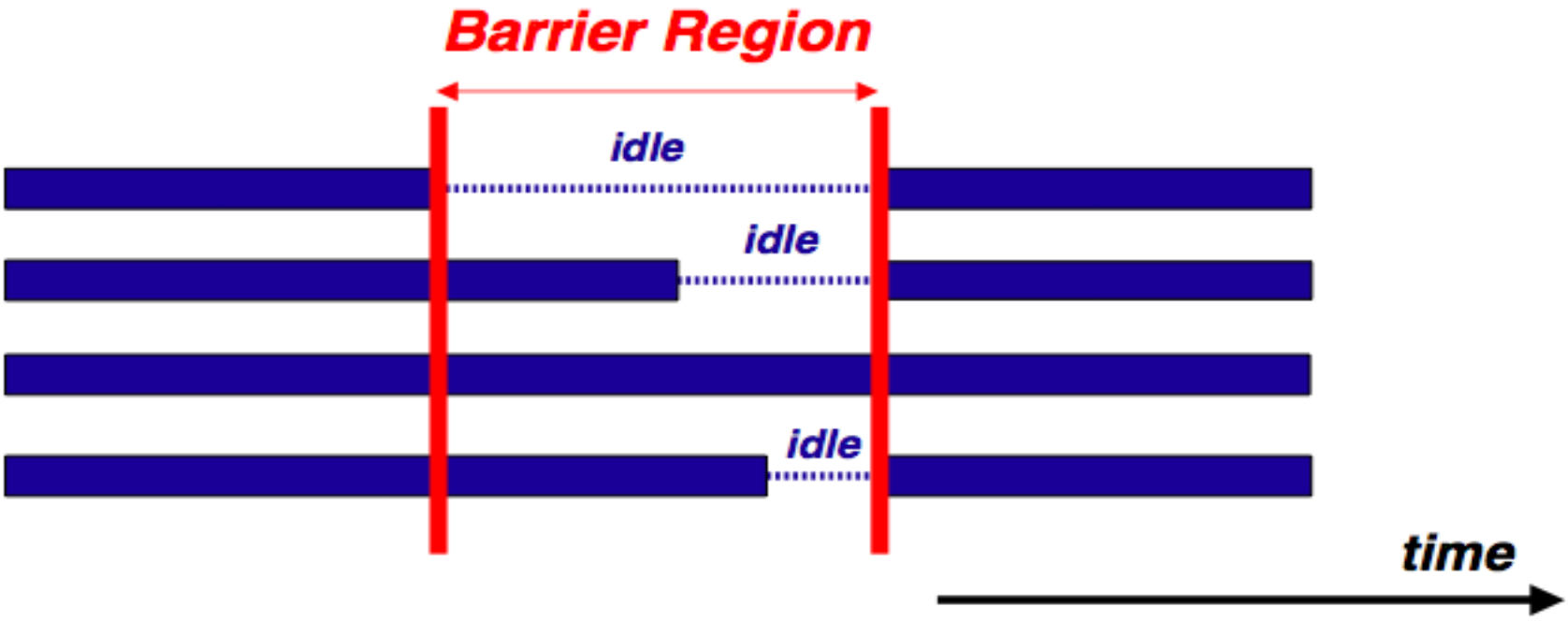
Barreira /3

- Barreira
  - Todas as threads esperam na barreira e só continuam após todas terem atingido a barreira



Barreira /4

- Barreira
  - Diagrama de tempo
  - Algumas threads (cores?) podem ficar ociosas



Barreira /5

- Barreira
  - Sintaxe

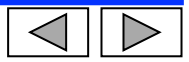
```
#pragma omp barrier
```

```
!$omp barrier
```

Informática UFRGS

Cláusula Nowait

- Cláusula Nowait
  - Para minimizar custo de sincronização, algumas diretivas OpenMP suportam a cláusula “nowait”
  - A cláusula é opcional
  - Se usada, threads não esperam (sincronizam) ao final da construção (associada à “nowait”)



UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Cláusula Nowait

- Cláusula Nowait
  - Sintaxe
    - Em Fortran, é colocada no final da diretiva
    - Em C, é uma cláusula ao lado do pragma

```
#pragma omp for nowait
{
        :
}
```

```
!$omp do
        :
        :
!$omp end do nowait
```

Região Paralela

- Região Paralela
  - É um bloco de código executado por múltiplas threads simultaneamente
  - Sintaxe (Fortran e C)

```
!$omp parallel [clause[[,] clause] ...]

    "this is executed in parallel"

!$omp end parallel (implied barrier)
```

```
#pragma omp parallel [clause[[,] clause] ...]
{
    "this is executed in parallel"

} (implied barrier)
```

Construtores em Work-Sharing

- Construtores (diretivas) em Work-Sharing
  - for, sections, single

```
#pragma omp for
{
    ....
}

!$OMP DO
    ....
!$OMP END DO
```

```
#pragma omp sections
{
        ....
}

!$OMP SECTIONS
        ....
!$OMP END SECTIONS
```

```
#pragma omp single
{
        ....
}

!$OMP SINGLE
        ....
!$OMP END SINGLE
```

Construtores em Work-Sharing

- Construtores (diretivas) em Work-Sharing
  - O trabalho é distribuído sobre as threads
  - O trabalho deve estar embutido na região paralela
  - Deve ser "encontrado" por todas as threads do time ou nenhuma
  - Não há barreira implícita na entrada
  - Há uma barreira implícita na saída
    - Possível exceção com cláusula "nowait"
  - Um construtor "work-sharing" não dispara novas threads

Construtores em Work-Sharing

- Construtores (diretivas) em Work-Sharing
  - Fortran possui mais um construtor: “workshare”
  - Sintaxe

```
!$OMP WORKSHARE

    <array syntax>

!$OMP END WORKSHARE [NOWAIT]
```

Construtores em Work-Sharing

- Construtores (diretivas) em Work-Sharing
  - Fortran construtor: “workshare”
  - Exemplo
    - Soma de 2 vetores é executada como um loop paralelo

```
!$OMP WORKSHARE
    A(1:M) = A(1:M) + B(1:M)
!$OMP END WORKSHARE NOWAIT
```

Construtor for/do

- Construtor “for/do”
  - As iterações do loop são distribuídas para as threads
  - Sintaxe

```
#pragma omp for [clause[[,] clause] ...]
   <original for-loop>
```

```
!$omp do [clause[[,] clause] ...]
        <original do-loop>
!$omp end do [nowait]
```

Construtor for/do

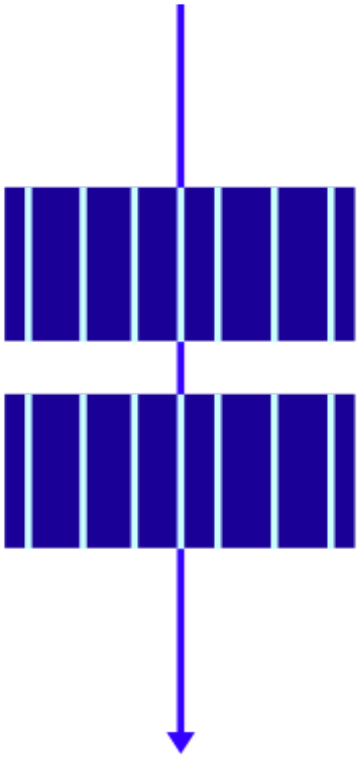
- Construtor “for/do”
  - Cláusulas suportadas
    - private
    - firstprivate
    - lastprivate
    - reduction
    - ordered
    - schedule
    - nowait

Construtor for/do

- Construtor “for/do”
  - Exemplo

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp for nowait
      for (i=0; i<n-1; i++)
          b[i] = (a[i] + a[i+1])/2;
    #pragma omp for nowait
      for (i=0; i<n; i++)
          d[i] = 1.0/c[i];

  } /*-- End of parallel region --*/
```

(implied barrier)



Construtor Sections

- Construtor “Sections”
  - Os blocos de código individuais são distribuídos sobre as threads
  - Os blocos “section” devem estar dentro par “{}”

```
#pragma omp sections [clause(s)]
{
#pragma omp section
          <code block1>
#pragma omp section
          <code block2>
#pragma omp section
            :
}
```

```
!$omp sections [clause(s)]
!$omp section
          <code block1>
!$omp section
          <code block2>
!$omp section
            :
!$omp end sections [nowait]
```

Informática UFRGS

Construtor Sections

- Construtor “Sections”
  - Claúsulas suportadas
    - private
    - firstprivate
    - lastprivate
    - reduction
    - nowait

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

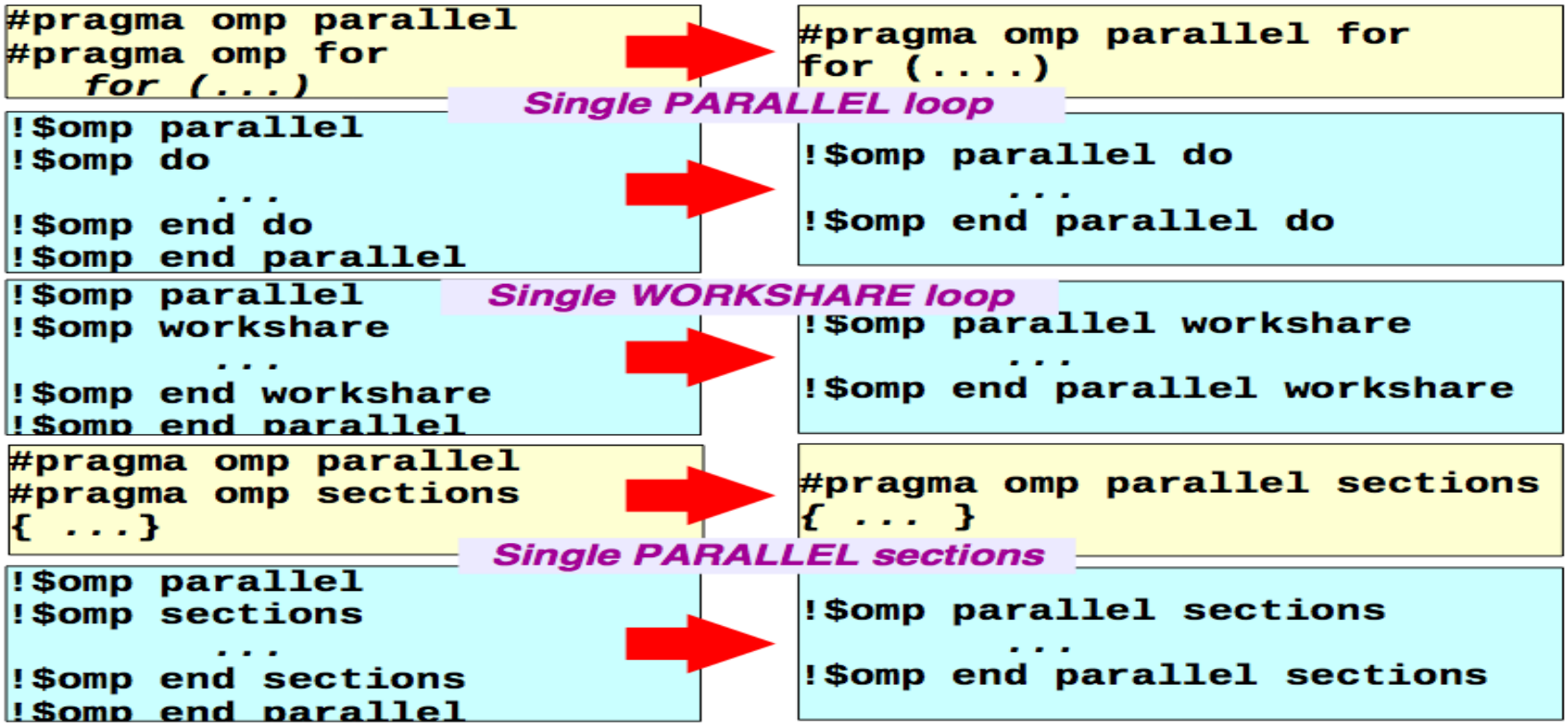
Construtor Sections

- Construtor “Sections”
  - Exemplo

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp sections nowait
    {
      #pragma omp section
        for (i=0; i<n-1; i++)
          b[i] = (a[i] + a[i+1])/2;
      #pragma omp section
        for (i=0; i<n; i++)
          d[i] = 1.0/c[i];
    } /*-- End of sections --*/

  } /*-- End of parallel region --*/
```

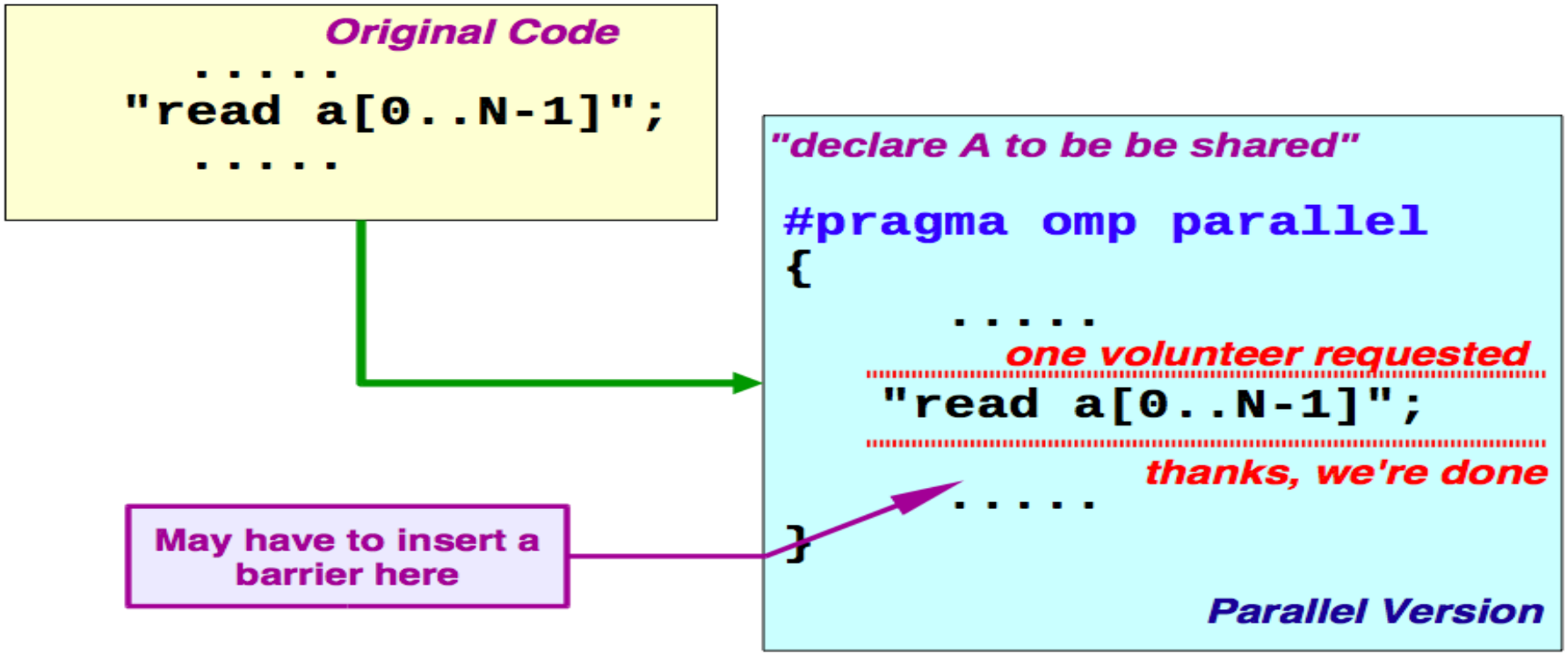
Combinação construtores

- Combinação de construtores
  - Esquema



Construtor Single

- Construtor “Single”
  - Apropriado para rotinas de I/O e inicializações

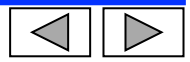


Construtor Single

- Construtor “Single”
  - Somente uma thread (qualquer) executa o bloco

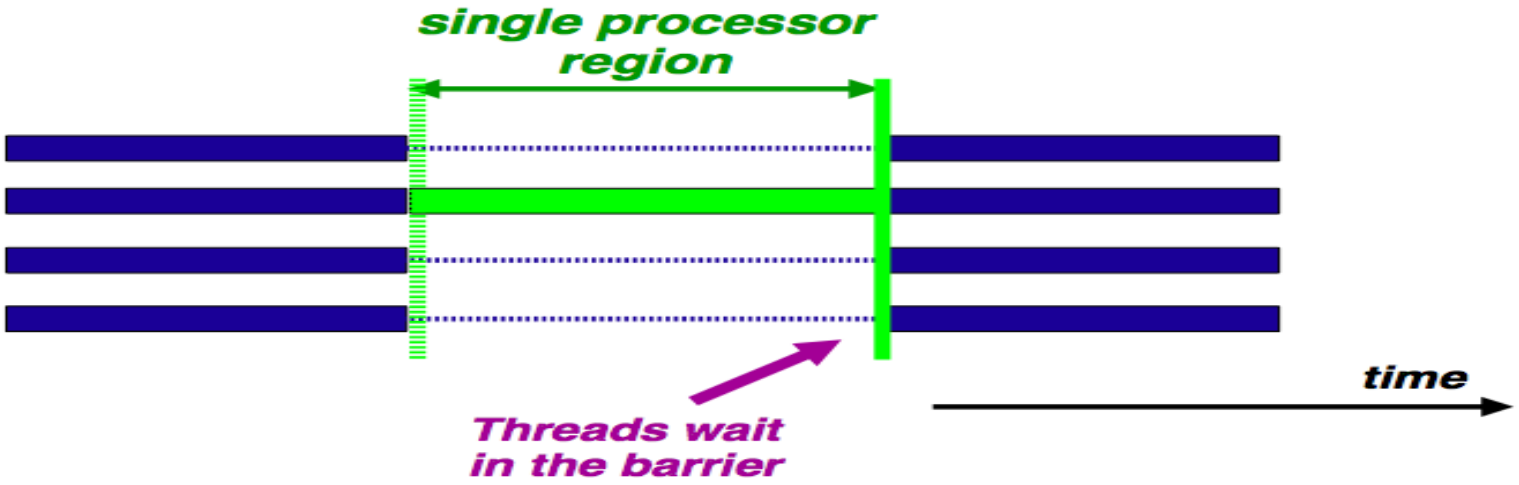
```
#pragma omp single [private][firstprivate] \
                   [copyprivate][nowait]
{
        <code-block>
}
```

```
!$omp single [private][firstprivate]
   <code-block>
!$omp end single [copyprivate][nowait]
```



Construtor Single

- Construtor “Single”
  - Usualmente, há uma barreira implícita no final da região
  - Pode acarretar um problema (gargalo) de escalabilidade (lei de Amdahl)
    - Aumento da zona sequencial



Construtor Single

- Construtor “Single”
  - Lei de Amdahl
    - Ganho em desempenho quando se otimiza parte de um processamento
    - Ganho limitado à fração de processamento da parte otimizada
    - Fm: fração de processamento da parte otimizada
      - Sempre menor que 1
    - Ge: ganho de desempenho com a otimização (pode ser paralelização)
      - Sempre maior que 1

$$ganho = \frac{1}{(1 - Fm) + \frac{Fm}{Ge}}$$

Construtor Single

- Construtor “Single”
  - Lei de Amdahl
    - Exemplo
      - Fm: 10%
      - Ge: 2; por exemplo, speedup 2 por se usar 2 cpus na parte dada por Fm
      - ganho (final) = 1,052

$$ganho = \frac{1}{(1 - Fm) + \frac{Fm}{Ge}}$$

Construtor Master

- Construtor “Master”
  - Somente a thread master executa o bloco de código

```
#pragma omp master
{<code-block>}
```

```
!$omp master
        <code-block>
!$omp end master
```

There is no implied barrier on entry or exit !

Região Crítica

- Região (seção) crítica
  - Se "sum" é uma variável compartilhada, o loop não pode ser executado em paralelo

```
for (i=0; i < N; i++){
      .....
   sum += a[i];
      .....
}
```

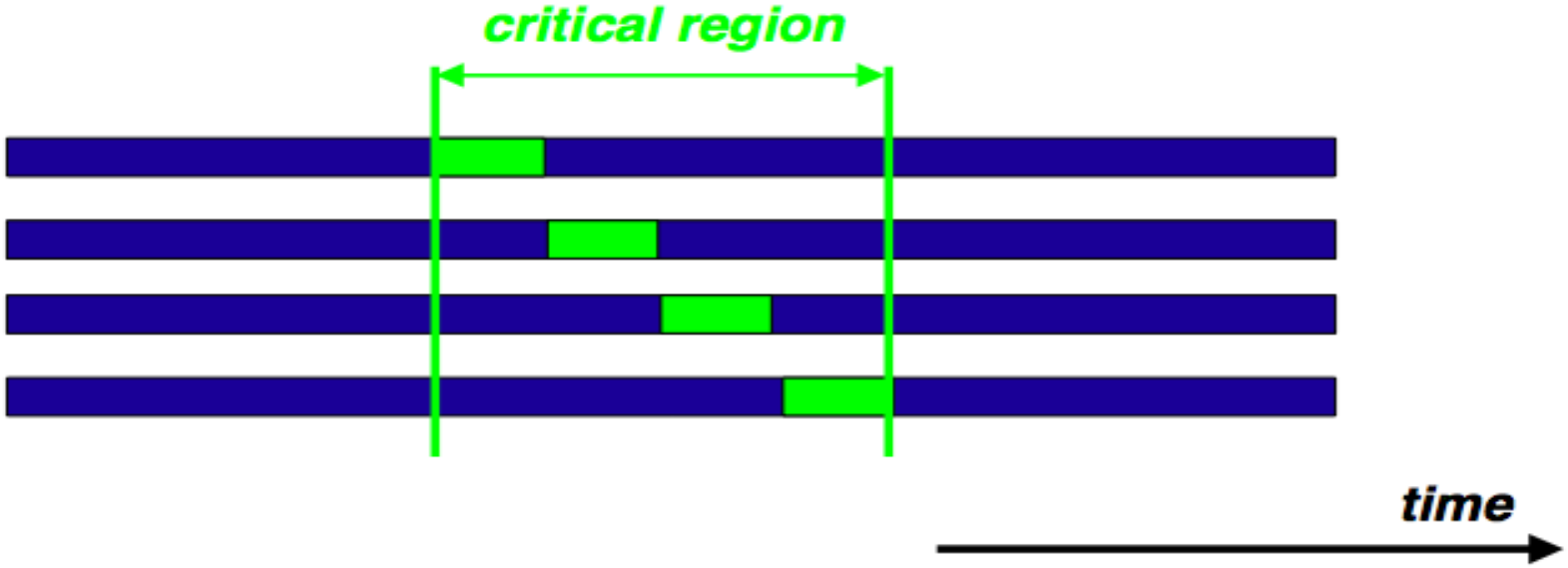
Região Crítica

- Região (seção) crítica
  - Pode-se usar uma seção crítica para resolver o problema

```
for (i=0; i < N; i++){
        .....                    one at a time can proceed
    sum += a[i];
        .....                    next in line, please
}
```

Região Crítica

- Região (seção) crítica
  - Útil
    - Para evitar condições de corrida ou para executar I/O mas ainda em ordem randômica



Construtores Critical e Atomic

- Construtor “Critical”
  - Todas as threads executam o código mas uma de cada vez
  - Sem barreira implícita na entrada e na saída

```
#pragma omp critical [(name)]
{<code-block>}
```

```
!$omp critical [(name)]
        <code-block>
!$omp end critical [(name)]
```

There is no implied barrier on entry or exit !

Construtores Critical e Atomic

- Construtor “Atomic”
  - Somente os “loads” e os “stores” são atômicos

```
#pragma omp atomic
   <statement>
```

```
!$omp atomic
   <statement>
```

This is a lightweight, special form of a critical section

```
#pragma omp atomic
    a[indx[i]] += b[i];
```

Novidades com OpenMP 3.0

- Novidade com OpenMP 3.0
  - Suporte para threads
  - Maior número (tipos) de aplicações poderão ser paralelizadas

Novidades com OpenMP 3.0

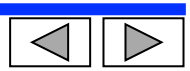
- Novidade com OpenMP 3.0
  - Exemplo Lista Encadeada
    - Uma tarefa independente para cada item da lista
    - Sem OpenMP 3.0
      - Inicialmente contar # de iterações
      - Então transformar “while loop” em “for loop”

Novidades com OpenMP 3.0

- Novidade com OpenMP 3.0
  - Exemplo Lista Encadeada

```
        ........
while(my_pointer) {

        (void) do_independent_work (my_pointer);

  my_pointer = my_pointer->next ;
} // End of while loop

        ........
```

Hard to do before OpenMP 3.0:
First count number of iterations, then convert while loop to for loop



Novidades com OpenMP 3.0

- Novidade com OpenMP 3.0
  - Exemplo Lista Encadeada
    - Solução com OpenMP 3.0

```
my_pointer = listhead;

#pragma omp parallel
{
    #pragma omp single nowait
    {
        while(my_pointer) {
          #pragma omp task firstprivate(my_pointer)
          {
              (void) do_independent_work (my_pointer);
          }
          my_pointer = my_pointer->next ;
        }
    } // End of single - no implied barrier (nowait)
} // End of parallel region - implied barrier
```

OpenMP Task is specified here (executed in parallel)

Informática UFRGS

ESTUDO DE CASO: REDE NEURAL

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
  - Aplicação: rede neural
  - Baixo desempenho em sistema SMP da Sun
  - Analisador de desempenho mostra quais rotinas consomem mais tempo de cpu

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
  - Saída de analisador de desempenho
  - Observar: "calc_r_loop_on..."

Excl. User CPU sec.	%	Incl. User CPU sec.	Excl. Wall sec.	Name
120.710	100.0	120.710	128.310	<Total>
116.960	96.9	116.960	122.610	calc_r_loop_on_neighbours
0.900	0.7	118.630	0.920	calc_r
0.590	0.5	1.380	0.590	_doprnt
0.410	0.3	1.030	0.430	init_visual_input_on_V1
0.280	0.2	0.280	1.900	_write
0.200	0.2	0.200	0.200	round_coord_cyclic
0.130	0.1	0.130	0.140	__arint_set_n
0.130	0.1	0.550	0.140	__k_double_to_decimal
0.090	0.1	1.180	0.090	fprintf

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
  - Saída de analisador de desempenho
    - Somente fragmentos de rotina “calc_r” e ...

```
Attr. User   Excl. User   Incl. User   Name
CPU sec.     CPU sec.     CPU sec.
116.960        0.900      118.630       calc_r
116.960      116.960      116.960        *calc_r_loop_on_neighbours
```

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
  - Análise mais fina: comando em vermelho consome 96% do tempo

```
     struct cell{
       double x; double y; double r; double I;
     };
                          ......

     struct cell V1[NPOSITIONS_Y][NPOSITIONS_X];
     double       h[NPOSITIONS][NPOSITIONS];
                          ......
   Excl. User CPU  Excl. Wall
     sec.      %       sec.
                                1040. void
                                1041. calc_r_loop_on_neighbours
                                            (int y1, int x1)
     0.080    0.1      0.080   1042. {
                                1043. struct interaction_structure *next_p;
                                1044.
     0.130    0.1      0.130   1045. for (next_p = JJ[y1][x1].next;
     0.460    0.4      0.470   1046.       next_p != NULL;
                                1047.       next_p = next_p->next) {
## 116.290   96.3    121.930   1048.      h[y1][x1] += next_p->strength *
                                            V1[next_p->y][next_p->x].r;
                                1049.
                                1052.    }
                                1053. }
```

What is the problem ?

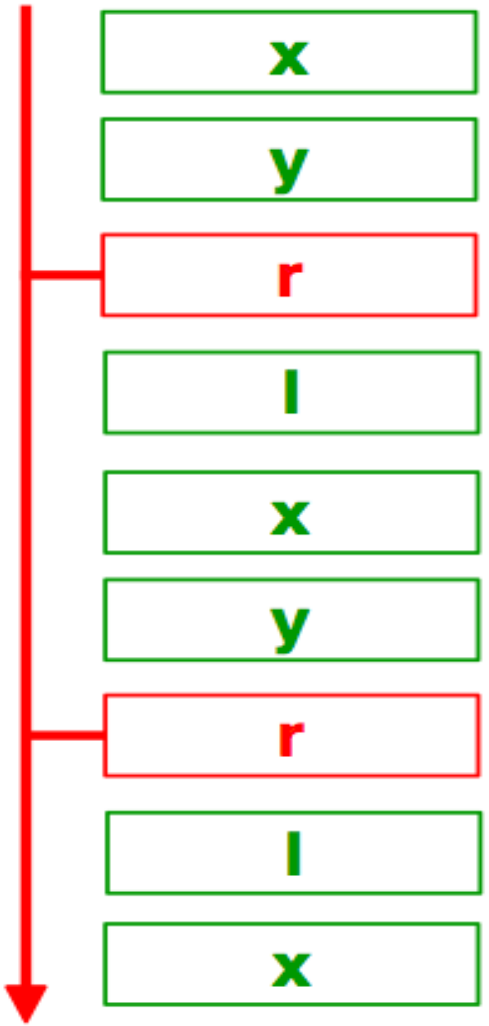
96% of the time spent in this single statement

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
  - Problema de estrutura de dados
    - Se usa somente ¼ da linha de cache
    - Para problemas suficientemente grandes
      - Haverá tráfego em memória adicional
        - Pressão de interconexão
      - Perda de capacidade de cache de dados
        - Redução de localidade temporal

Estudo de Caso Rede Neural

- Estudo de caso: melhora …
  - Problema de estrutura de dados
    - Tudo isto afeta negativamente tanto o desempenho sequencial quanto o paralelo
    - Solução
      - Dividir a estrutura em 2 partes
      - Uma contém somente os valores de "r"
      - A outra contém os conjuntos {x, y, l}



Estudo de Caso Rede Neural

- Estudo de caso: melhora …
  - Fragmento da modificação de código

```
double V1_R[NPOSITIONS_Y][NPOSITIONS_X];

void
calc_r_loop_on_neighbours(int y1, int x1)
{
  struct interaction_structure *next_p;

  double sum = h[y1][x1];

  for (next_p = JJ[y1][x1].next;
       next_p != NULL;
       next_p = next_p->next) {
       sum += next_p->strength * V1_R[next_p->y][next_p->x];
  }
  h[y1][x1] = sum;
}
```

Estudo de Caso Rede Neural

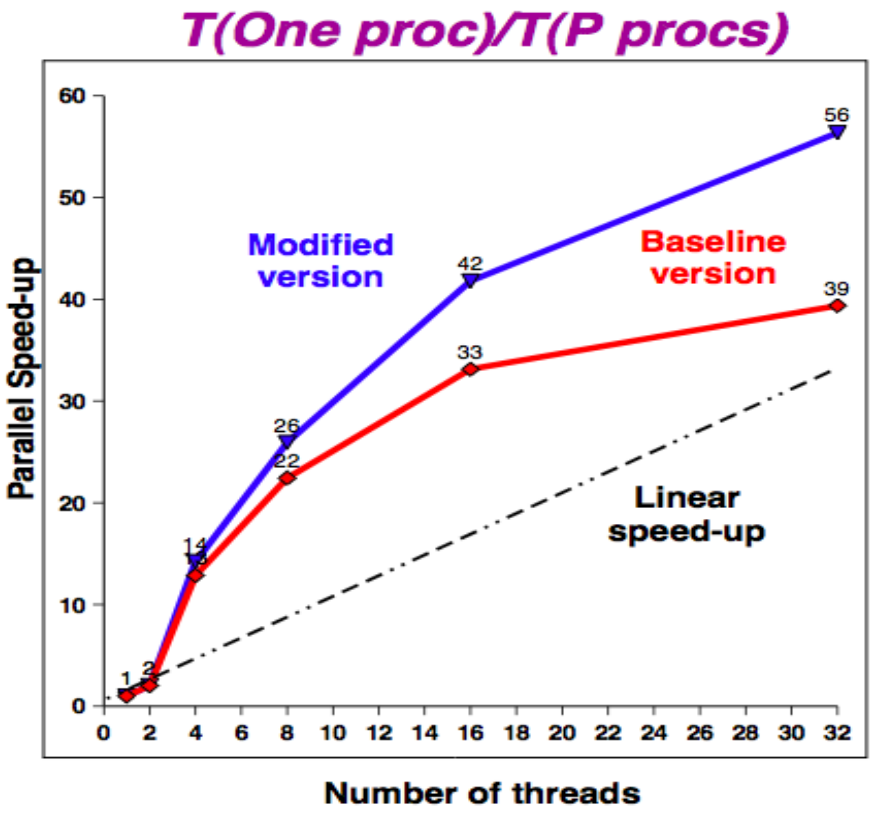
- Estudo de caso: melhora …
  - Paralelização com OpenMP

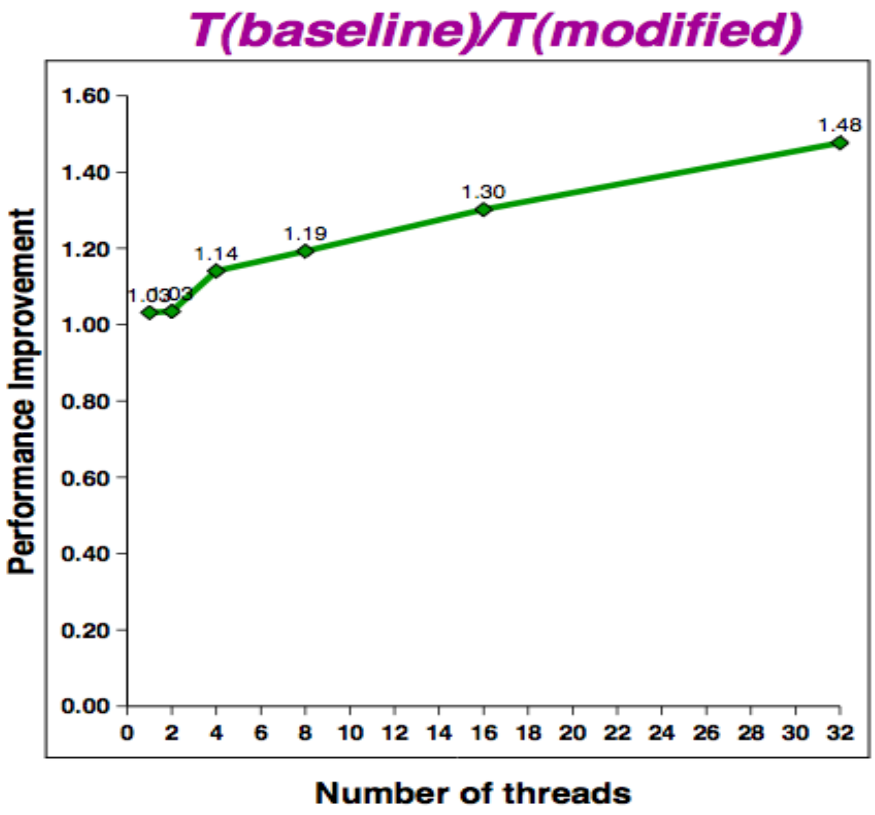
```
void calc_r(int t)
{

#include <omp.h>

#pragma omp parallel for default(none)  \
        private(y1,x1) shared(h,V1,g,T,beta_inv,beta)
  for (y1 = 0; y1 < NPOSITIONS_Y; y1++) {
    for (x1 = 0; x1 < NPOSITIONS_X; x1++) {

      calc_r_loop_on_neighbours(y1,x1);
      h[y1][x1] += V1[y1][x1].I;

      <statements deleted>

    }
  }

/*-- End of OpenMP parallel for --*/
```

Can be executed in parallel

Estudo de Caso Rede Neural

- Estudo de caso: melhora …
  - Resultados (melhoria) com OpenMP





Note:
Single processor run time is 5001 seconds for the baseline version (4847 for the modified version)

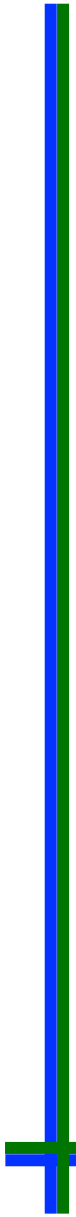
Instituto de Informática

OpenMP:
Uma Introdução

Cláudio Geyer

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Informática
UFRGS



Exercícios

- Exercícios:
  - A)

Revisão

- Revisão

Informática UFRGS

Referências

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL