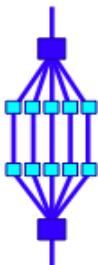


Instituto de Informática

OpenMP: Uma Introdução

Cláudio Geyer

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

Sumário

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

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:
 - 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

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>

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:
 - 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:
 - 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, Mattson and 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:
 - 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/>

Organizações OpenMP



The OpenMP logo features the text "OpenMP" in a large, teal, sans-serif font. The "O" is significantly larger than the other letters. A horizontal teal bar is positioned above the text, and another horizontal teal bar is positioned below the "Open" portion of the text. A small "TM" trademark symbol is located to the right of the "P".

<http://www.openmp.org>



The OMPunity logo consists of the text "OMPunity" in a teal, sans-serif font. The "OMP" is larger and more prominent than "unity". A horizontal teal bar is positioned above the text, and another horizontal teal bar is positioned below the text.

<http://www.compunity.org>





The screenshot shows the OpenMP.org website with the following content:

OpenMP THE OPENMP API SPECIFICATION FOR PARALLEL PROGRAMMING

OpenMP News

»IWOMP 2009 in June - Important Dates

International Workshop on OpenMP
IWOMP 2009
Evolving OpenMP in an Age of Extreme Parallelism

June 3rd - June 5th, 2009
Dresden, Germany
<http://www.iwomp.org>

The 2009 International Workshop on OpenMP (IWOMP 2009) will be held on the campus of Technische Universität Dresden, Germany. IWOMP is the premier event focusing on parallel programming with OpenMP. The workshop serves as a forum to present the latest research ideas and results related to this shared memory programming model. It also offers the opportunity to interact with OpenMP users, developers and the people working on the next release of the standard.

The first day consists of tutorials focusing on topics of interest to current and prospective OpenMP developers, suitable for both beginners as well as those interested in learning of recent developments in the evolving OpenMP standard.

The second and third day consists of two keynotes, one invited talk, 15 technical papers and a poster session during which research ideas and results will be presented and discussed. The keynotes "Is OpenMP the right approach for future generation architectures?" by Jose Duato Marin, Universidad Politecnica de Valencia and "Can OpenMP be extended to deal with Hardware Accelerator?" by François Bodin, CAPS Enterprise provide the framework for the other presentations.

To enable the exchange of information regarding latest developments it is still possible to submit a poster for all registered workshop participants until May 20 via email to iwomp09@zih.tu-dresden.de.

Important Dates

APRIL 22: Downtown Hotel Deadline
APRIL 24: Guesthouse Deadline
MAY 18: Early Registration Deadline
MAY 20: Poster Deadline
MAY 30: Late Registration Deadline
Tutorial and Workshop in Dresden: June 3-5, 2009

Posted on April 21, 2009

»Download Book Examples and Discuss

The OpenMP API
supports multi-platform shared-memory parallel programming in C/C++ and Fortran. OpenMP is a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer.
»Read about OpenMP

Get
»OpenMP specs

Use
»OpenMP Compilers

Learn

Using OpenMP

PORTABLE SHARED MEMORY PARALLEL PROGRAMMING

BARBARA CHAPMAN, GABRIEL COFFI, BARBARA L. LISKOV, AND KENNETH M. YONGE

»Using OpenMP – the book
»Using OpenMP – the examples
»Using OpenMP – the forum
»Wikipedia
»OpenMP Tutorial
»More Resources

Discuss

Subscribe to the News Feed

» Specifications
» About OpenMP
» Compilers
» Resources
» Discussion Forum

Events
The 5th International Workshop on OpenMP - Evolving OpenMP in an Age of Extreme Parallelism - will take place June 3-5, 2009 in Dresden, Germany.
»iwomp.org

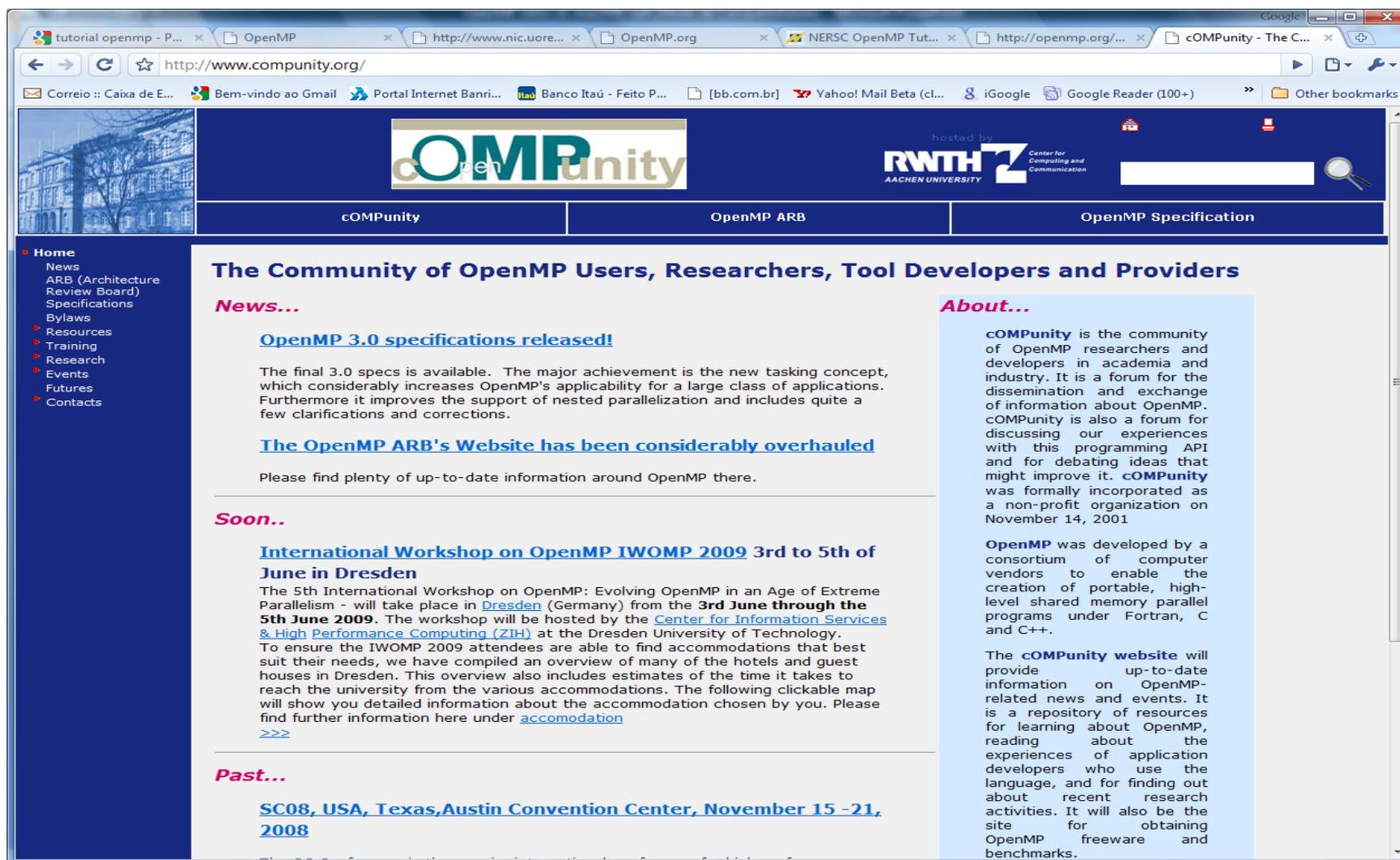
ISC 09 - Hamburg, Germany - June 23-26.
»www.supercomp.de/isc09/

Input Register
Alert the OpenMP.org webmaster about new products, events, or updates and we'll post it here.
»webmaster@openmp.org

Search OpenMP.org
Google Custom Search
Search

Archives

- April 2009
- March 2009
- February 2009
- January 2009
- November 2008



The screenshot shows the homepage of the OpenMP Community website. The browser address bar displays <http://www.compunity.org/>. The page features a navigation menu with links to **COMPunity**, **OpenMP ARB**, and **OpenMP Specification**. A sidebar on the left lists various categories such as Home, News, ARB (Architecture Review Board), Specifications, Bylaws, Resources, Training, Research, Events, Futures, and Contacts.

The main content area is titled **The Community of OpenMP Users, Researchers, Tool Developers and Providers**. It is organized into three sections:

- News...**
 - [OpenMP 3.0 specifications released!](#)**
The final 3.0 specs is available. The major achievement is the new tasking concept, which considerably increases OpenMP's applicability for a large class of applications. Furthermore it improves the support of nested parallelization and includes quite a few clarifications and corrections.
 - [The OpenMP ARB's Website has been considerably overhauled](#)**
Please find plenty of up-to-date information around OpenMP there.
- Soon...**
 - [International Workshop on OpenMP IWOMP 2009 3rd to 5th of June in Dresden](#)**
The 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism - will take place in [Dresden](#) (Germany) from the **3rd June through the 5th June 2009**. The workshop will be hosted by the [Center for Information Services & High Performance Computing \(ZIH\)](#) at the Dresden University of Technology. To ensure the IWOMP 2009 attendees are able to find accommodations that best suit their needs, we have compiled an overview of many of the hotels and guest houses in Dresden. This overview also includes estimates of the time it takes to reach the university from the various accommodations. The following clickable map will show you detailed information about the accommodation chosen by you. Please find further information here under [accomodation](#)
>>>
- Past...**
 - [SC08, USA, Texas, Austin Convention Center, November 15 -21, 2008](#)**

On the right side, there is an **About...** section:

- cOMPunity** is the community of OpenMP researchers and developers in academia and industry. It is a forum for the dissemination and exchange of information about OpenMP. cOMPunity is also a forum for discussing our experiences with this programming API and for debating ideas that might improve it. **cOMPunity** was formally incorporated as a non-profit organization on November 14, 2001.
- OpenMP** was developed by a consortium of computer vendors to enable the creation of portable, high-level shared memory parallel programs under Fortran, C and C++.
- The **cOMPunity website** will provide up-to-date information on OpenMP-related news and events. It is a repository of resources for learning about OpenMP, reading about the experiences of application developers who use the language, and for finding out about recent research activities. It will also be the site for obtaining OpenMP freeware and benchmarks.

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

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 portátil
 - 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

OpenMP e Multicore

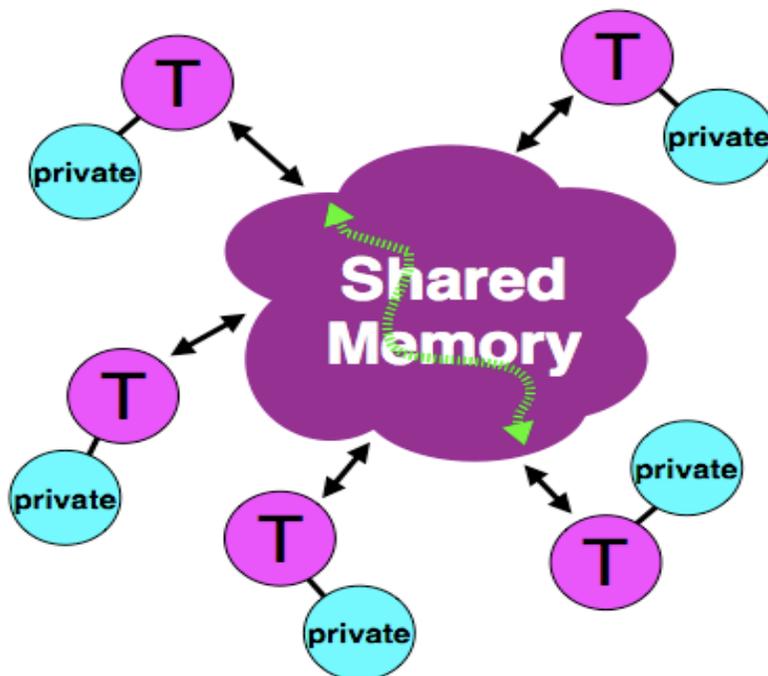
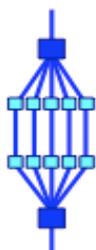
- OpenMP é especialmente indicado para processadores multicore
 - Modelos de memória e de threads podem ser mapeados de forma natural
 - Leve
 - Maduro
 - Muito usado e disponível



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

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



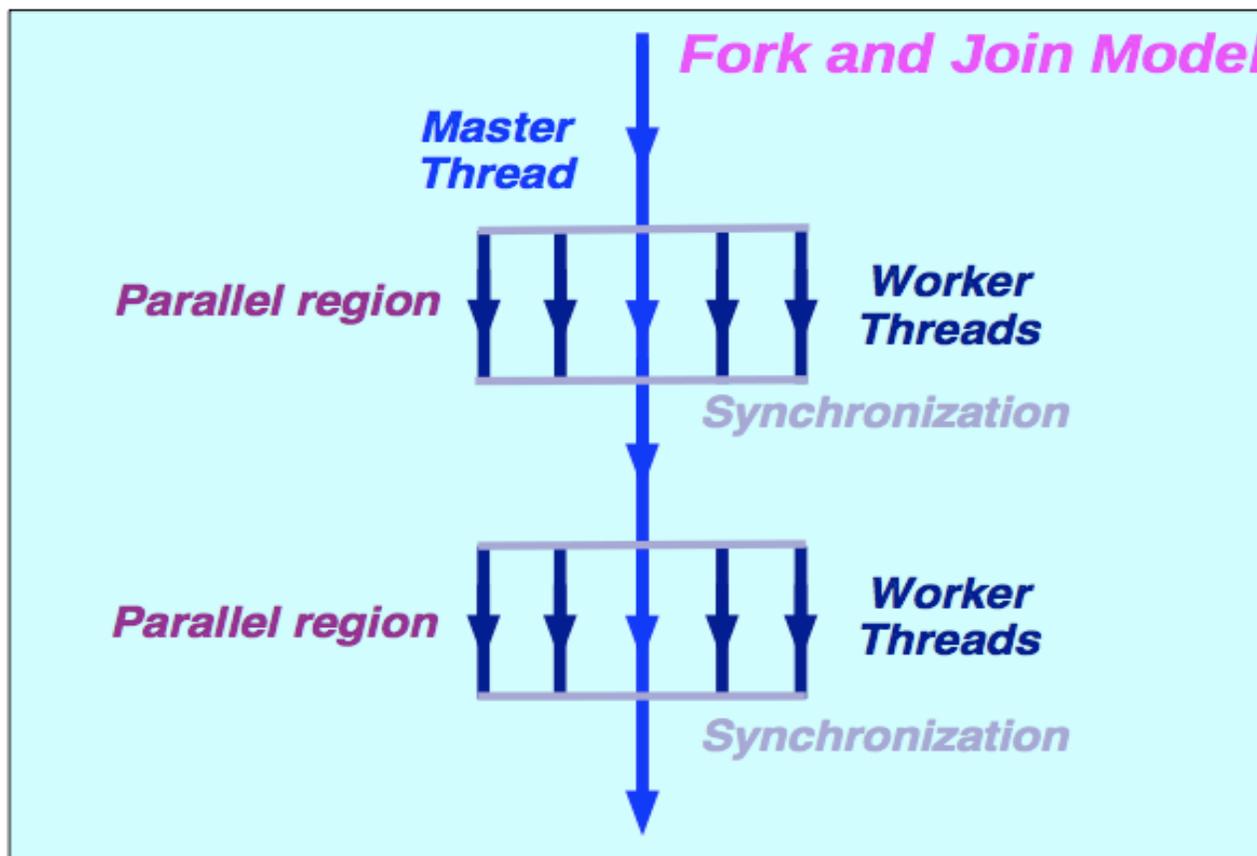
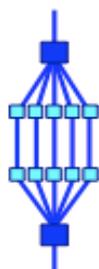
Atributos de Dados Compartilhados

- 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

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

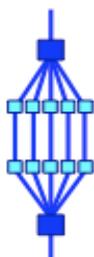
```
#pragma omp parallel for  
for (int i=0; i<n; i++)  
    c[i] = a[i] + b[i];
```

```
% 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

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

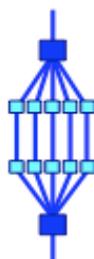
- Ambiente de execução
 - Quantidade de tarefas
 - Id da tarefa
 - Ajuste dinâmico de tarefas
 - Paralelismo aninhado
 - Tempo de parede
 - Bloqueios
- Variáveis de ambiente
 - Quantidade de tarefas
 - Tipo de escalonamento
 - Ajuste dinâmico de tarefas
 - Paralelismo aninhado



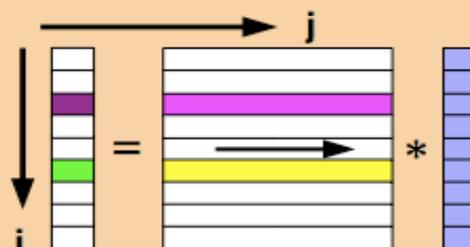
Exemplo Produto Vetorial ($M*V$)

- 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

Example - Matrix times vector



```
#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 =  $\sum$  b[i=0][j]*c[j]
    a[0] = sum
    i = 1
    sum =  $\sum$  b[i=1][j]*c[j]
    a[1] = sum
```

TID = 1

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

... etc ...



Exemplo de Avaliação de Desempenho

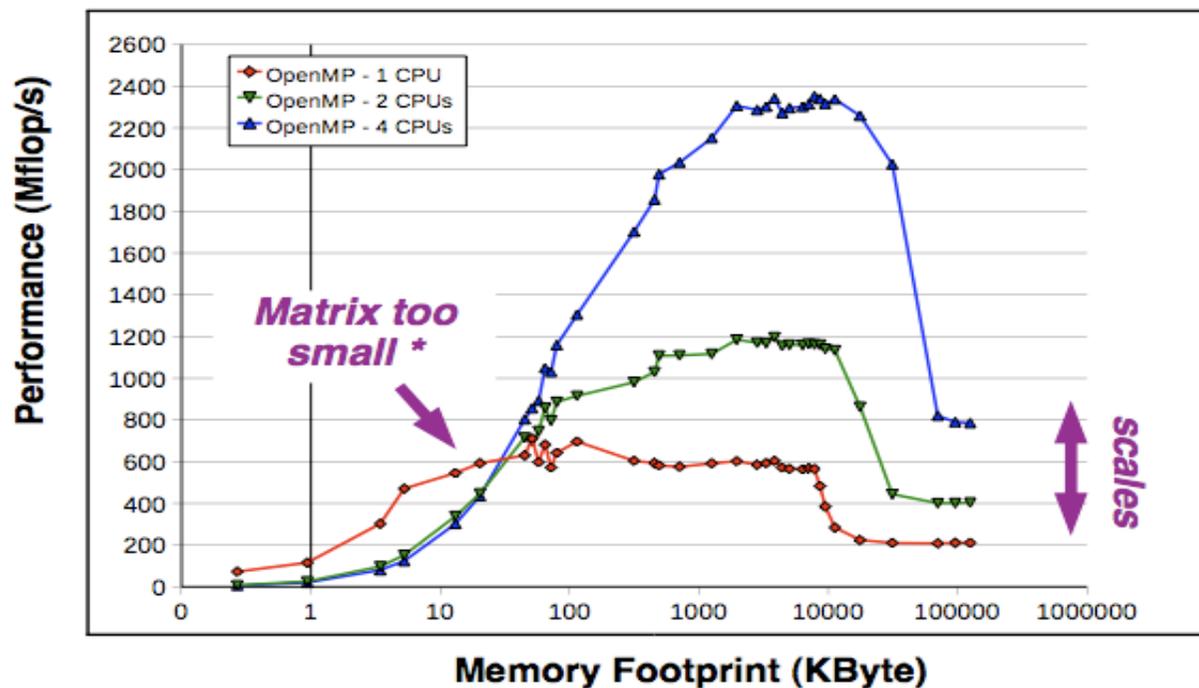
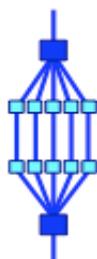
- Exemplo de avaliação de desempenho
 - Execução do mesmo problema com variação do número de cpus
 - Uma curva para cada quantidade de cpus
 - Eixo X:
 - Tamanho do problema
 - Eixo Y:
 - Tempo de execução (tempo paralelo)
 - Notar a perda de desempenho para entradas pequenas

Desempenho de OpenMP

NTU Talk
January 14
2009

17

OpenMP performance



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

RvdPV1

An Overview of OpenMP



Outro Exemplo Sintético

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

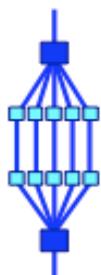
Outro Exemplo Sintético

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

Outro Exemplo Sintético

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

A more elaborate example



```

#pragma omp parallel if (n>limit) default(none) \
    shared(n,a,b,c,x,y,z) private(f,i,scale)
{
    f = 1.0;
    #pragma omp for nowait
    for (i=0; i<n; i++)
        z[i] = x[i] + y[i];

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

    #pragma omp barrier

    scale = sum(a,0,n) + sum(z,0,n) + f;

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

Annotations:

- Statement is executed by all threads**: Points to the initialization of `f = 1.0;` and the final calculation of `scale`.
- parallel loop (work is distributed)**: Points to the first `for` loop.
- parallel loop (work is distributed)**: Points to the second `for` loop.
- synchronization**: Points to the `#pragma omp barrier` statement.
- parallel region**: A vertical label on the right side of the code block, enclosed in a dashed box, indicating the scope of the parallel execution.

OpenMP em mais Detalhes



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

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 --*/
```

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 --*/
```

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 --*/
```

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];
```

Barreira /2

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

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

wait !

barrier

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

Barreira /3

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

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

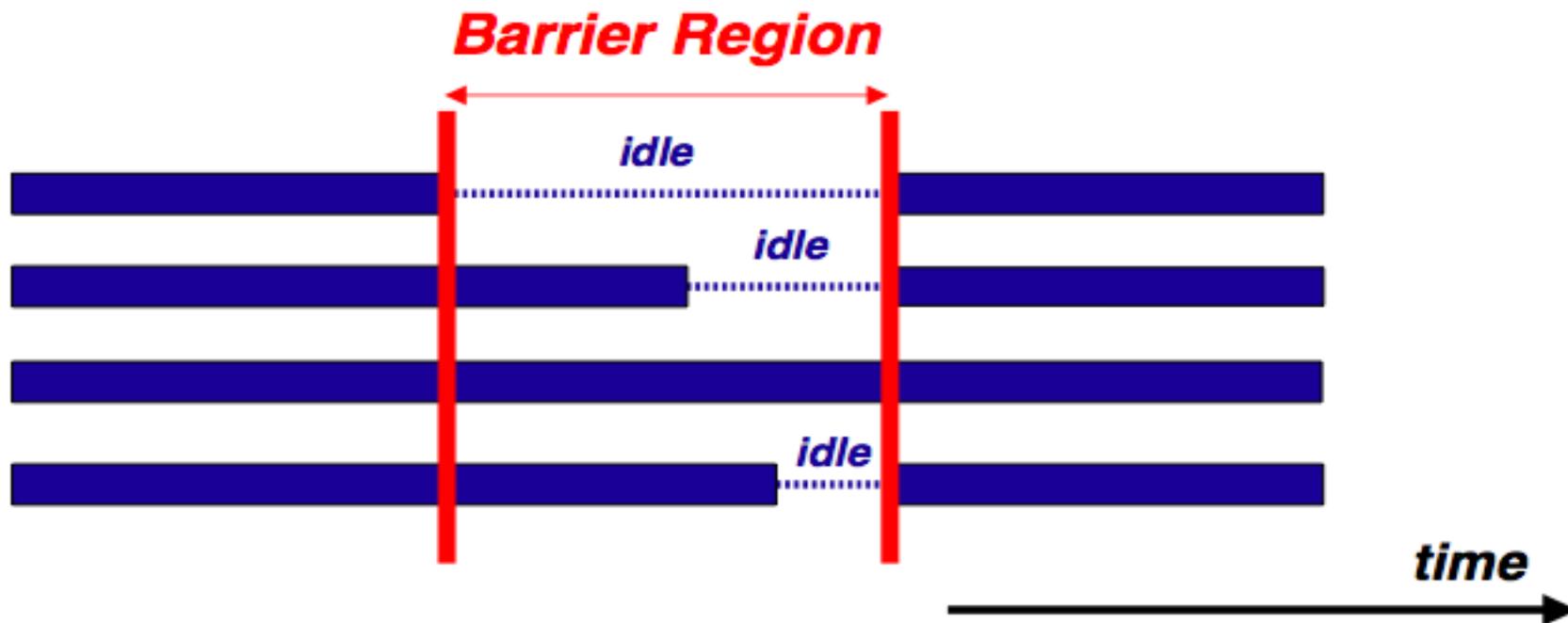
wait !

barrier

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

Barreira /4

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



Barreira /5

- Barreira
 - Sintaxe

```
#pragma omp barrier
```

```
!$omp barrier
```

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")

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]
```

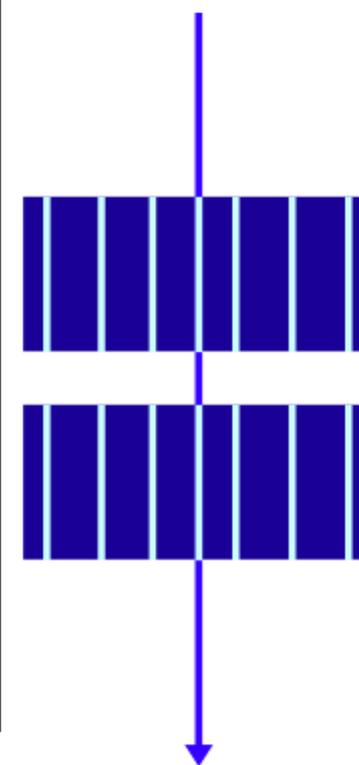
Construtor for/do

- 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]
```

Constructor Sections

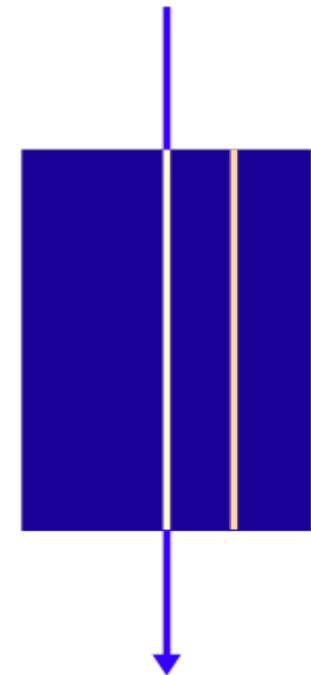
- Constructor “Sections”
 - Cláusulas suportadas
 - private
 - firstprivate
 - lastprivate
 - reduction
 - nowait

Construtor Sections

- Construtor “Sections”
 - Exemplo

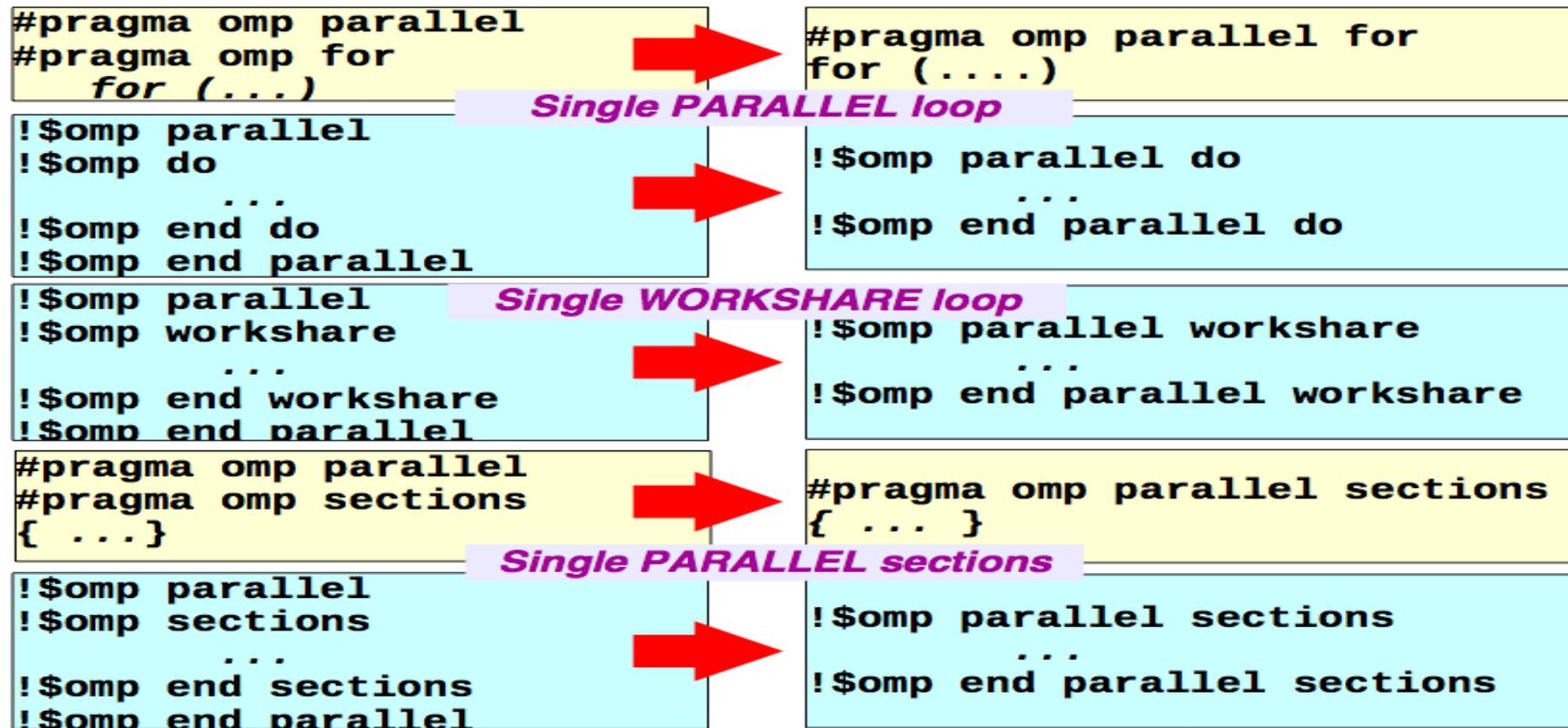
```
#pragma omp parallel default(none)\
    shared(n,a,b,c,d) private(i)
{
    #pragma omp sections nowait
    {
        #pragma omp section
        for (i=0; i<n-1; i++)
            b[i] = (a[i] + a[i+1])/2;

        #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

```
Original Code  
.....  
"read a[0..N-1]";  
.....
```

```
"declare A to be shared"  
#pragma omp parallel  
{  
.....  
..... one volunteer requested .....  
"read a[0..N-1]";  
..... thanks, we're done .....  
.....  
}  
  
Parallel Version
```

May have to insert a barrier here

Construtor Single

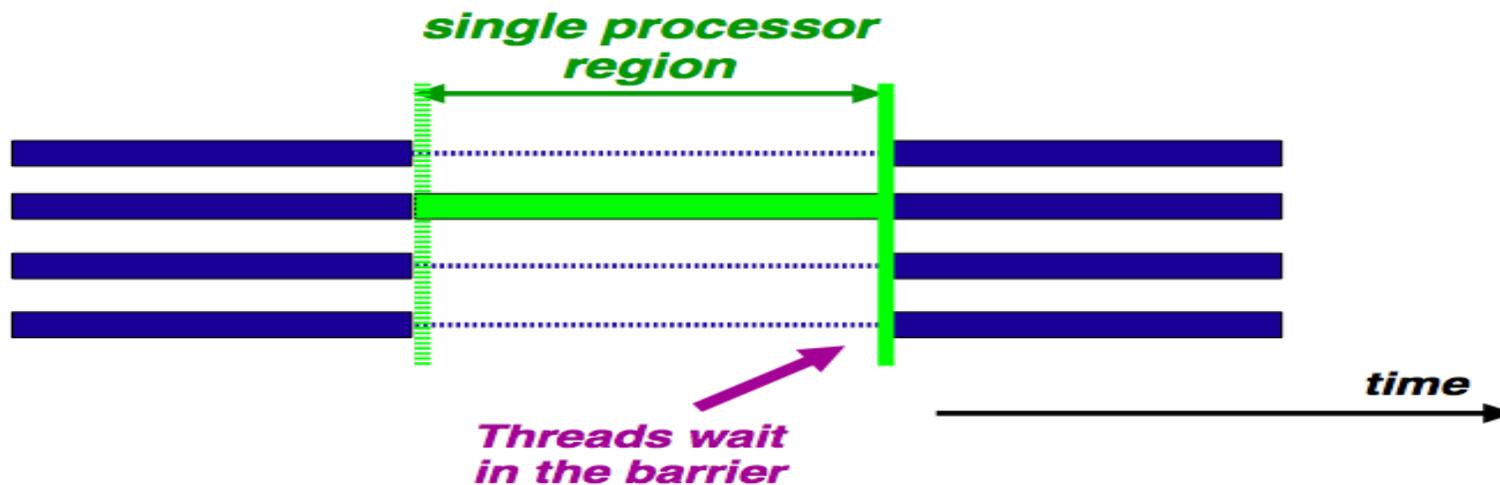
- Construtor “Single”
 - Somente uma thread (qualquer) executa o bloco

```
#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
 - F_m : fração de processamento da parte otimizada
 - Sempre menor que 1
 - G_e : ganho de desempenho com a otimização (pode ser paralelização)
 - Sempre maior que 1

$$ganho = \frac{1}{(1 - F_m) + \frac{F_m}{G_e}}$$

Construtor Single

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

$$ganho = \frac{1}{(1 - F_m) + \frac{F_m}{G_e}}$$

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++){
```

```
.....
```

```
sum += a[i];
```

```
.....
```

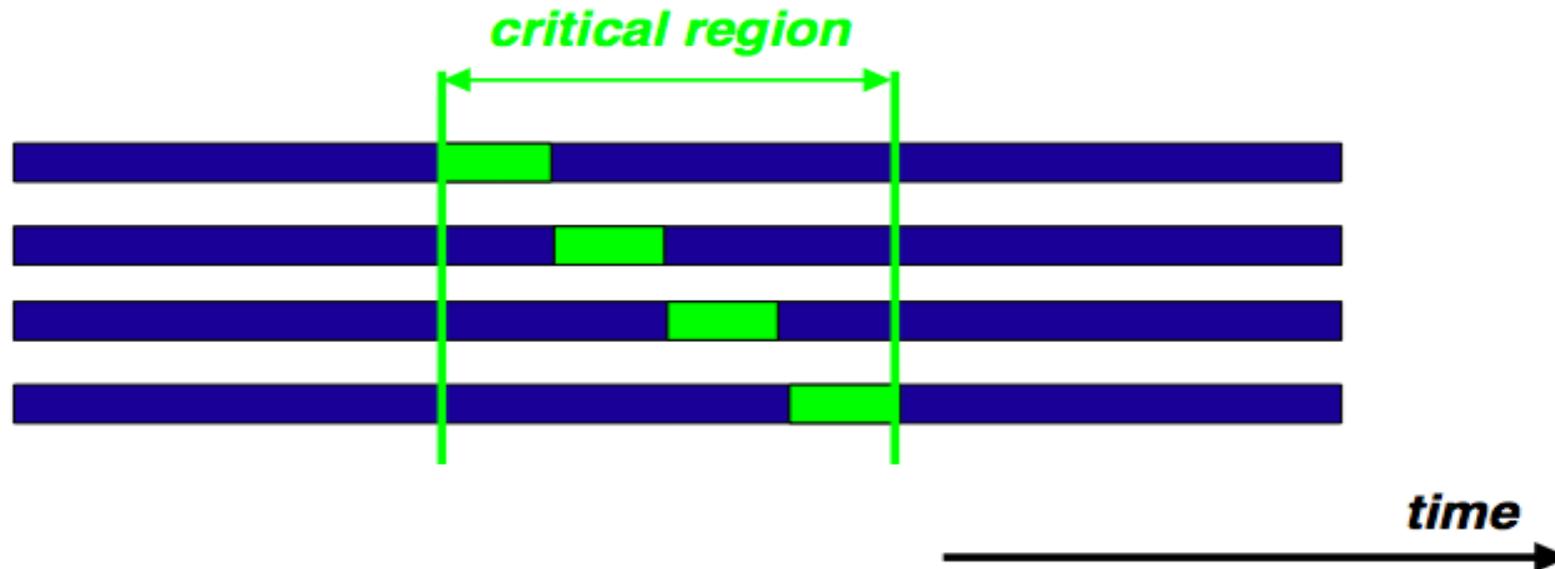
```
}
```

one at a time can proceed

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)**



ESTUDO DE CASO: REDE NEURAL



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

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

What is the problem ?

```

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;
      ## 116.290 96.3 121.930 1047.     next_p = next_p->next) {
      1048.     h[y1][x1] += next_p->strength *
              V1[next_p->y][next_p->x].r;
      1049.
      1052.   }
      1053. }
```

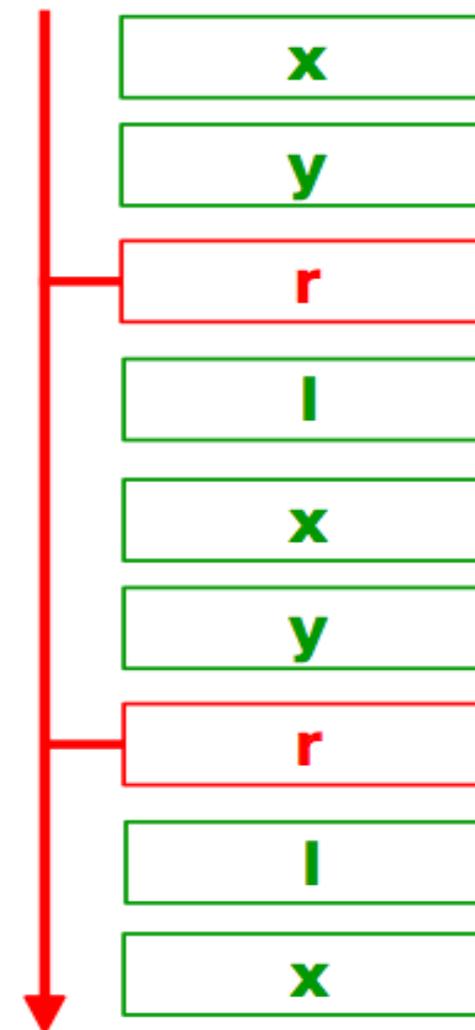
96% of the time spent in this single statement

Estudo de Caso Rede Neural

- Estudo de caso: melhora de desempenho com OpenMP
 - Problema de estrutura de dados
 - Se usa somente $\frac{1}{4}$ 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: 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;

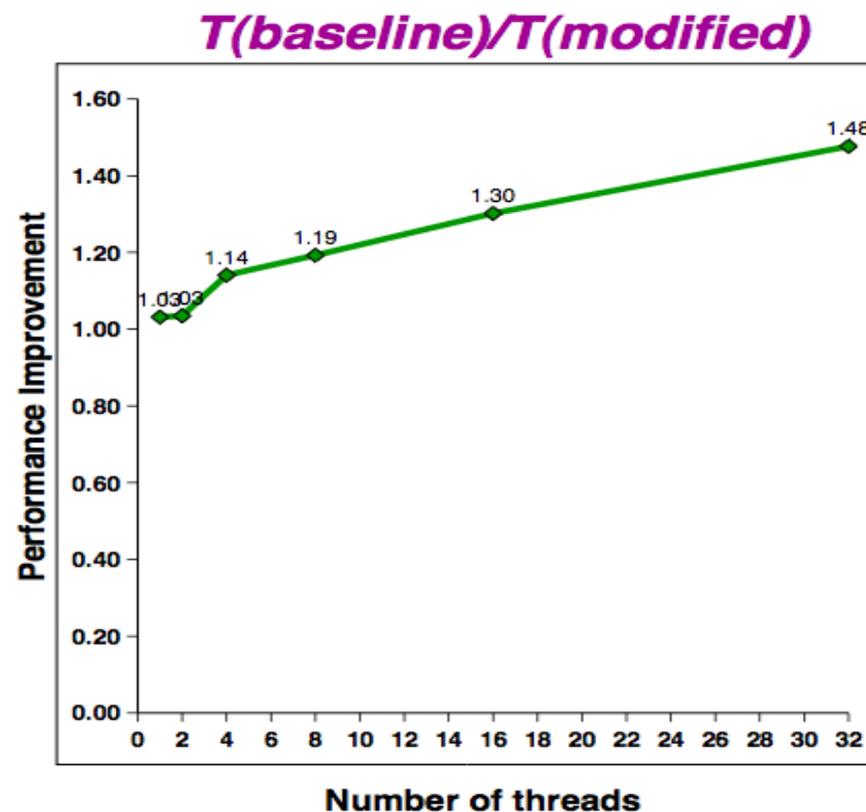
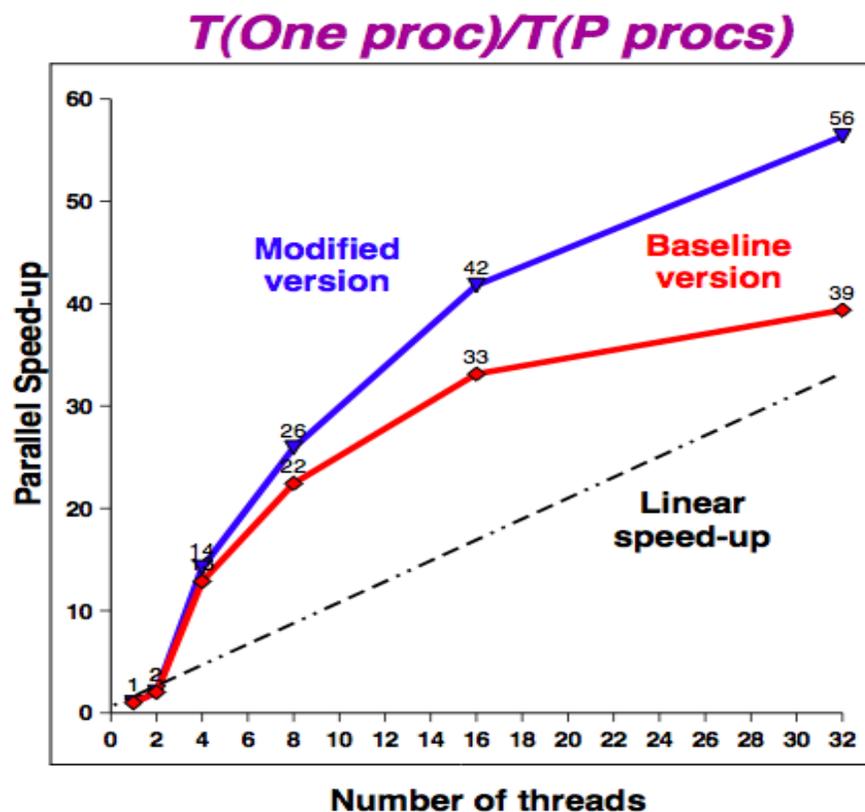
            <statements deleted>
        }
    }

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

*Can be executed
in parallel*

Estudo de Caso Rede Neural

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

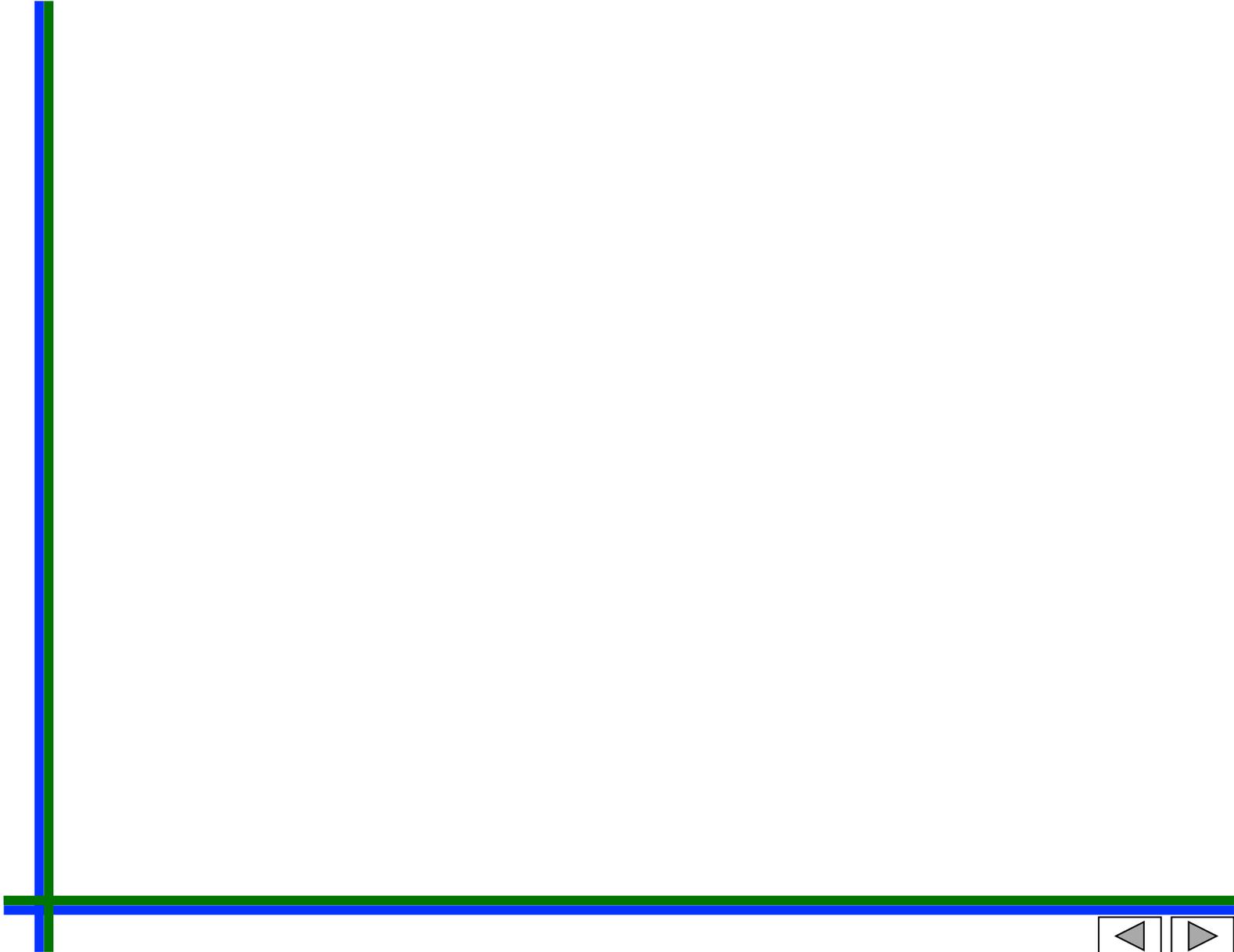


Note:
Single processor run time is 5001 seconds for the baseline version (4847 for the modified version)

Instituto de Informática

OpenMP: Uma Introdução

Cláudio Geyer



Exercícios

- Exercícios:
 - A)

Revisão

- Revisão

Referências