# Getting OpenMP Up To Speed

**Ruud van der Pas**

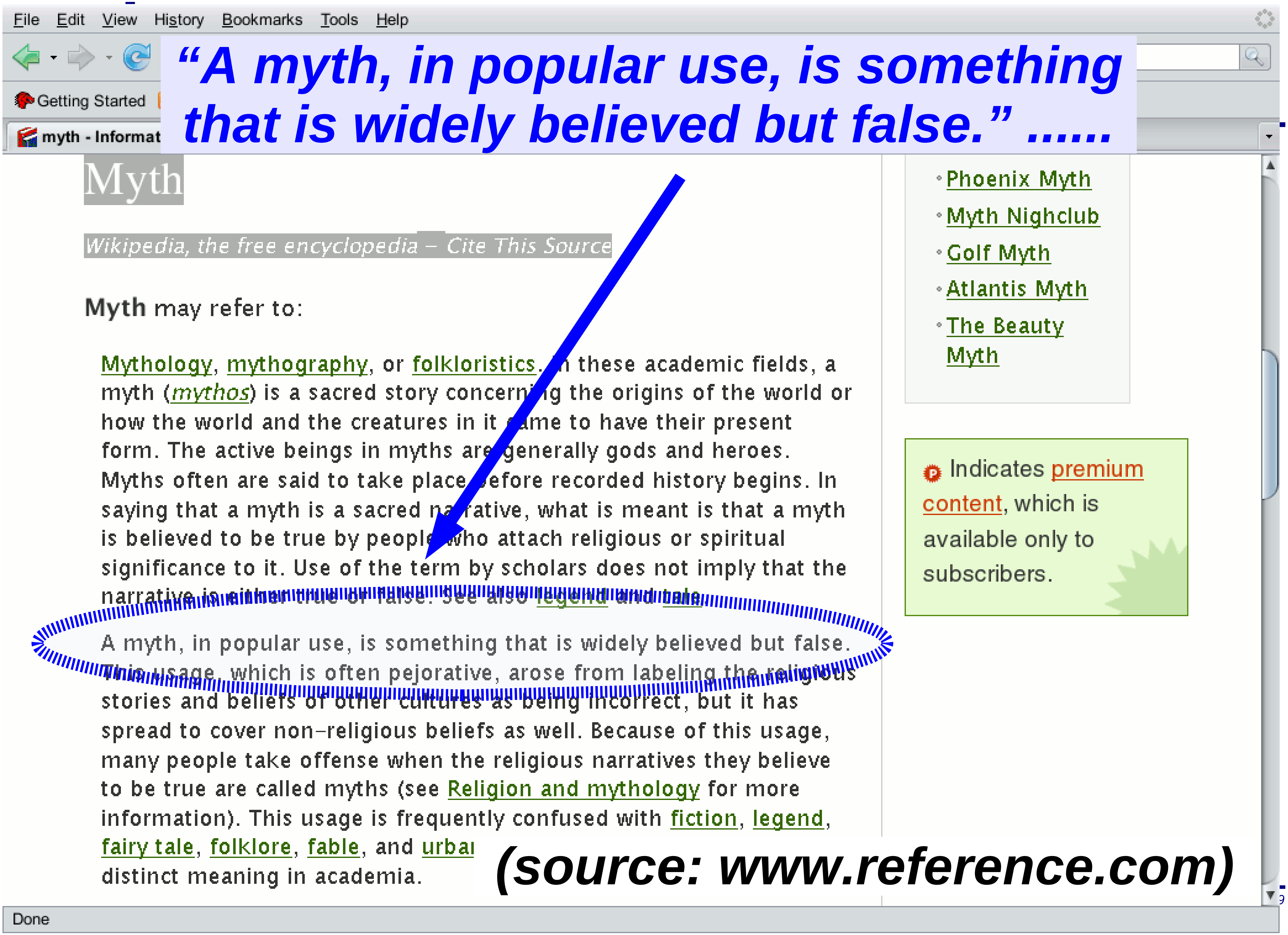
**Senior Staff Engineer
Sun Microsystems
Menlo Park, CA, USA**

**IWOMP 2009
TU Dresden
Dresden, Germany
June 3-5, 2009**

# Outline

- ❑ ***The Myth***
- ❑ ***Deep Trouble***
- ❑ ***Get Real***
- ❑ ***The Wrapping***

**“A myth, in popular use, is something that is widely believed but false.” ……**



# Myth

Wikipedia, the free encyclopedia – Cite This Source

**Myth** may refer to:

Mythology, mythography, or folkloristics. In these academic fields, a myth (*mythos*) is a sacred story concerning the origins of the world or how the world and the creatures in it came to have their present form. The active beings in myths are generally gods and heroes. Myths often are said to take place before recorded history begins. In saying that a myth is a sacred narrative, what is meant is that a myth is believed to be true by people who attach religious or spiritual significance to it. Use of the term by scholars does not imply that the narrative is either true or false. See also legend and tale.

A myth, in popular use, is something that is widely believed but false. This usage, which is often pejorative, arose from labeling the religious stories and beliefs of other cultures as being incorrect, but it has spread to cover non-religious beliefs as well. Because of this usage, many people take offense when the religious narratives they believe to be true are called myths (see Religion and mythology for more information). This usage is frequently confused with fiction, legend, fairy tale, folklore, fable, and urba
distinct meaning in academia.

- Phoenix Myth
- Myth Nighclub
- Golf Myth
- Atlantis Myth
- The Beauty Myth

Indicates premium content, which is available only to subscribers.

**(source: www.reference.com)**

# The Myth

## “OpenMP Does Not Scale”

# Hmmm .... What Does That Really Mean ?

# Some Questions I Could Ask

***"Do you mean you wrote a parallel program, using OpenMP and it doesn't perform?"***

***"I see. Did you make sure the program was fairly well optimized in sequential mode?"***

***"Oh. You didn't. By the way, why do you expect the program to scale?"***

***"Oh. You just think it should and you used all the cores. Have you estimated the speed up using Amdahl's Law?"***

***"No, this law is not a new EU environmental regulation. It is something else."***

***"I understand. You can't know everything. Have you at least used a tool to identify the most time consuming parts in your program?"***

# Some More Questions I Could Ask

*“Oh. You didn't. You just parallelized all loops in the program. Did you try to avoid parallelizing innermost loops in a loop nest?”*

*“Oh. You didn't. Did you minimize the number of parallel regions then?”*

*“Oh. You didn't. It just worked fine the way it was.*

*“Did you at least use the nowait clause to minimize the use of barriers?”*

*“Oh. You've never heard of a barrier. Might be worth to read up on.”*

*“Do all processors roughly perform the same amount of work?”*

*“You don't know, but think it is okay. I hope you're right.”*

# I Don't Give Up That Easily

***“Did you make optimal use of private data, or did you share most of it?”***

***“Oh. You didn't. Sharing is just easier. I see.***

***“You seem to be using a cc-NUMA system. Did you take that into account?”***

***“You've never heard of that either. How unfortunate. Could there perhaps be any false sharing affecting performance?”***

***“Oh. Never heard of that either. May come handy to learn a little more about both.”***

***“So, what did you do next to address the performance ?”***

***“Switched to MPI. Does that perform any better then?”***

***“Oh. You don't know. You're still debugging the code.”***

# Going Into Pedantic Mode

***“While you're waiting for your MPI debug run to finish (are you sure it doesn't hang by the way), please allow me to talk a little more about OpenMP and Performance.”***

# Deep Trouble

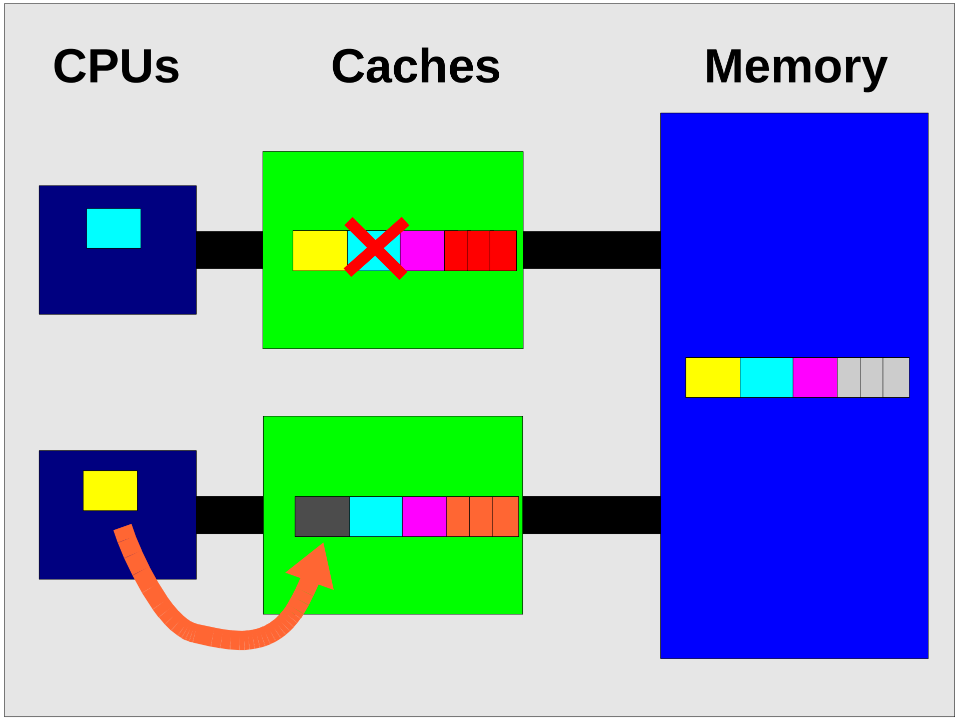
# OpenMP and Performance

- ***The transparency of OpenMP is a mixed blessing***
  - ***Makes things pretty easy***
  - ***May mask performance bottlenecks***
- ***In the ideal world, an OpenMP application just performs well***
- ***Unfortunately, this is not the case***
- ***Two of the more obscure effects that can negatively impact performance are cc-NUMA behavior and False Sharing***
- ***Neither of these are restricted to OpenMP, but they are important enough to cover in some detail here***

# *False Sharing*

# False Sharing

***A store into a shared cache line invalidates the other copies of that line:***



***The system is not able to distinguish between changes within one individual line***

# False Sharing Red Flags

- ***Be alert, when all of these three conditions are met:***
  - ***Shared data is modified by multiple processors***
  - ***Multiple threads operate on the same cache line(s)***
  - ***Update occurs simultaneously and very frequently***
- ***Use local data where possible***
- ***Shared read-only data does not lead to false sharing***

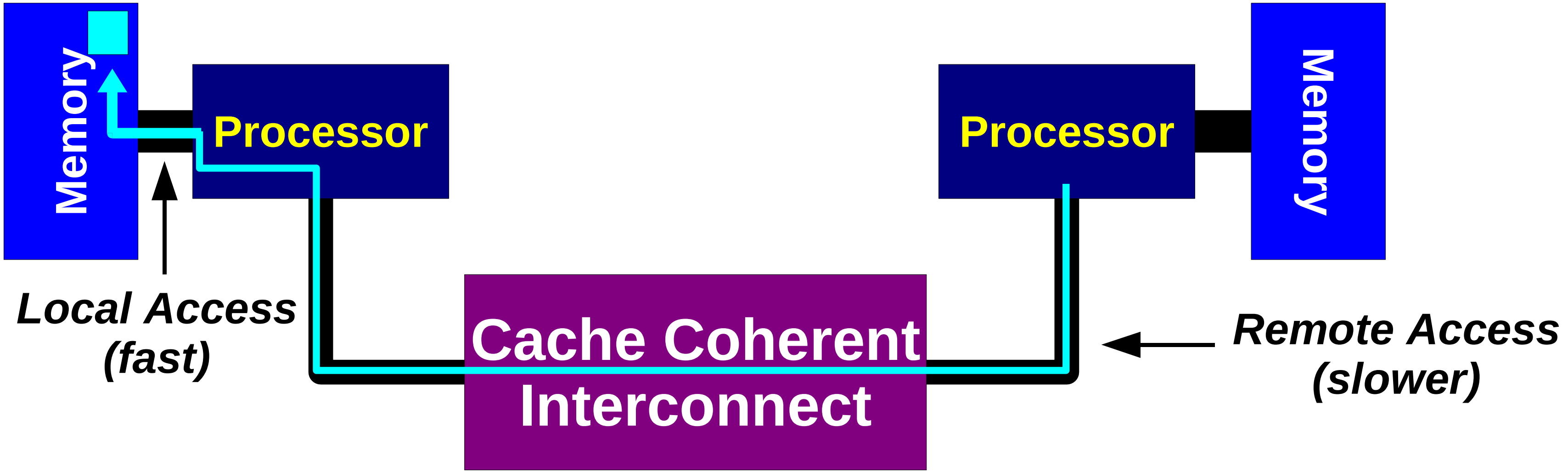
## Comments:

☞ ***In a Distributed Memory programming model, data is local by default and explicitly shared by exchanging messages/buffers***

☞ ***In a Shared Memory programming model, it is often the other way round: most data is shared by default and has to be made private explicitly***

# *Considerations for cc-NUMA*

# A generic cc-NUMA architecture

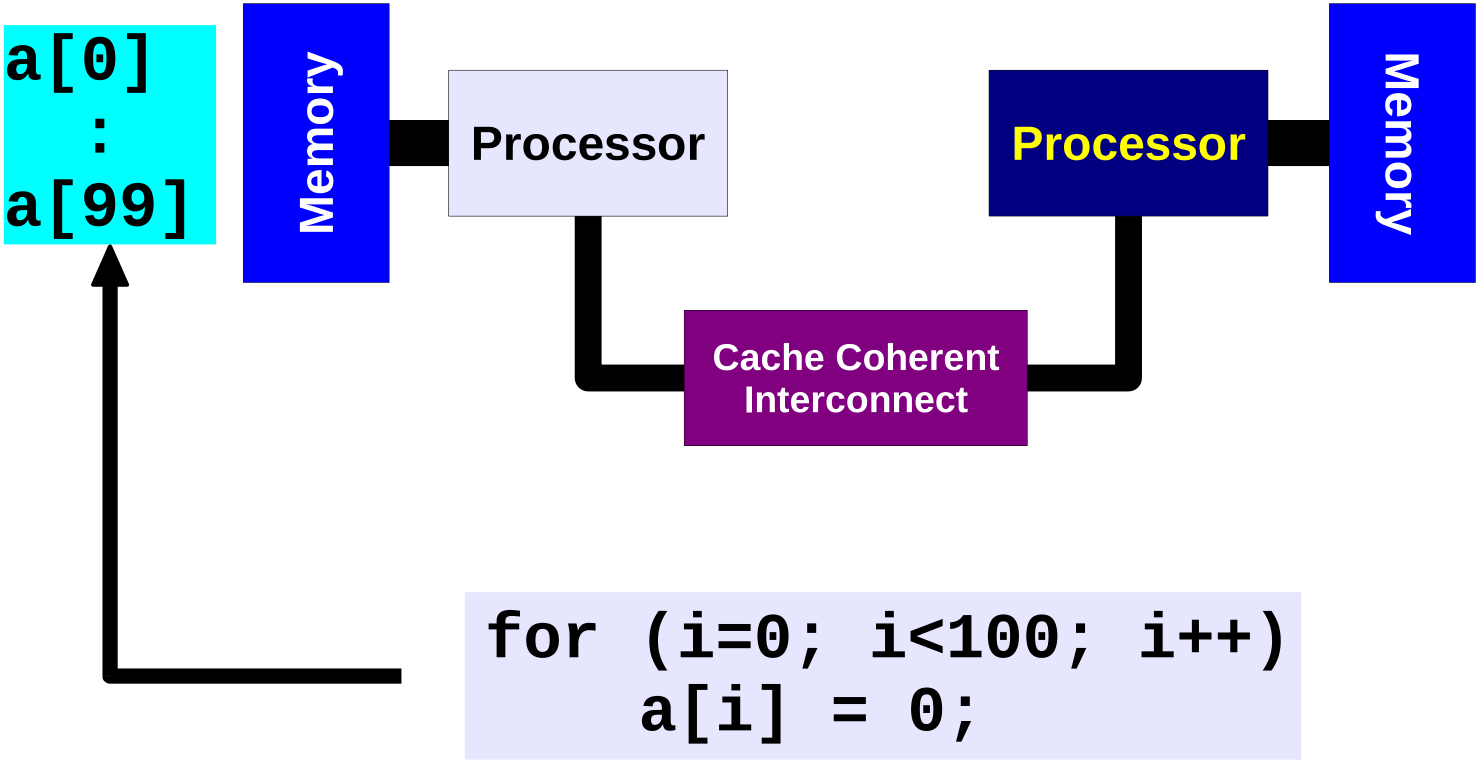


***Main Issue: How To Distribute The Data ?***

# About Data Distribution

- ❑ *Important aspect on a cc-NUMA system*
  - *If not optimal - longer access times, memory hotspots*
- ❑ *OpenMP does not provide support for cc-NUMA*
- ❑ *Placement comes from the Operating System*
  - *This is therefore Operating System dependent*
- ❑ *Solaris and Linux use “First Touch” to place data*
  - *Solaris: can use the “madvise” (3C) system call to control/change*
    - ✔ *Low level, but flexible, API to handle placement*

# About “First Touch” placement/1

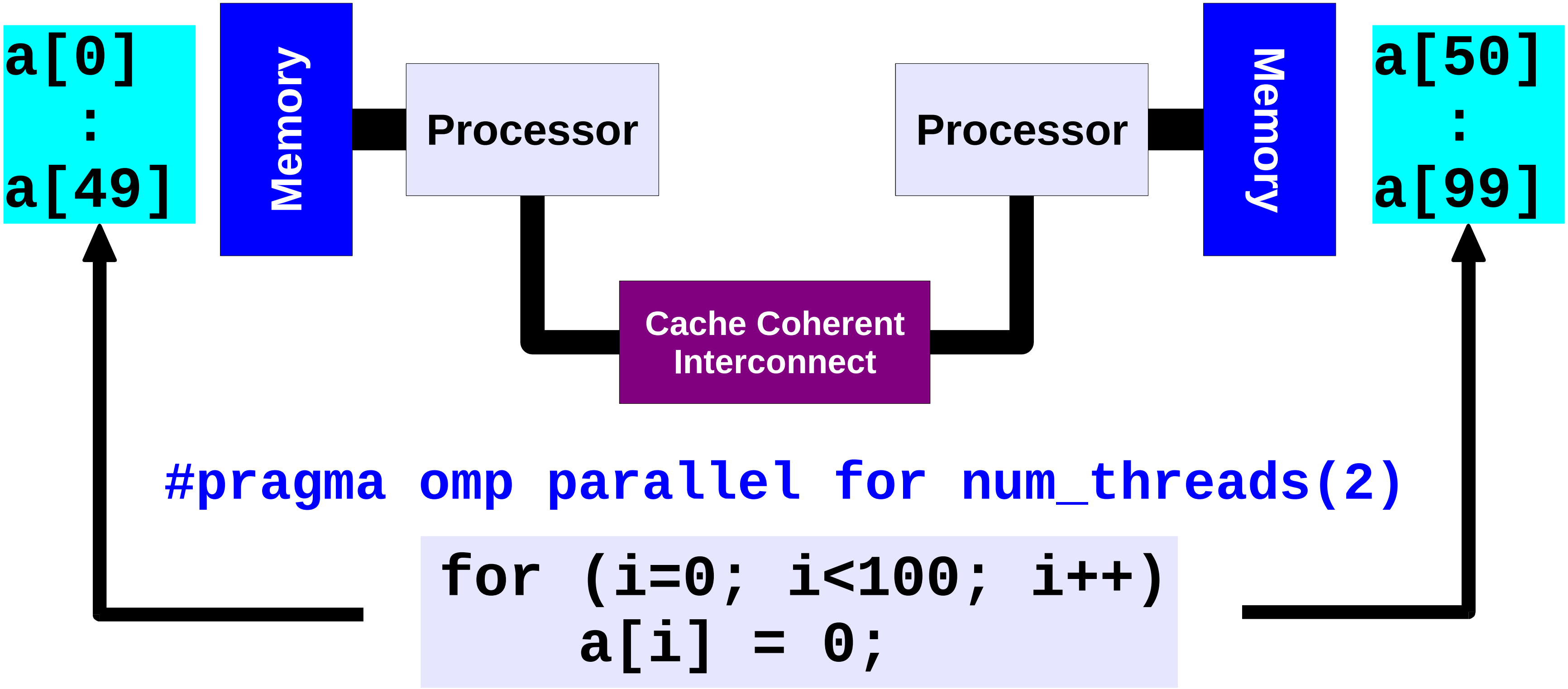


```
for (i=0; i<100; i++)
    a[i] = 0;
```

***First Touch***

***All array elements are in the memory of the processor executing this thread***

# About “First Touch” placement/2



```
#pragma omp parallel for num_threads(2)
 for (i=0; i<100; i++)
       a[i] = 0;
```

***First Touch***

***Both memories each have “their half” of the array***

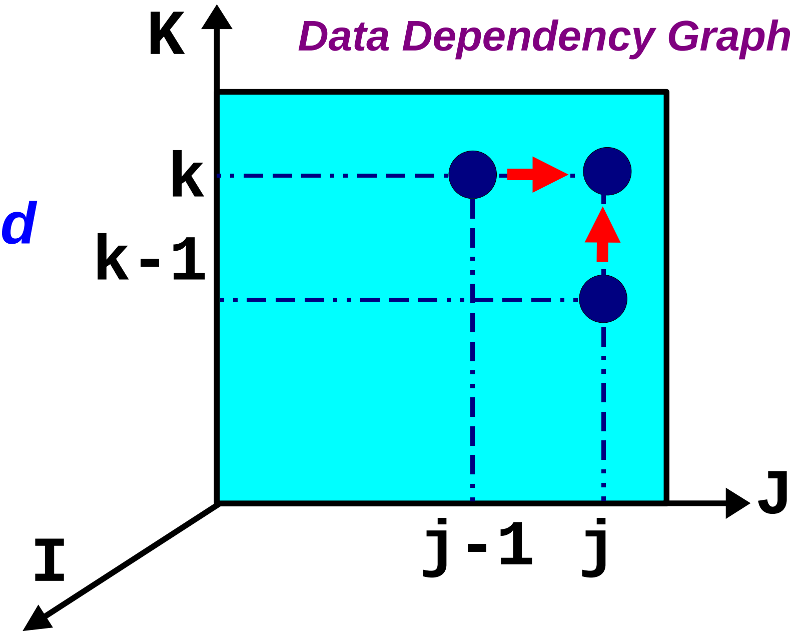
# Get Real

# *Block Matrix Update*

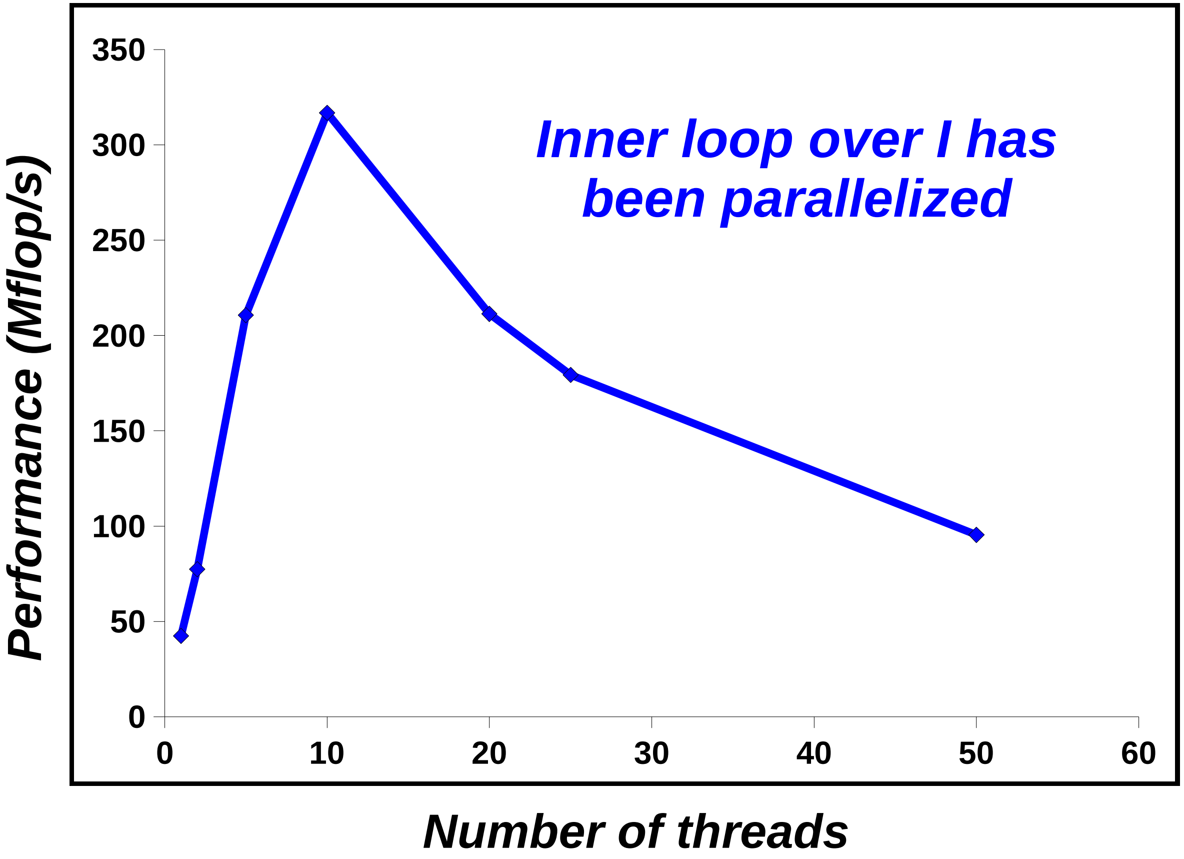
# A 3D matrix update

```
      do k = 2, n
         do j = 2, n
!$omp parallel do default(shared) private(i) &
!$omp schedule(static)
            do i = 1, m
               x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
            end do
!$omp end parallel do
         end do
      end do
```

- ❑ *The loops are correctly nested for serial performance*
- ❑ *Due to a data dependency on J and K, only the inner loop can be parallelized*
- ❑ *This will cause the barrier to be executed $(N-1)^2$ times*


*Data Dependency Graph*

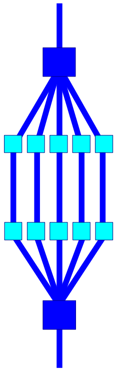
# The performance



**Scaling is very poor (as to be expected)**

| | | |
|---|---|---|
| *Dimensions* | : | *M=7,500 N=20* |
| *Footprint* | : | *~24 MByte* |

# Performance Analyzer data



*Using 10 threads*

```
Name                                                 Excl. User     Incl.     Excl.
                                                     CPU            User CPU  Wall
                                                       sec.      %    sec.      sec.
<Total>                                              10.590 100.0  10.590     1.550
__mt_EndOfTask_Barrier_                               5.740  54.2   5.740     0.240
__mt_WaitForWork_                                     3.860  36.4   3.860     0.
__mt_MasterFunction_                                  0.480   4.5   0.680     0.480
MAIN_                                                 0.230   2.2   1.200     0.470
block_3d_ -- MP doall from line 14 [_$d1A14      3d_]  0.170   1.6   5.910     0.170
block_3d_                                             0.040   0.4   6.460     0.040
memset                                                0.030   0.3   0.030     0.080
```

*Using 20 threads*

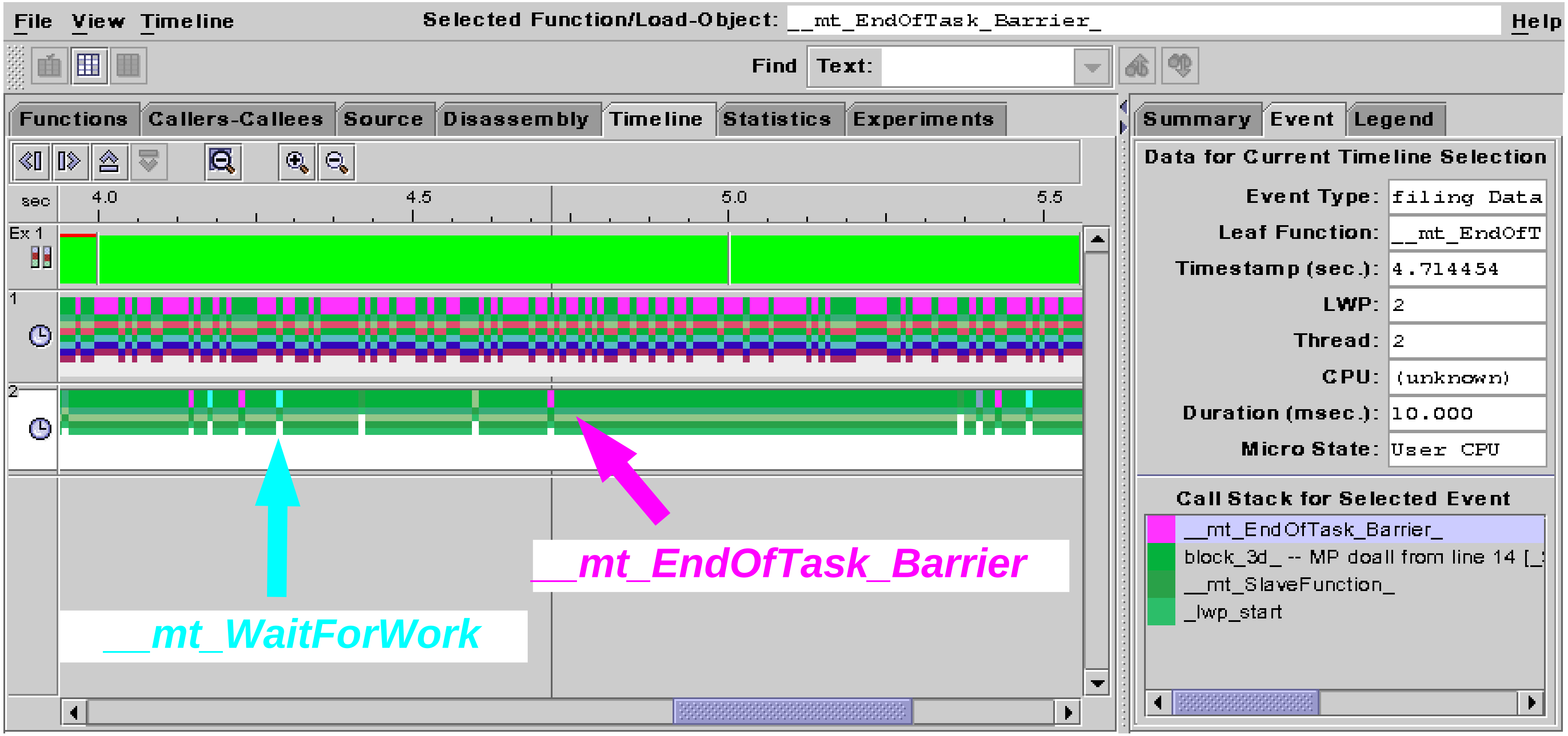
```
Name                                                 Excl. User     Incl.     Excl.
                                                      CPU            User CPU  Wall
                                                       sec.      %    sec.      sec.
<Total>                                              47.120 100.0  47.120     2.900
__mt_EndOfTask_Barrier_                              25.700  54.5  25.700     0.980
__mt_WaitForWork_                                    19.880  42.2  19.880     0.
__mt_MasterFunction_                                  1.100   2.3   1.320     1.100
MAIN_                                                 0.190   0.4   2.520     0.470
block_3d_ -- MP doall from line 14 [_$d1A14.block_3d_] 0.100   0.2  25.800     0.100
__mt_setup_doJob_int_                                 0.080   0.2   0.080     0.080
__mt_setup_job_                                       0.020   0.0   0.020     0.020
block_3d_                                             0.010   0.0  27.020     0.010
```

*do not scale at all*

*scales somewhat*

***Question: Why is __mt_WaitForWork so high in the profile ?***

# The Analyzer Timeline overview



Selected Function/Load-Object: __mt_EndOfTask_Barrier_

**Data for Current Timeline Selection**

| | |
|---|---|
| Event Type: | filing Data |
| Leaf Function: | __mt_EndOfT |
| Timestamp (sec.): | 4.714454 |
| LWP: | 2 |
| Thread: | 2 |
| CPU: | (unknown) |
| Duration (msec.): | 10.000 |
| Micro State: | User CPU |

**Call Stack for Selected Event**

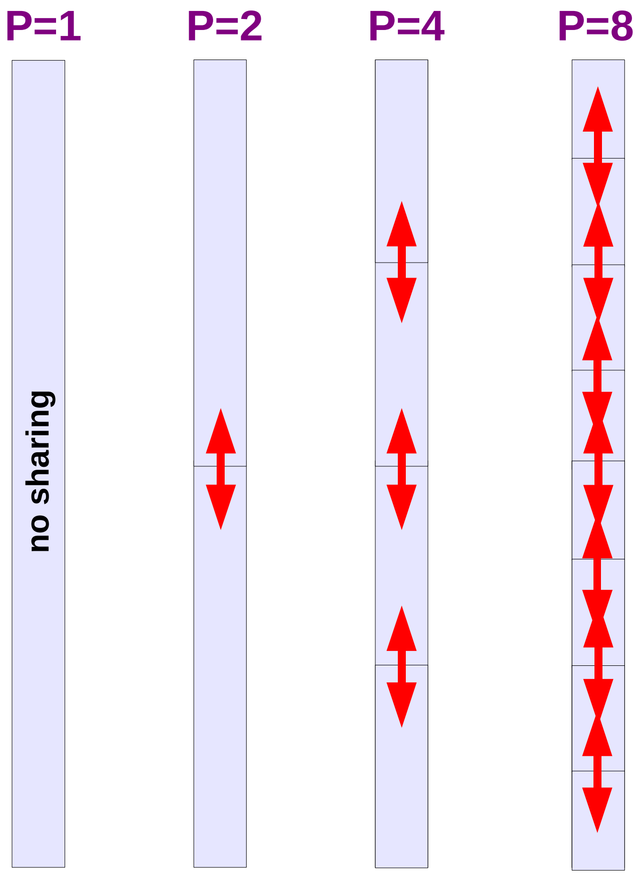
- __mt_EndOfTask_Barrier_
- block_3d_ -- MP doall from line 14 [_
- __mt_SlaveFunction_
- _lwp_start

*__mt_EndOfTask_Barrier*

*__mt_WaitForWork*

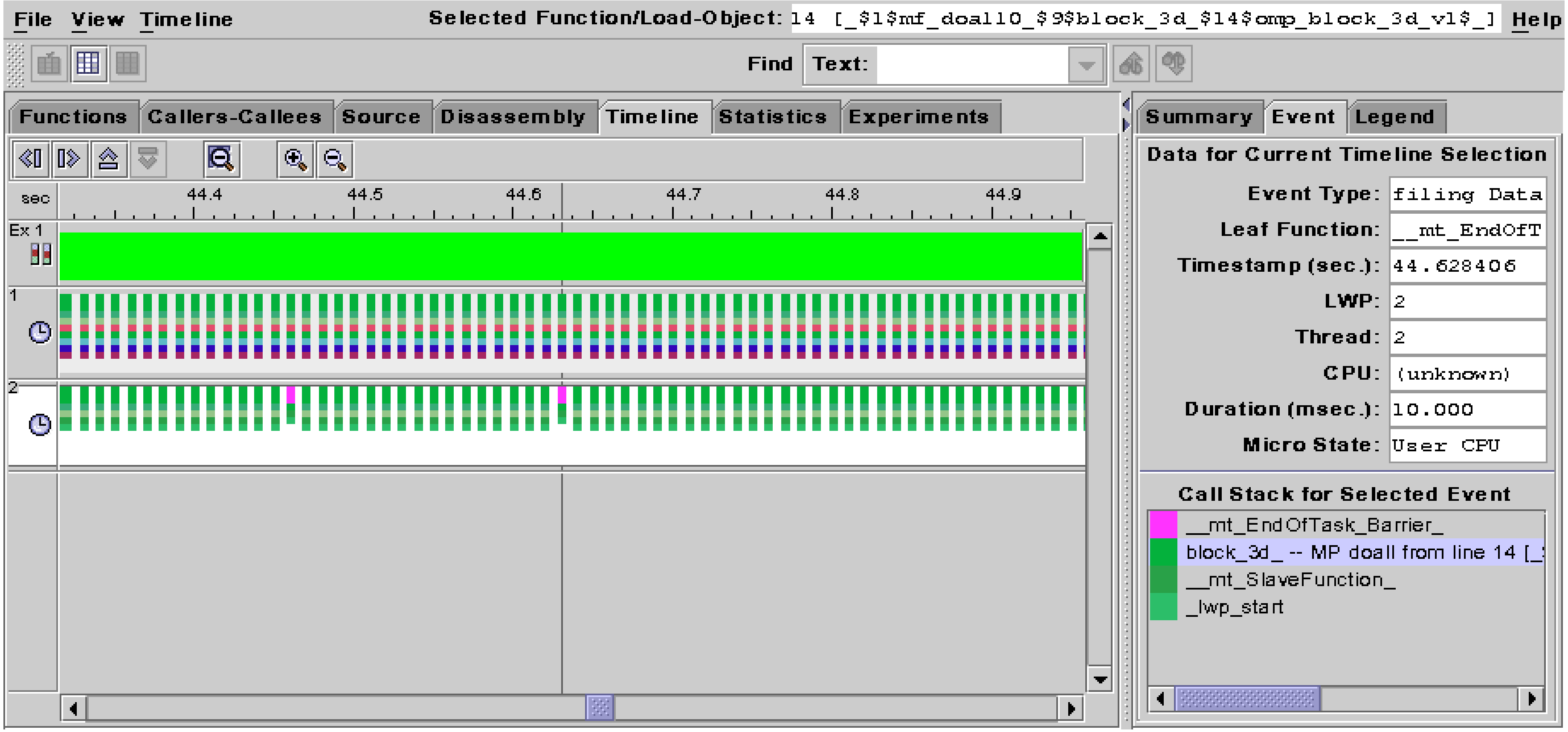
# This is False Sharing at work !

```
!$omp parallel do default(shared) private(i) &
!$omp schedule(static)
           do i = 1, m
              x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
           end do
!$omp end parallel do
```



*False sharing increases as we increase the number of threads*

# Sanity Check: Setting M=75000*



Selected Function/Load-Object: 14 [_$1$mf_doall0_$9$block_3d_$14$omp_block_3d_v1$_]

Data for Current Timeline Selection

Event Type: filing Data

Leaf Function: __mt_EndOfT

Timestamp (sec.): 44.628406

LWP: 2

Thread: 2

CPU: (unknown)

Duration (msec.): 10.000

Micro State: User CPU

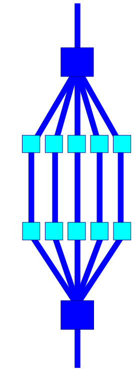
Call Stack for Selected Event

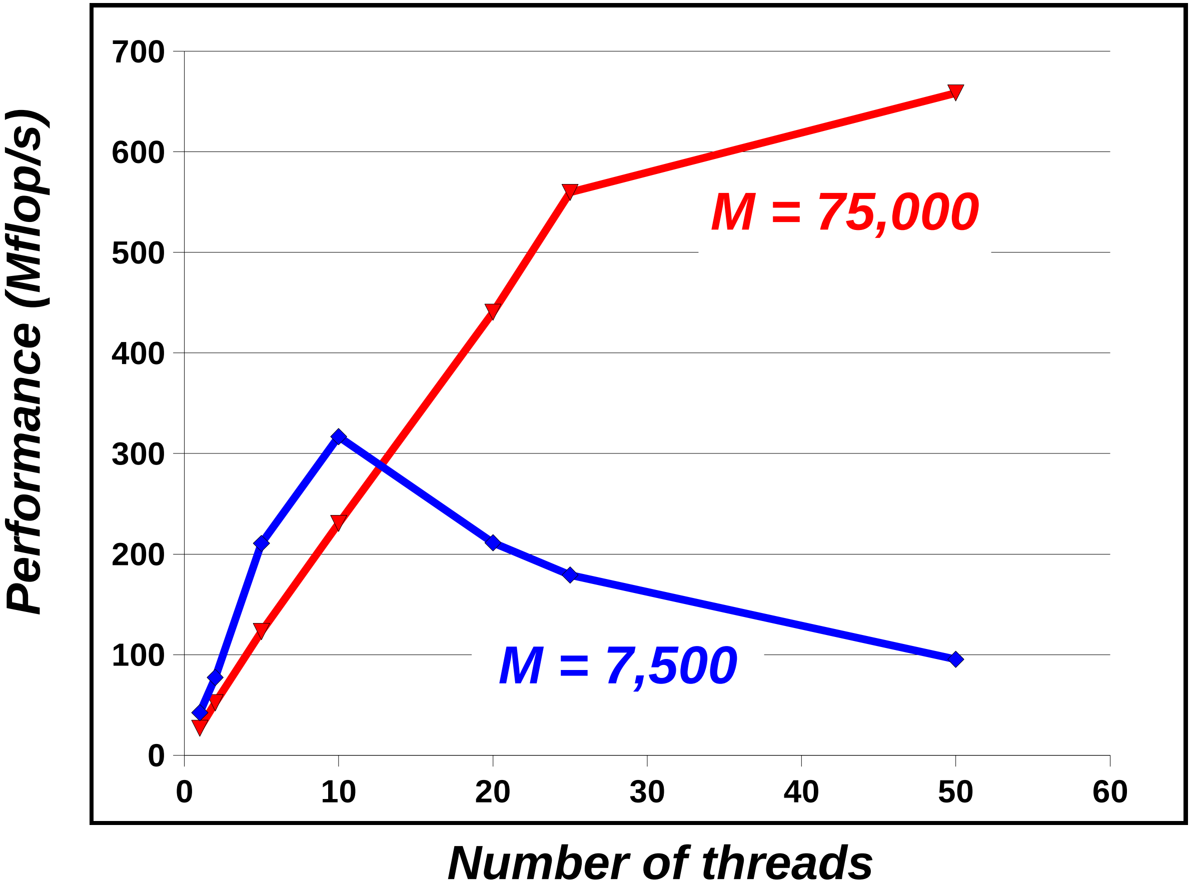
- __mt_EndOfTask_Barrier_
- block_3d_ -- MP doall from line 14 [_
- __mt_SlaveFunction_
- _lwp_start

## *Only a very few barrier calls now*

**) Increasing the length of the loop should decrease false sharing*

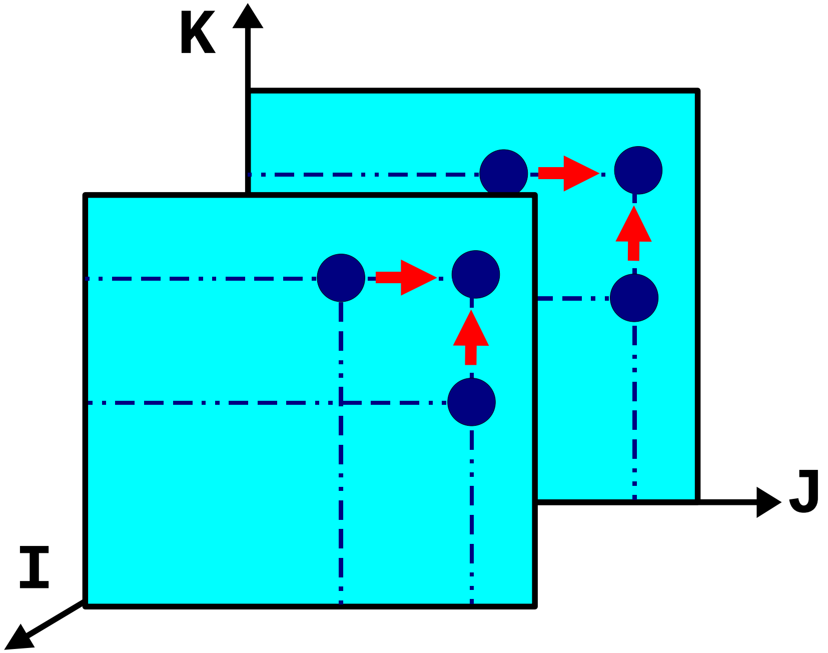
# Performance comparison





**For a higher value of M, the program scales better**

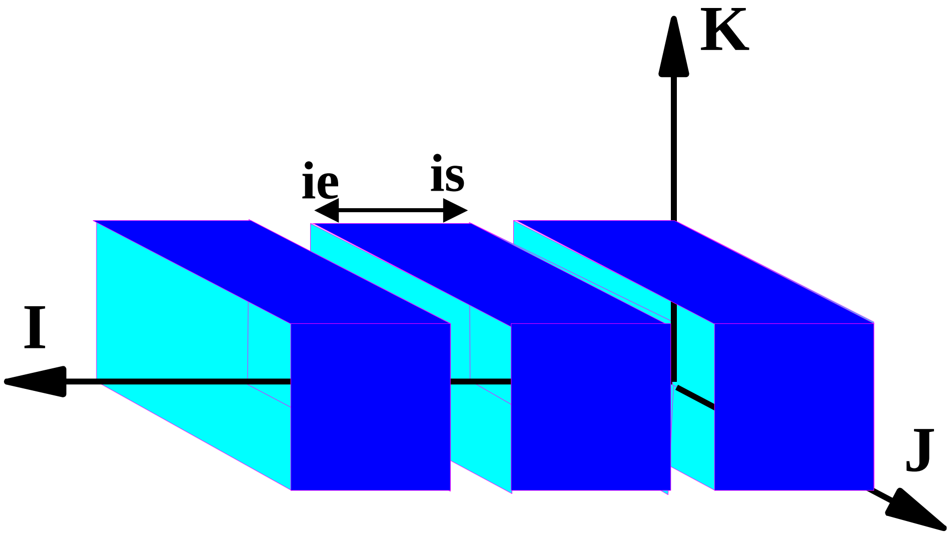
# Observation



- ❑ ***No data dependency on 'I'***
- ❑ ***Therefore we can split the 3D matrix in larger blocks and process these in parallel***

```
do k = 2, n
   do j = 2, n
      do i = 1, m
         x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
      end do
   end do
end do
```

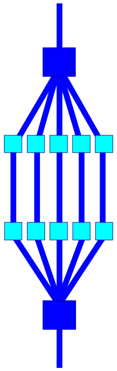
# The Idea



- ❑ *We need to distribute the M iterations over the number of processors*
- ❑ *We do this by controlling the start (IS) and end (IE) value of the inner loop*
- ❑ *Each thread will calculate these values for it's portion of the work*

```
do k = 2, n
   do j = 2, n
      do i = is, ie
         x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
      end do
   end do
end do
```

# The first implementation



```
subroutine kernel(is,ie,m,n,x,scale)
         .......
do k = 2, n
 do j = 2, n
    do i = is, ie
      x(i,j,k)=x(i,j,k-1)+x(i,j-1,k)*scale
    end do
  end do
end do
```

```
     use omp_lib
         ........
     nrem   = mod(m,nthreads)
     nchunk = (m-nrem)/nthreads

!$omp parallel default (none)&
!$omp private (P,is,ie)      &
!$omp shared  (nrem,nchunk,m,n,x,scale)

      P = omp_get_thread_num()

      if ( P < nrem ) then
        is = 1 + P*(nchunk + 1)
        ie = is + nchunk
      else
        is = 1 + P*nchunk+ nrem
        ie = is + nchunk - 1
      end if

      call kernel(is,ie,m,n,x,scale)

!$omp end parallel
```

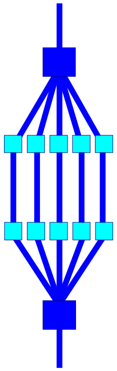
# Another Idea: Use OpenMP !

```
      use omp_lib

      implicit none
      integer     :: is, ie, m, n
      real(kind=8):: x(m,n,n), scale
      integer     :: i, j, k

!$omp parallel default(none) &
!$omp private(i,j,k) shared(m,n,scale,x)
      do k = 2, n
         do j = 2, n
!$omp do schedule(static)
            do i = 1, m
               x(i,j,k) = x(i,j,k-1) + x(i,j-1,k)*scale
            end do
!$omp end do nowait
         end do
      end do
!$omp end parallel
```

# How this works on 2 threads



Thread 0 Executes:

Thread 1 Executes:

```
k=2
j=2
```

parallel region

```
k=2
j=2
```

```
do i = 1,m/2
   x(i,2,2) = ...
end do
```

work sharing

```
do i = m/2+1,m
   x(i,2,2) = ...
end do
```

```
k=2
j=3
```

parallel region

```
k=2
j=3
```

```
do i = 1,m/2
   x(i,3,2) = ...
end do
```

work sharing

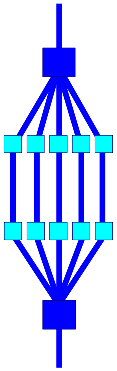
```
do i = m/2+1,m
   x(i,3,2) = ...
end do
```

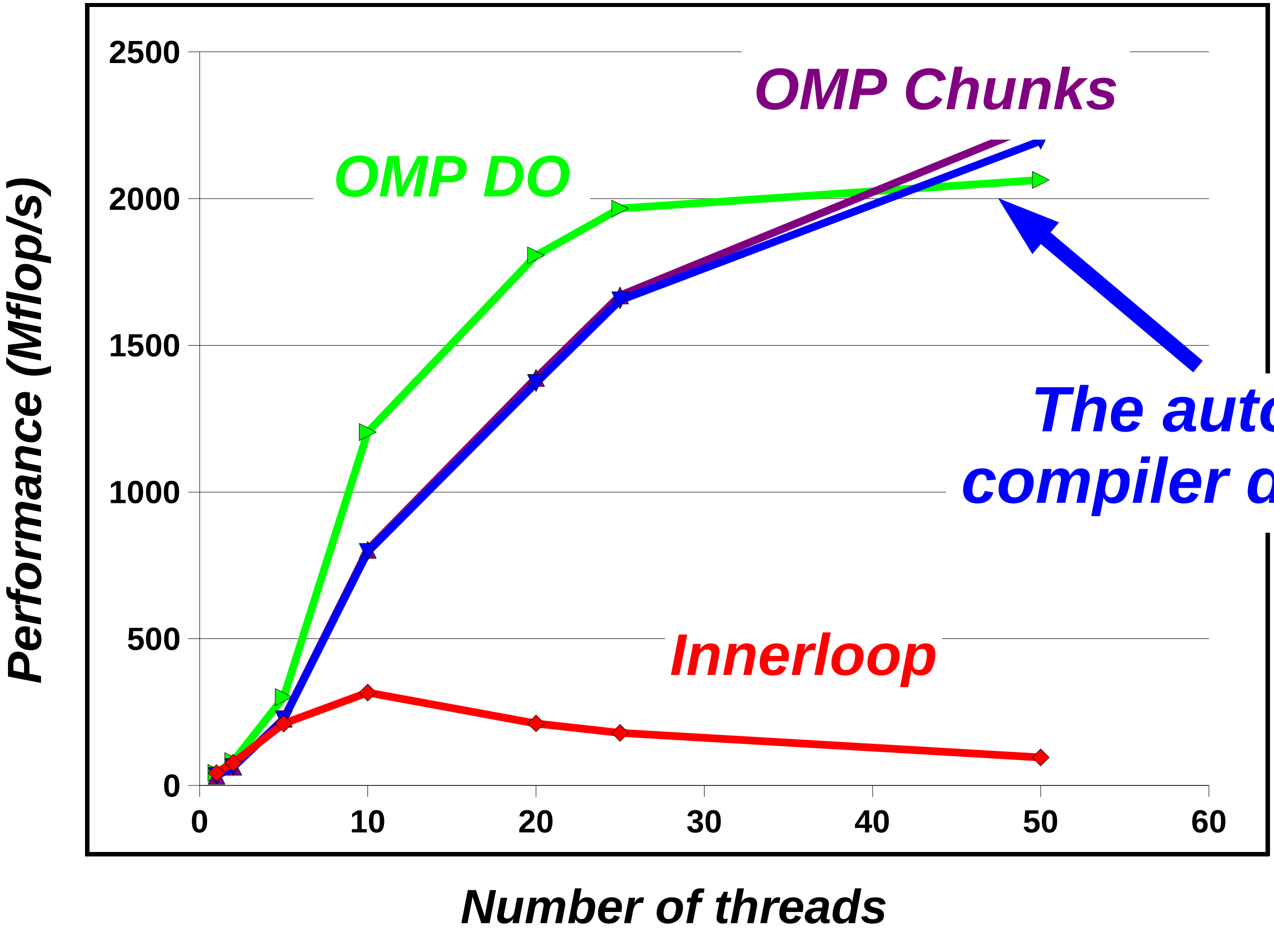
***This splits the operation in a way that is similar to our manual implementation***

# Performance

- ❑ ***We have set M=7500 N=20***
  - *This problem size does not scale at all when we explicitly parallelized the inner loop over 'I'*
- ❑ ***We have have tested 4 versions of this program***
  - *Inner Loop Over 'I' - Our first OpenMP version*
  - *AutoPar - The automatically parallelized version of 'kernel'*
  - *OMP_Chunks - The manually parallelized version with our explicit calculation of the chunks*
  - *OMP_DO - The version with the OpenMP parallel region and work-sharing DO*

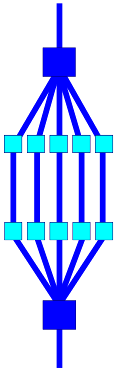
# The performance (M=7,500)





The auto-parallelizing compiler does really well !

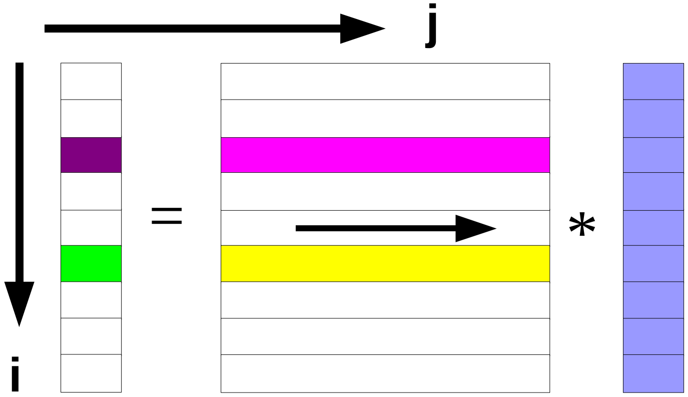
*Dimensions : M=7,500 N=20*
*Footprint : ~24 MByte*



# Matrix Times Vector

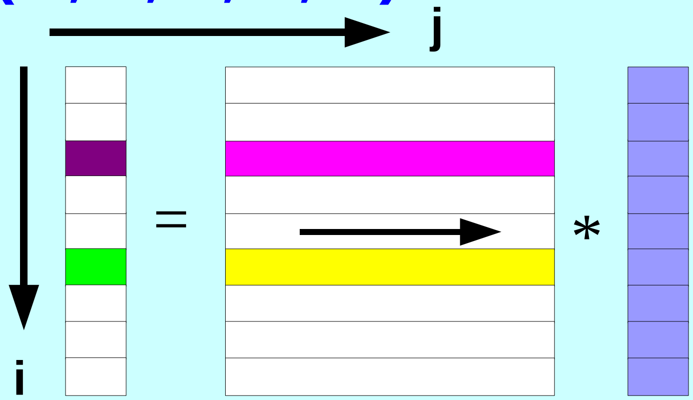
# The Sequential Source

```
for (i=0; i<m; i++)
{
  a[i] = 0.0;
  for (j=0; j<n; j++)
     a[i] += b[i][j]*c[j];
}
```

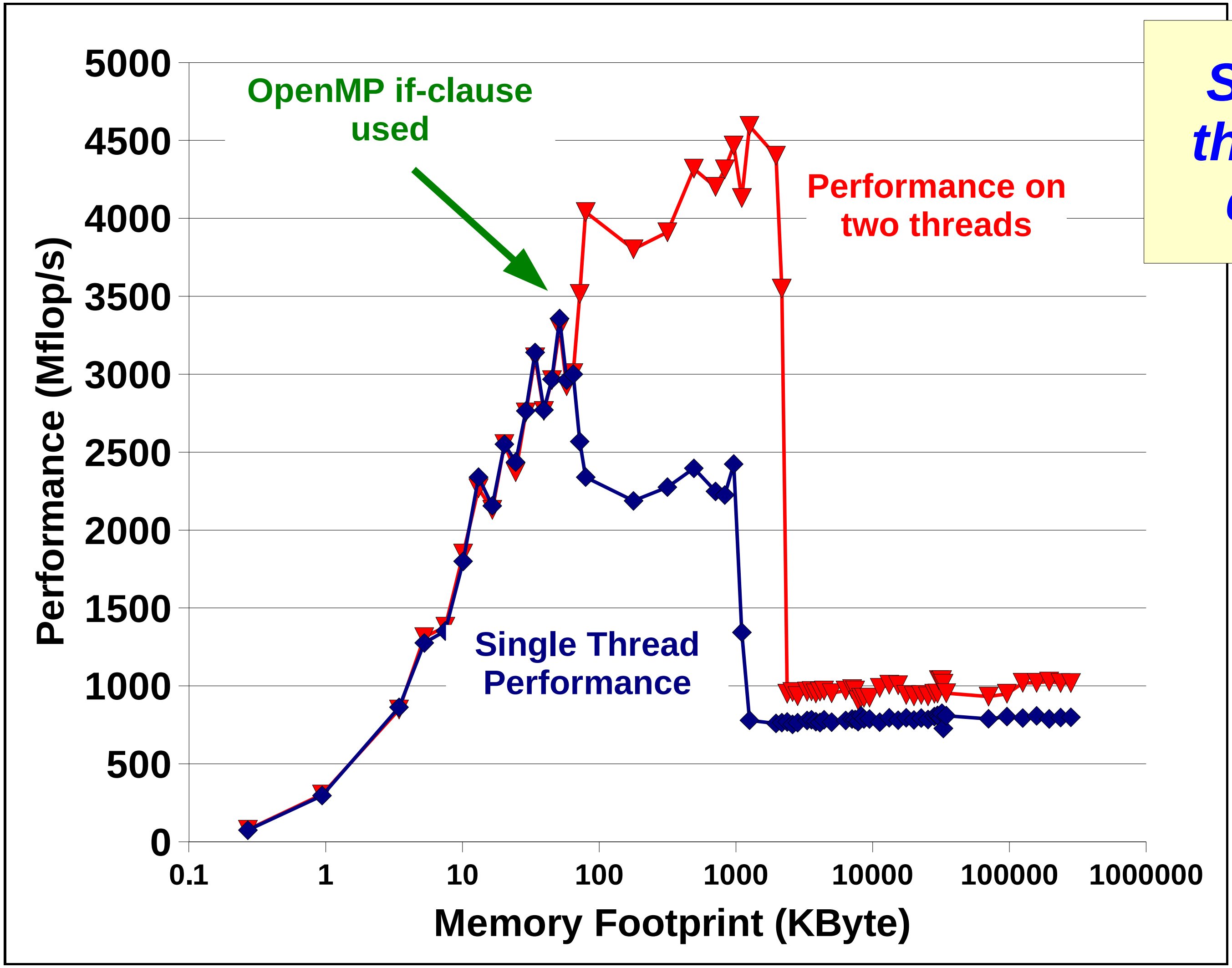


# The OpenMP Source

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



# Example - Matrix times vector

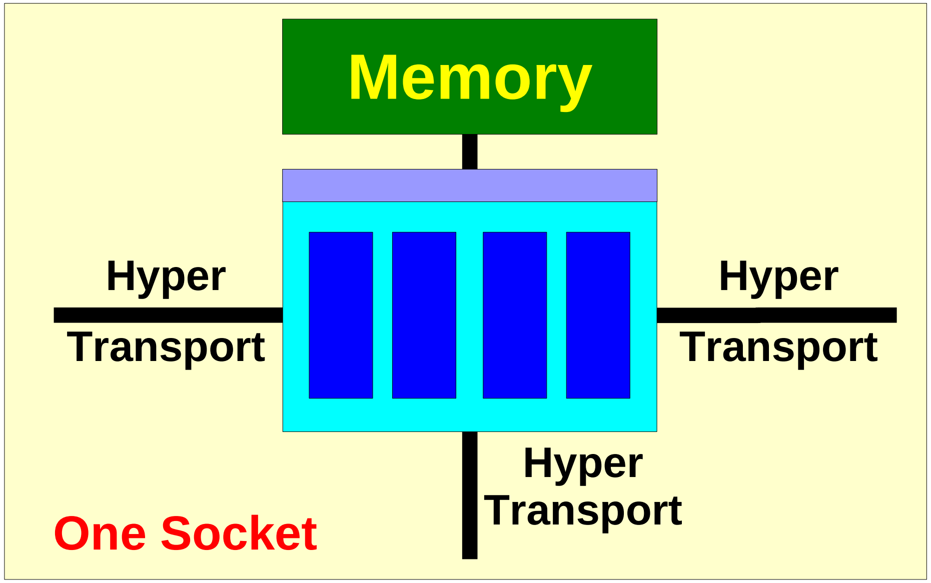


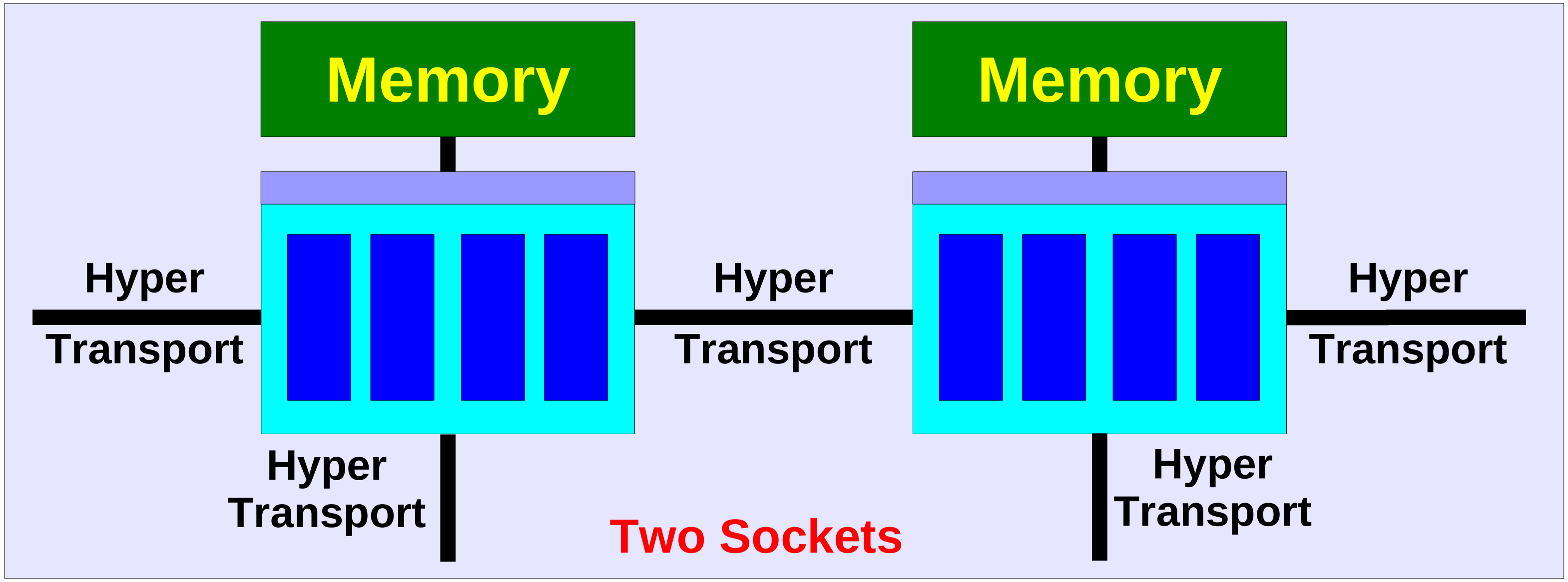
*Scalability on 2 threads is rather disappointing*

AMD Opteron Rev F @2600MHz

| | | |
|---|---|---|
| L1 cache | : | 64 KByte |
| Peak speed | : | 5200 Mflop/s |

# The Opteron Processor



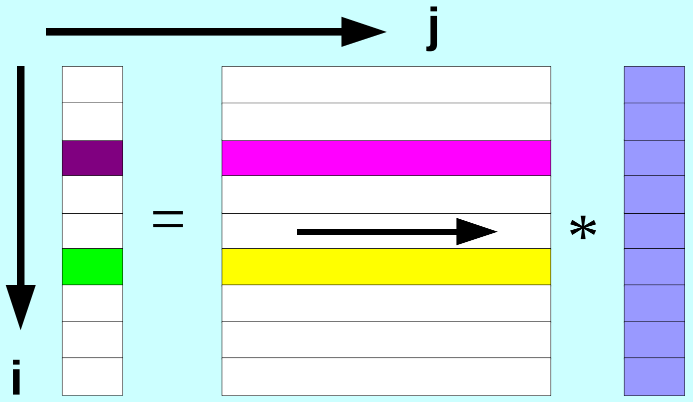


# Data Initialization

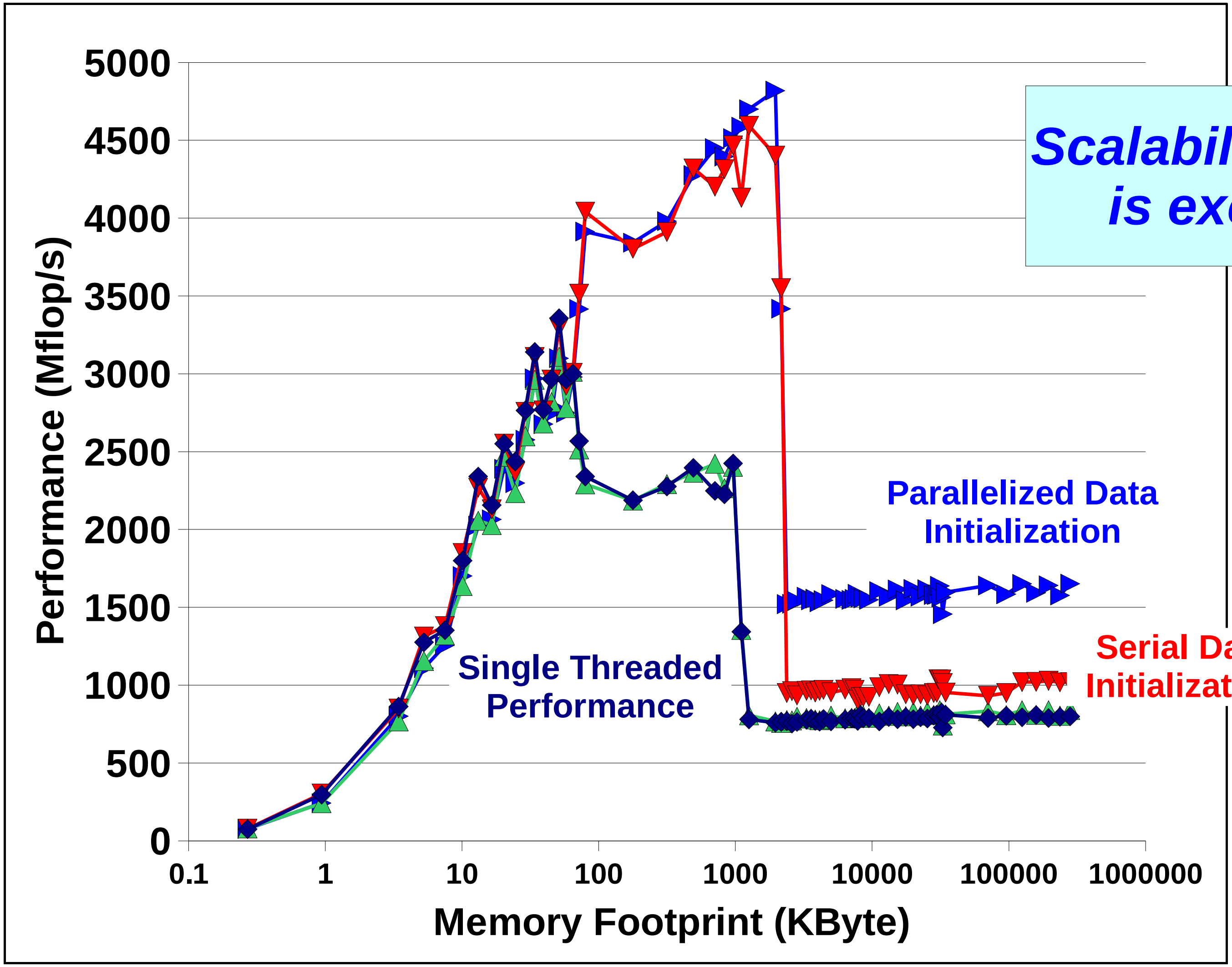
```
#pragma omp parallel default(none) \
        shared(m,n,a,b,c) private(i,j)
{
#pragma omp for
   for (j=0; j<n; j++)
      c[j] = 1.0;

#pragma omp for
   for (i=0; i<m; i++)
   {
      a[i] = -1957.0;
      for (j=0; j<n; j++)
         b[i*n+j] = i;
   } /*-- End of omp for --*/

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



# Example - Exploit First Touch

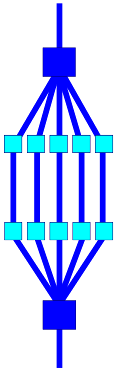


*Scalability on 2 threads is excellent now !*

*The only change is the way the data is distributed over the system*

AMD Opteron Rev F @2600MHz
L1 cache : 64 KByte
Peak speed : 5200 Mflop/s

# Summary Case Studies



- ❑ *There are several important basic aspects to consider when it comes to writing an efficient OpenMP program*
- ❑ *Moreover, there are also obscure additional aspects:*
  - *cc-NUMA*
  - *False Sharing*
- ❑ *Key problem is that most developers are not aware of these rules and .... blaming OpenMP is all that easy*
  - *In some cases it is a trade-off between ease of use and performance*
  - *OpenMP typically goes for the former, but .....*
    - ✔ ***With some extra effort can be made to scale well in many cases***

# *The Wrapping*

# Wrapping Things Up

***“While we're still waiting for your MPI debug run to finish, I want to ask you whether you found my information useful.”***

***“Yes, it is overwhelming. I know.”***

***“And OpenMP is somewhat obscure in certain areas. I know that as well.”***

***“I understand. You're not a Computer Scientist and just need to get your scientific research done.”***

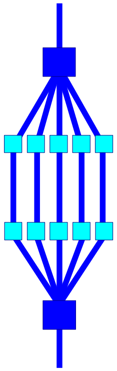
***“I agree this is not a good situation, but it is all about Darwin, you know. I'm sorry, it is a tough world out there.”***

# It Never Ends

*“Oh, your MPI job just finished! Great.”*

***“Your program does not write a file called 'core' and it wasn't there when you started the program?”***

***“You wonder where such a file comes from? Let's get a big and strong coffee first.”***



# That's It

# Thank You and ..... Stay Tuned !

**Ruud van der Pas**
***ruud.vanderpas@sun.com***