“OpenMP Does Not Scale”

Ruud van der Pas

Distinguished Engineer

Architecture and Performance

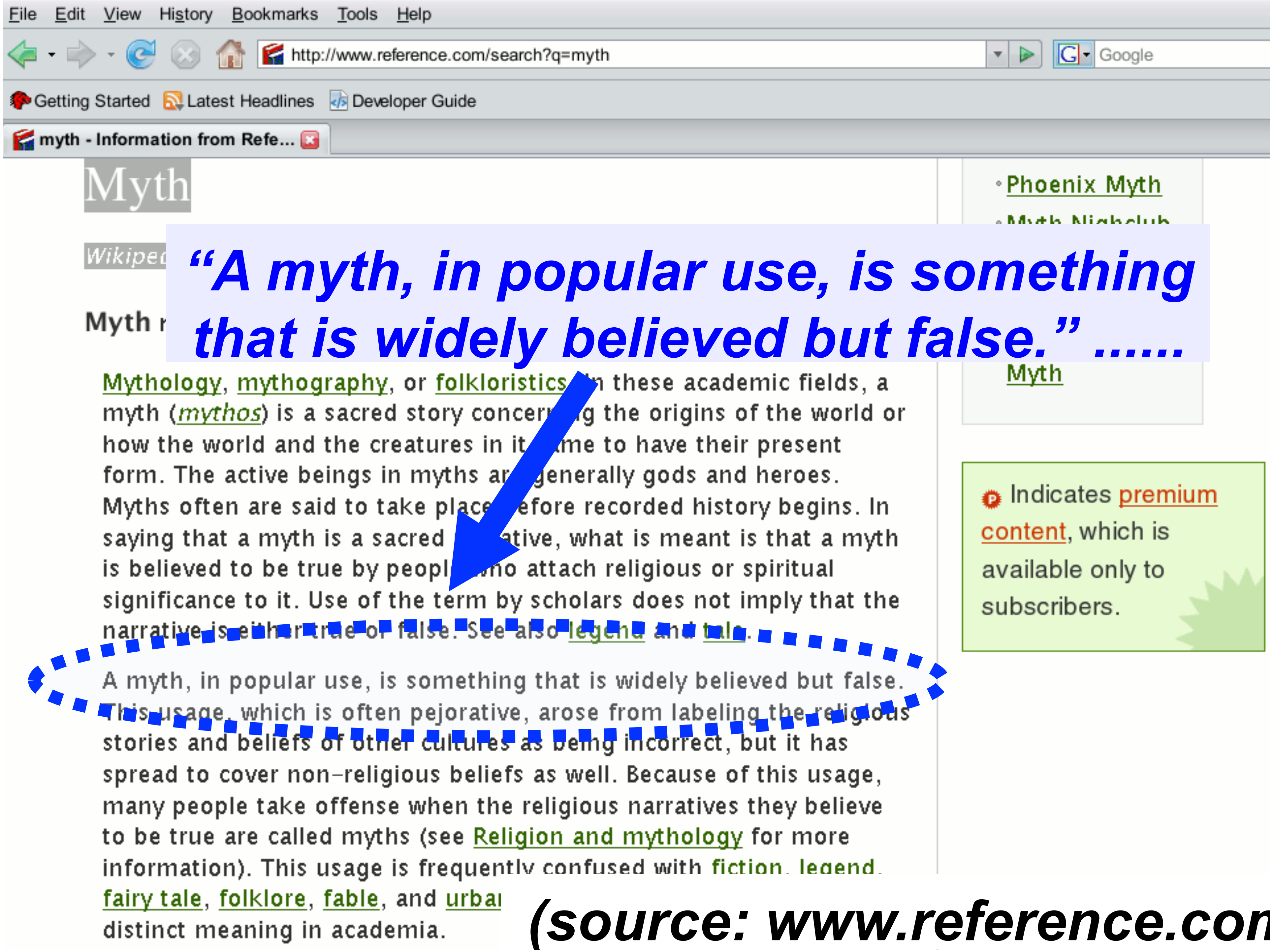
SPARC Microelectronics, Oracle

Santa Clara, CA, USA

Agenda

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

The Myth



"A myth, in popular use, is something that is widely believed but false." ......

(source: www.reference.com

"OpenMP Does Not Scale"

A Common Myth

A Programming Model Can Not "Not Scale"

What Can Not Scale:

The Implementation

The System Versus The Resource Requirements

Or ..... You

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?"

Some Questions I Could Ask

"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 financial bail out plan. It is something else."

Some Questions I Could Ask

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

“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.

More Questions I Could Ask

"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 threads roughly perform the same amount of work?"

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

ORACLE

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.

I Don't Give Up That Easily

"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."

The Grass Is Always Greener ...

"So, what did you do next to address the performance ?"

"Switched to MPI. I see. 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/1

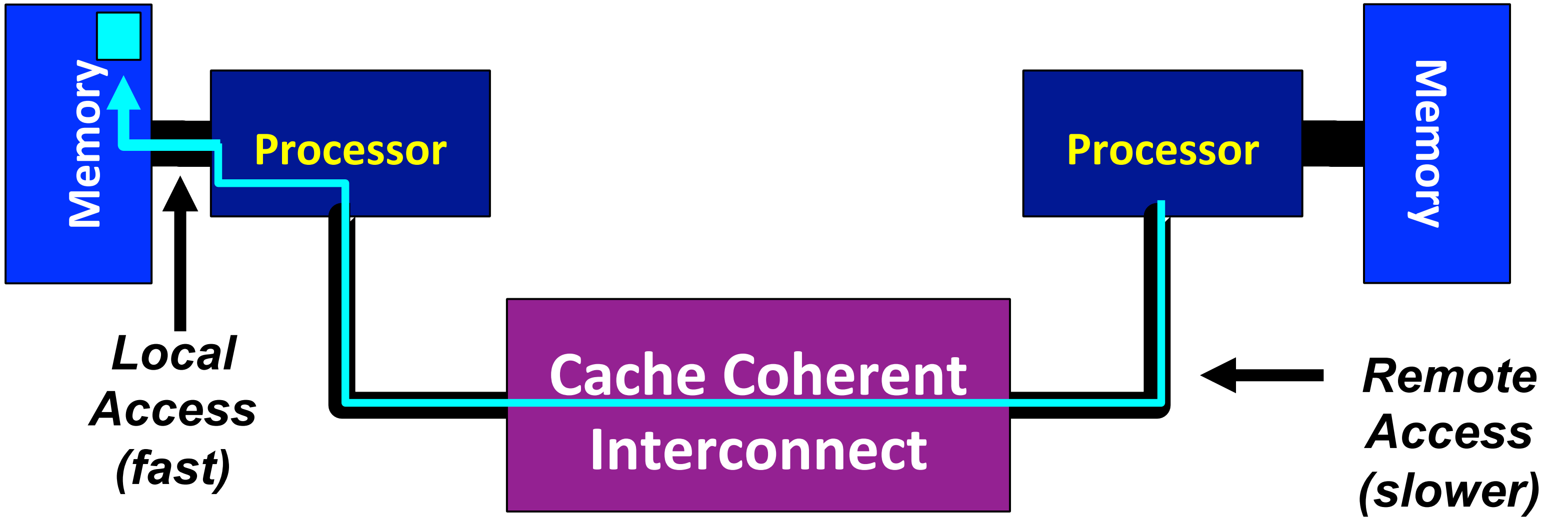
- The transparency and ease of use of OpenMP are a mixed blessing
  - → Makes things pretty easy
  - → May mask performance bottlenecks
- In the ideal world, an OpenMP application “just performs well”
- Unfortunately, this is not always the case

OpenMP And Performance/2

- Two of the more obscure things that can negatively impact performance are cc-NUMA effects and False Sharing
- ***<u>Neither of these are restricted to OpenMP</u>***
  - →They come with shared memory programming on modern cache based systems
  - →But they might show up because you used OpenMP
  - →In any case they are important enough to cover here

Considerations for cc-NUMA

A Generic cc-NUMA Architecture

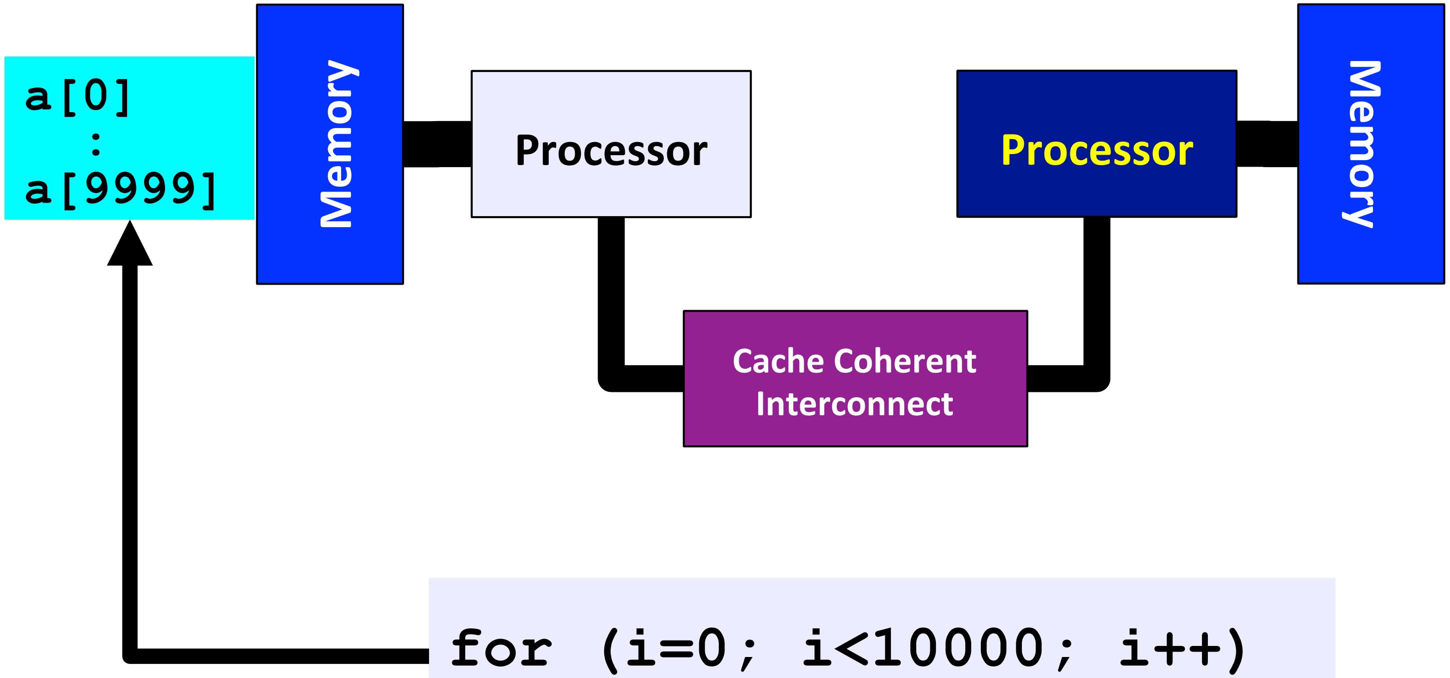


Main Issue: How To Distribute The Data ?

About Data Distribution

- Important aspect on cc-NUMA systems
  - →If not optimal, longer memory access times and hotspots
- OpenMP 4.0 does provide support for cc-NUMA
  - →Placement under control of the Operating System (OS)
  - →User control through OMP_PLACES
- Windows, Linux and Solaris all use the “First Touch” placement policy by default
  - →May be possible to override default (check the docs)

Example First Touch Placement/1

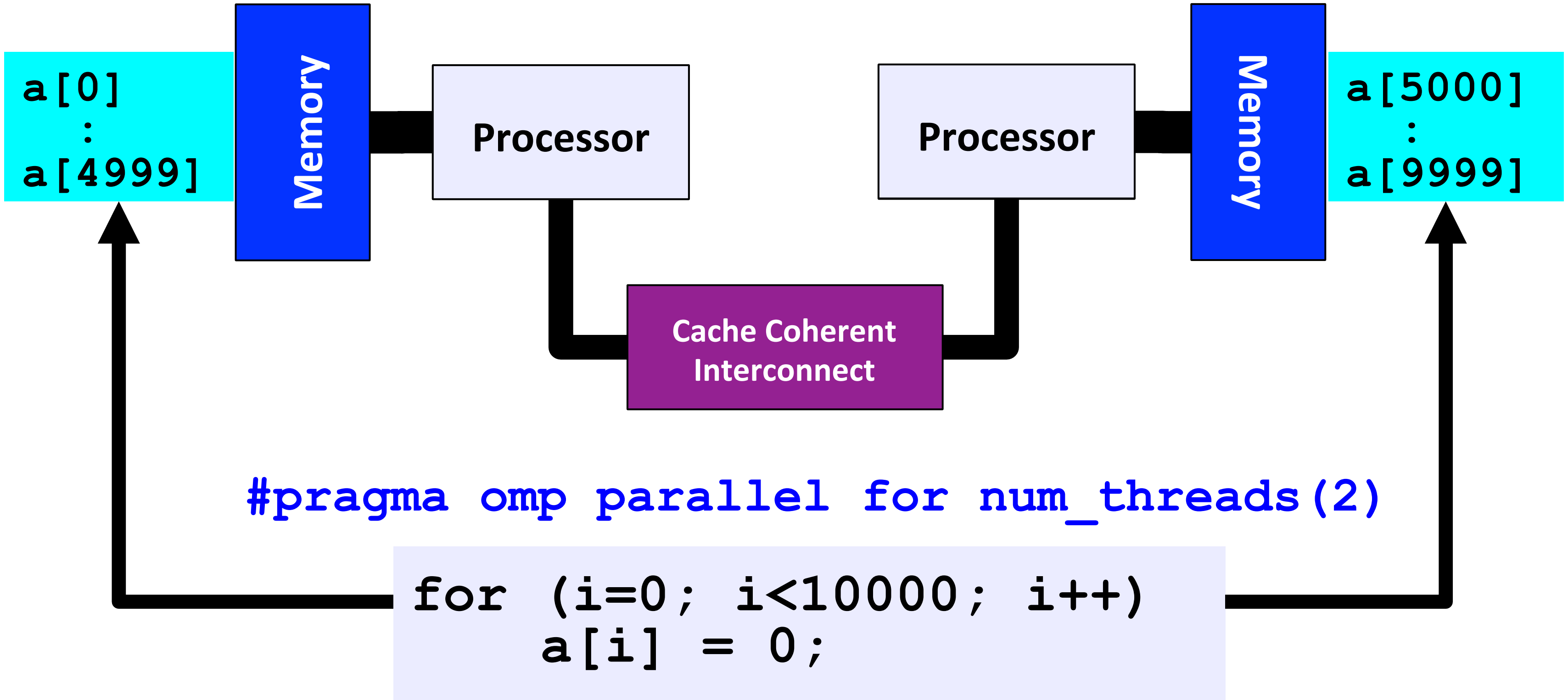



```
for (i=0; i<10000; i++)
     a[i] = 0;
```

First Touch

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

Example First Touch Placement/2




```
#pragma omp parallel for num_threads(2)
for (i=0; i<10000; i++)
      a[i] = 0;
```

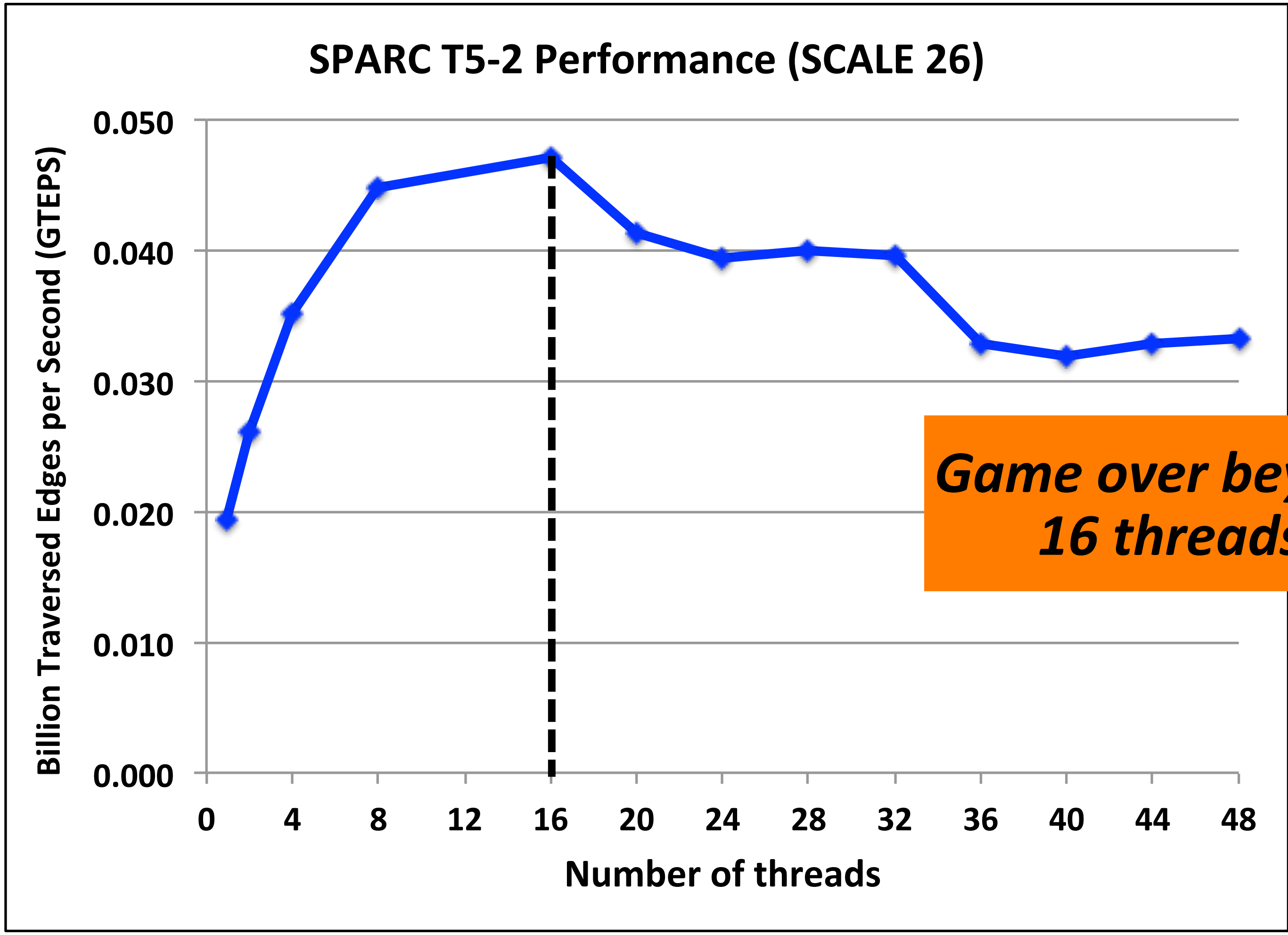
First Touch

Both memories now have "their own half" of the array

Get Real


"Don't Try This At Home (yet)"

The Initial Performance (35 GB)

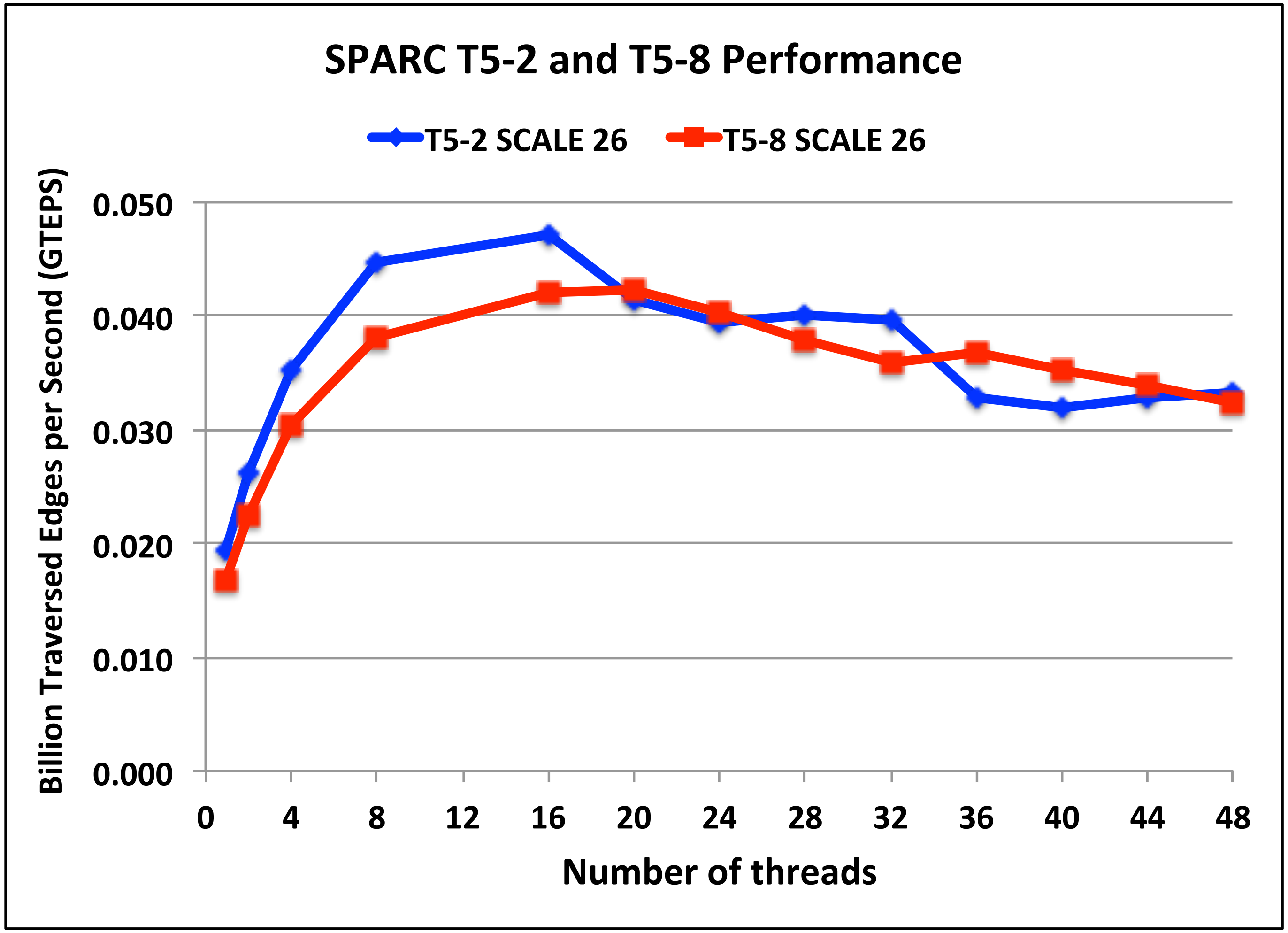


Game over beyond 16 threads

That doesn’t scale very well

Let’s use a bigger machine !

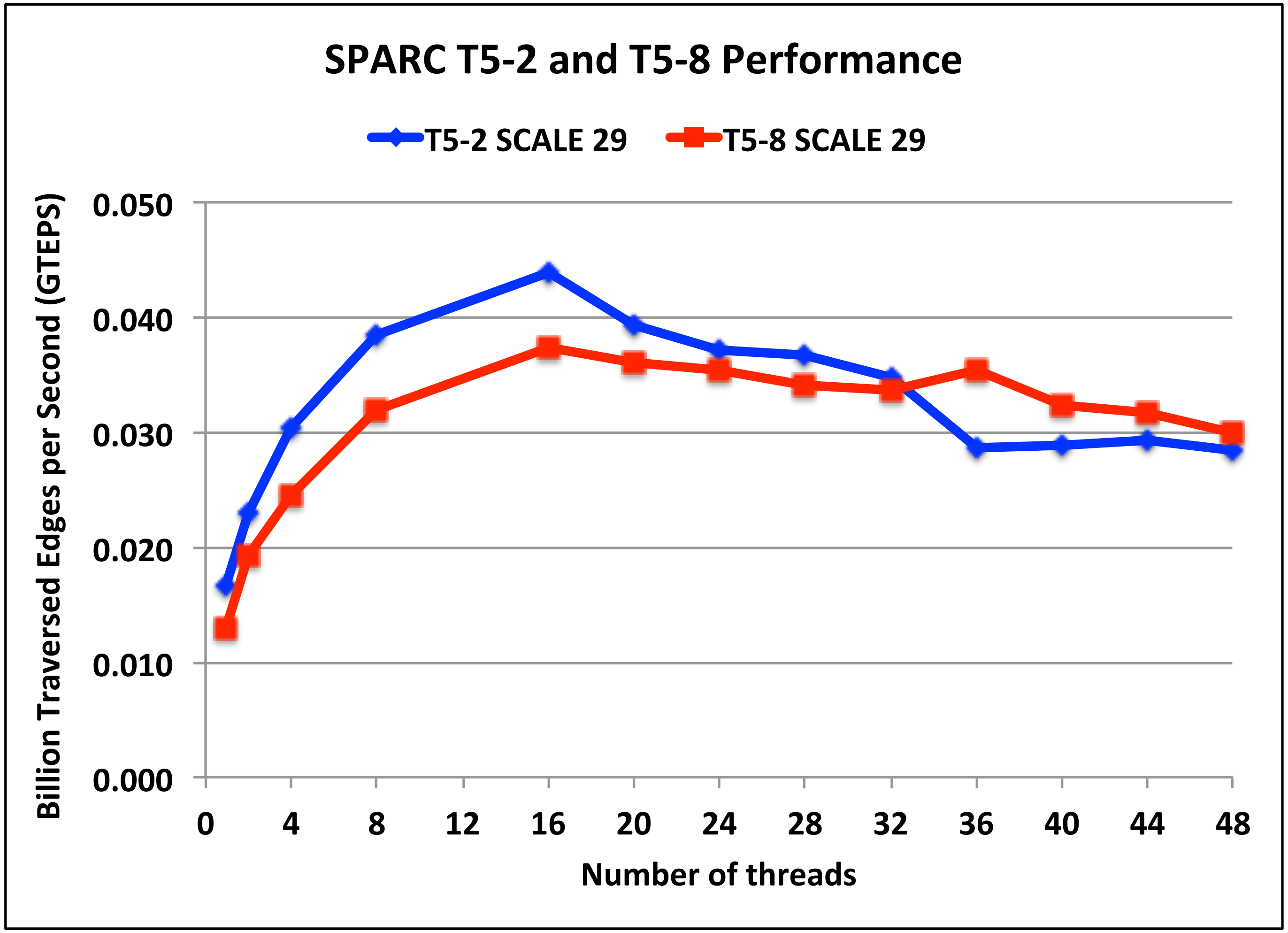
Initial Performance (35 GB)



Oops! That can't be true

Let's run a larger graph !

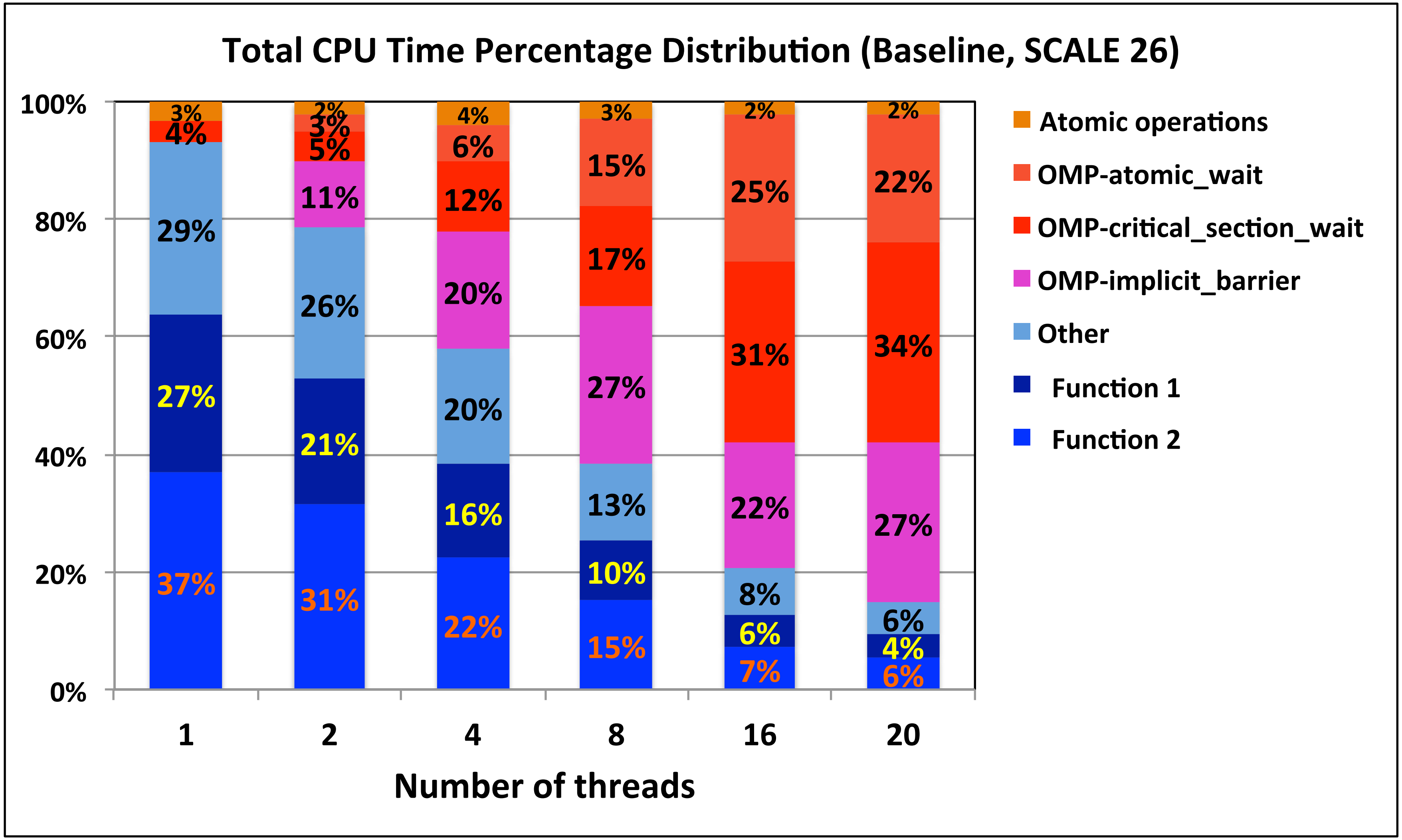
Initial Performance (280 GB)



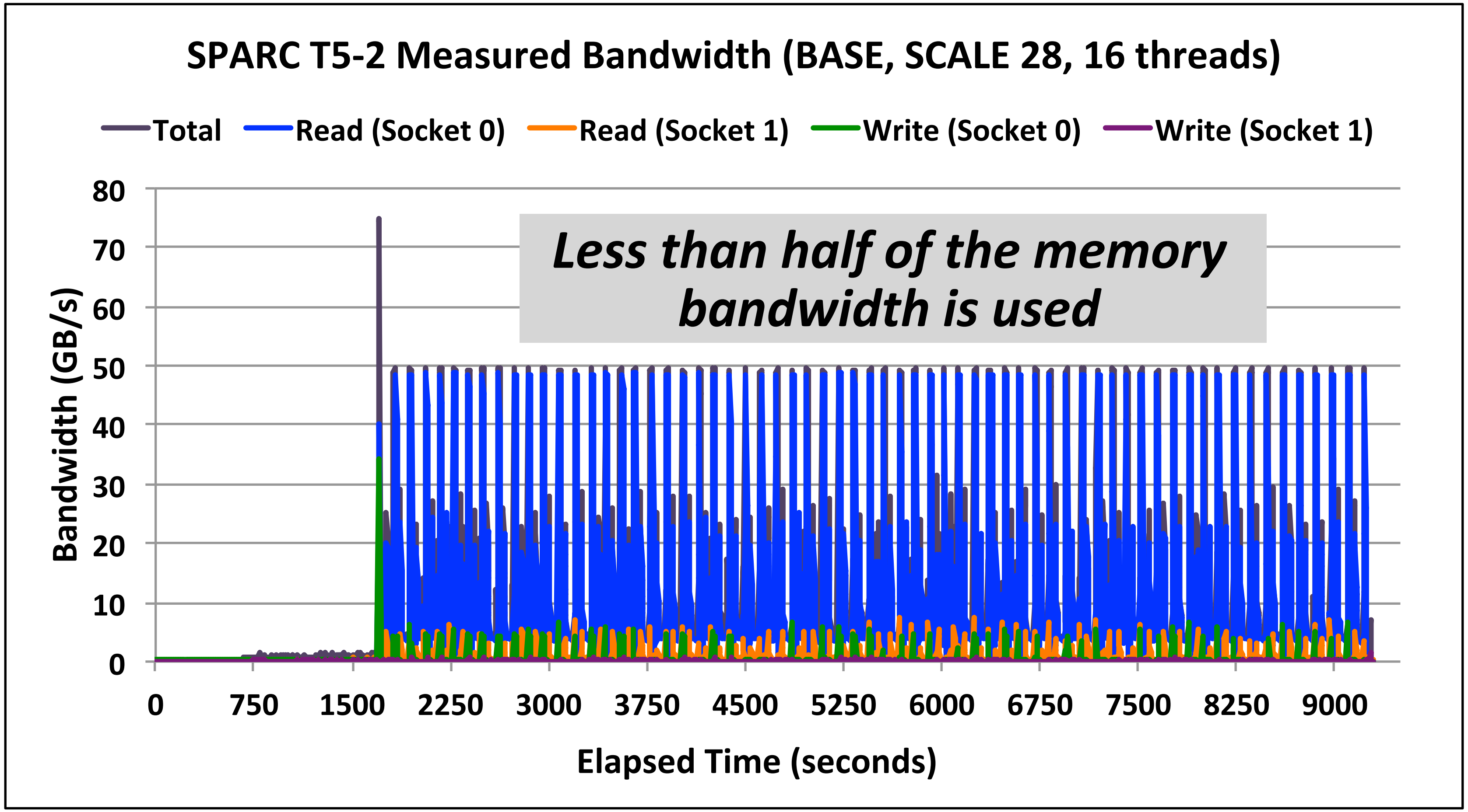
ORACLE

Let's Get Technical

Total CPU Time Distribution



Bandwidth Of The Original Code



Summary Original Version

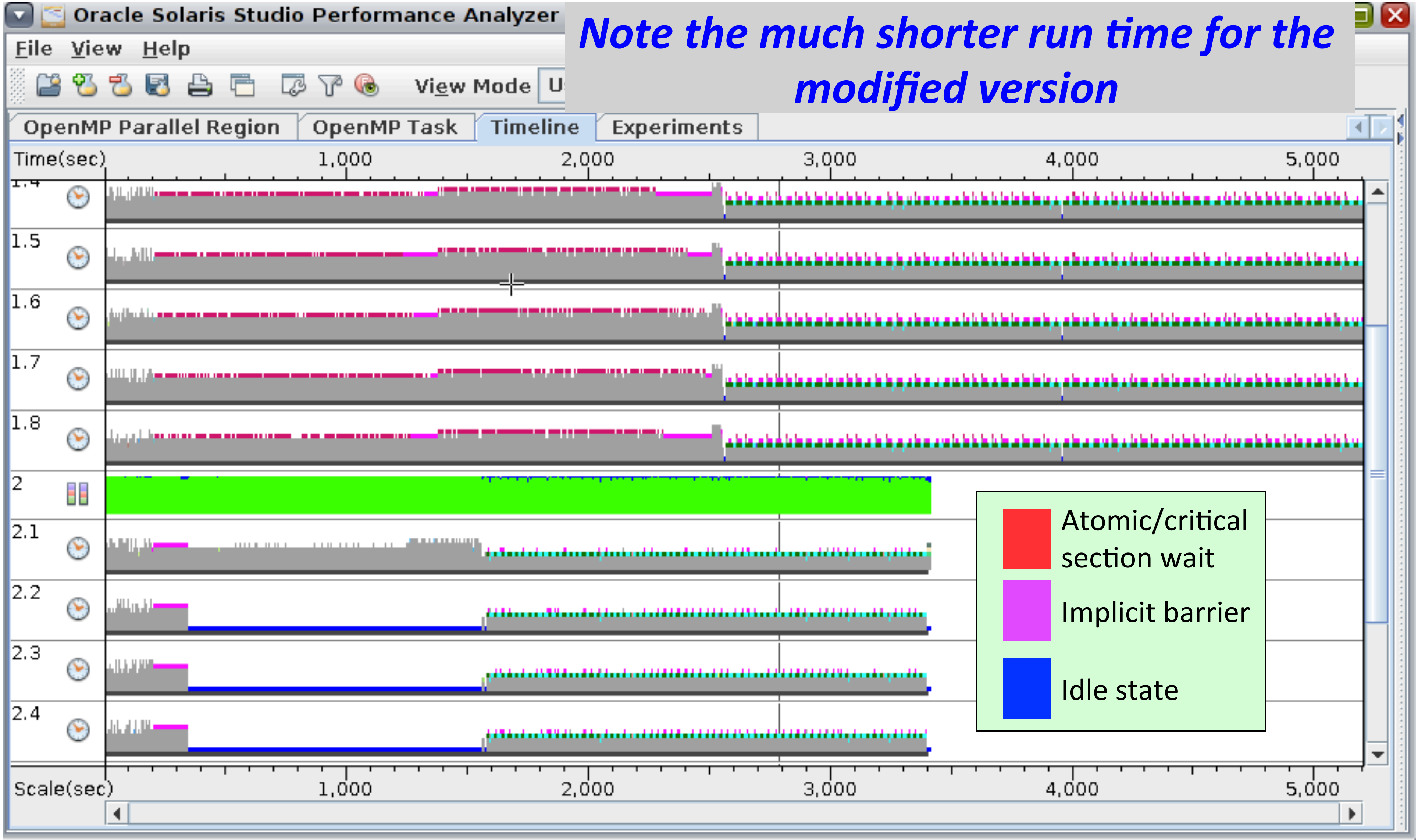
- Communication costs are too high
  - Increases as threads are added
  - This seriously limits the number of threads used
  - This is turn affects memory access on larger graphs
- The bandwidth is not balanced
- Fixes:
  - Find and fix many OpenMP inefficiencies
  - Use some efficient atomic functions

Methodology

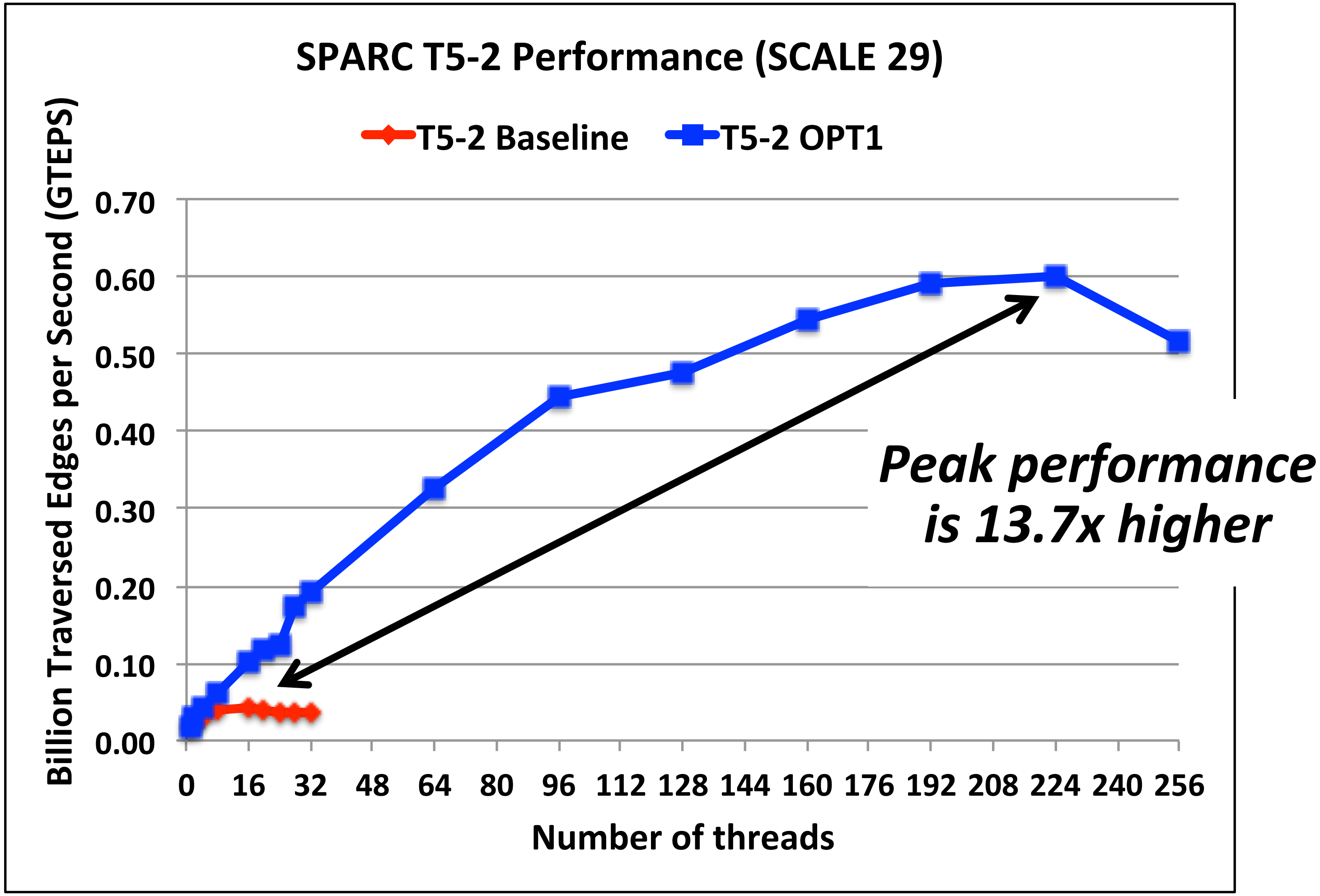
If The Code Does Not Scale Well
Use A Profiling Tool
Use The Checklist To Identify Bottlenecks
Tackle Them One By One
This Is An Incremental Approach
But Very Rewarding


Secret Sauce
BO
BO

Comparison Of The Two Versions



Performance Comparison



BO

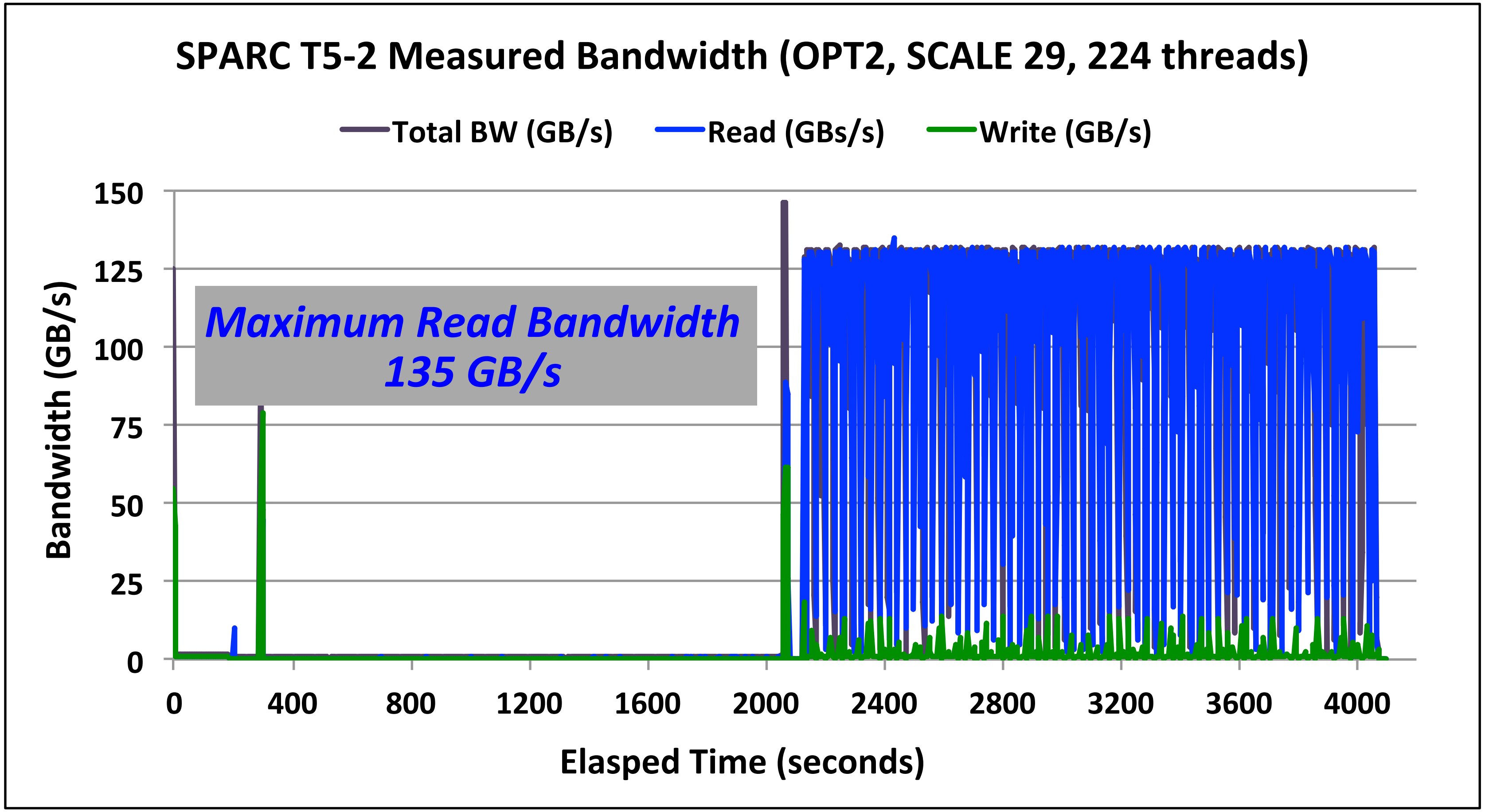
More
Secret Sauce

MO

Observations

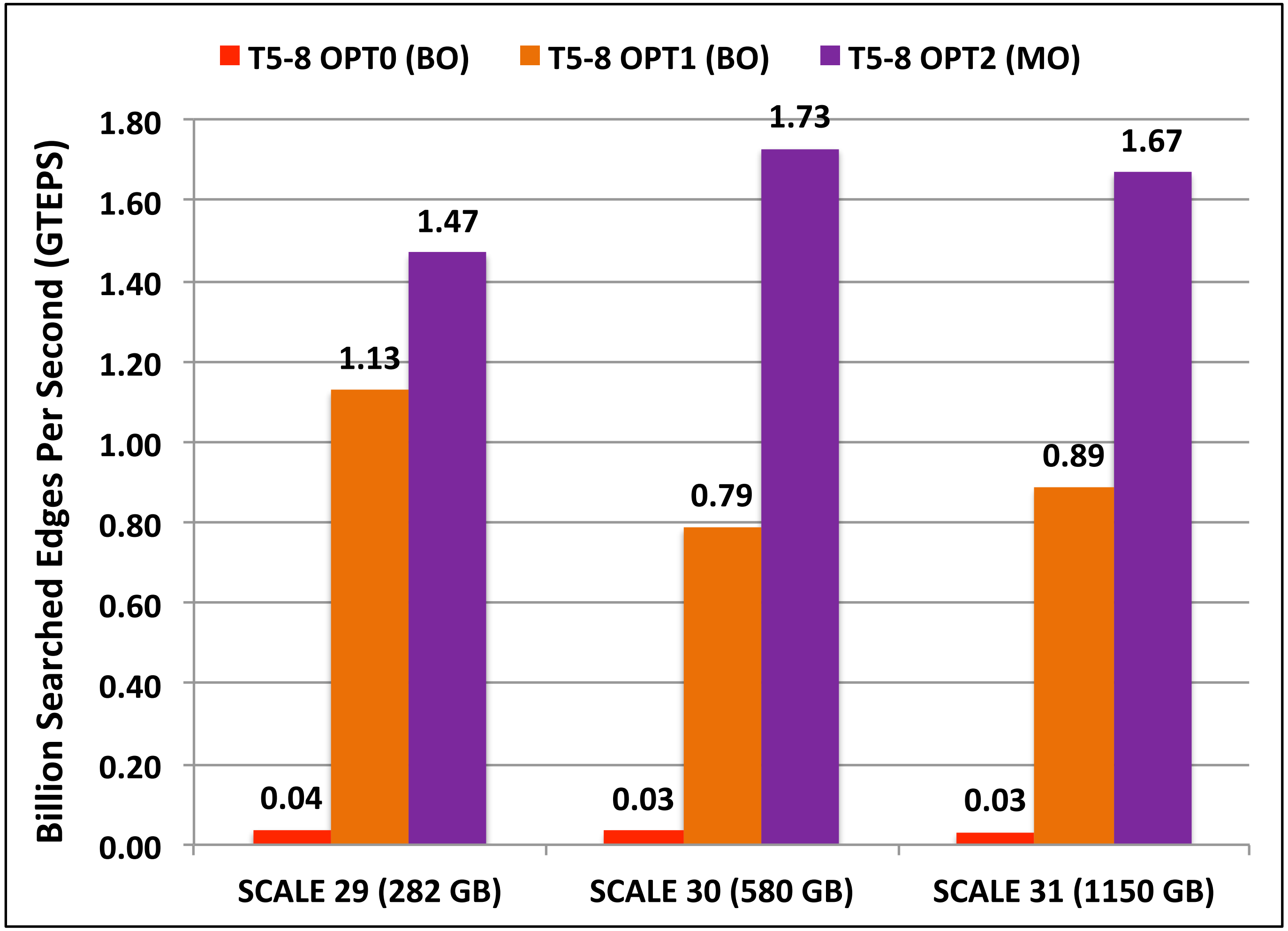
First Touch Placement Is Not Used
The Code Does Not Exploit Large Pages
But Needs It ….
Used A Smarter Memory Allocator

Bandwidth Of The New Code



The Result

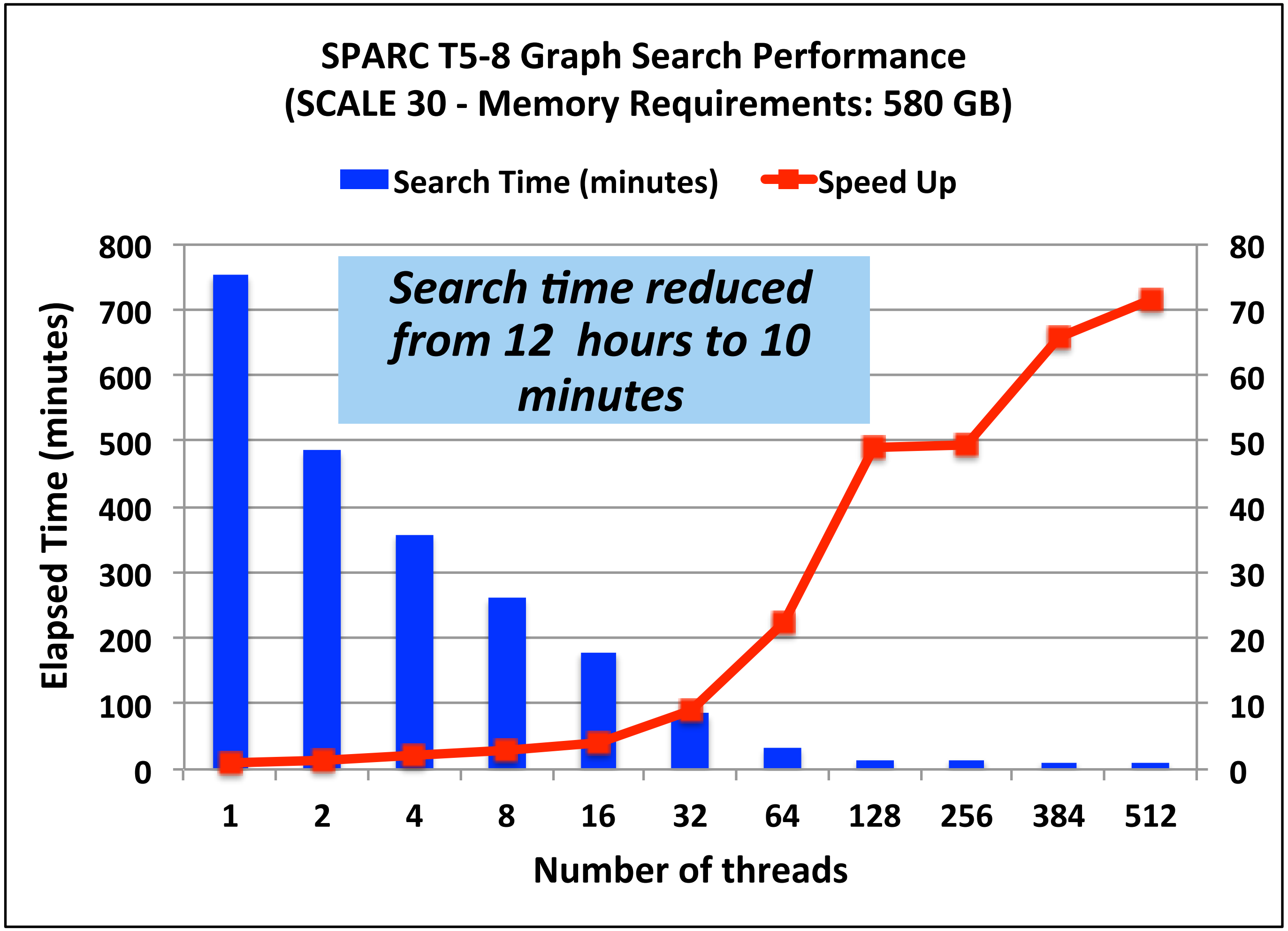
39-52x improvement over original code



ORACLE

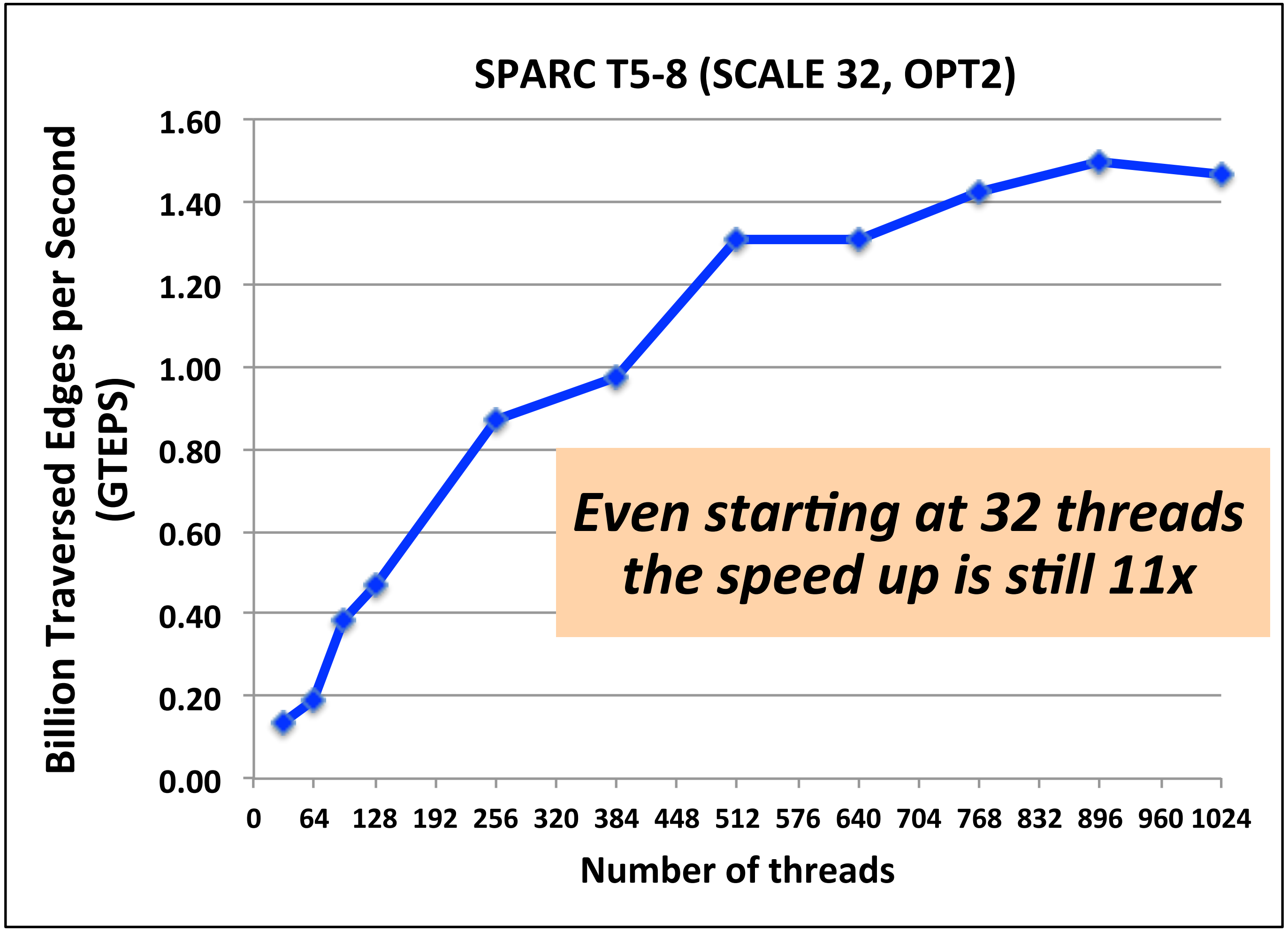
Bigger Is Definitely Better!

72x speed up !



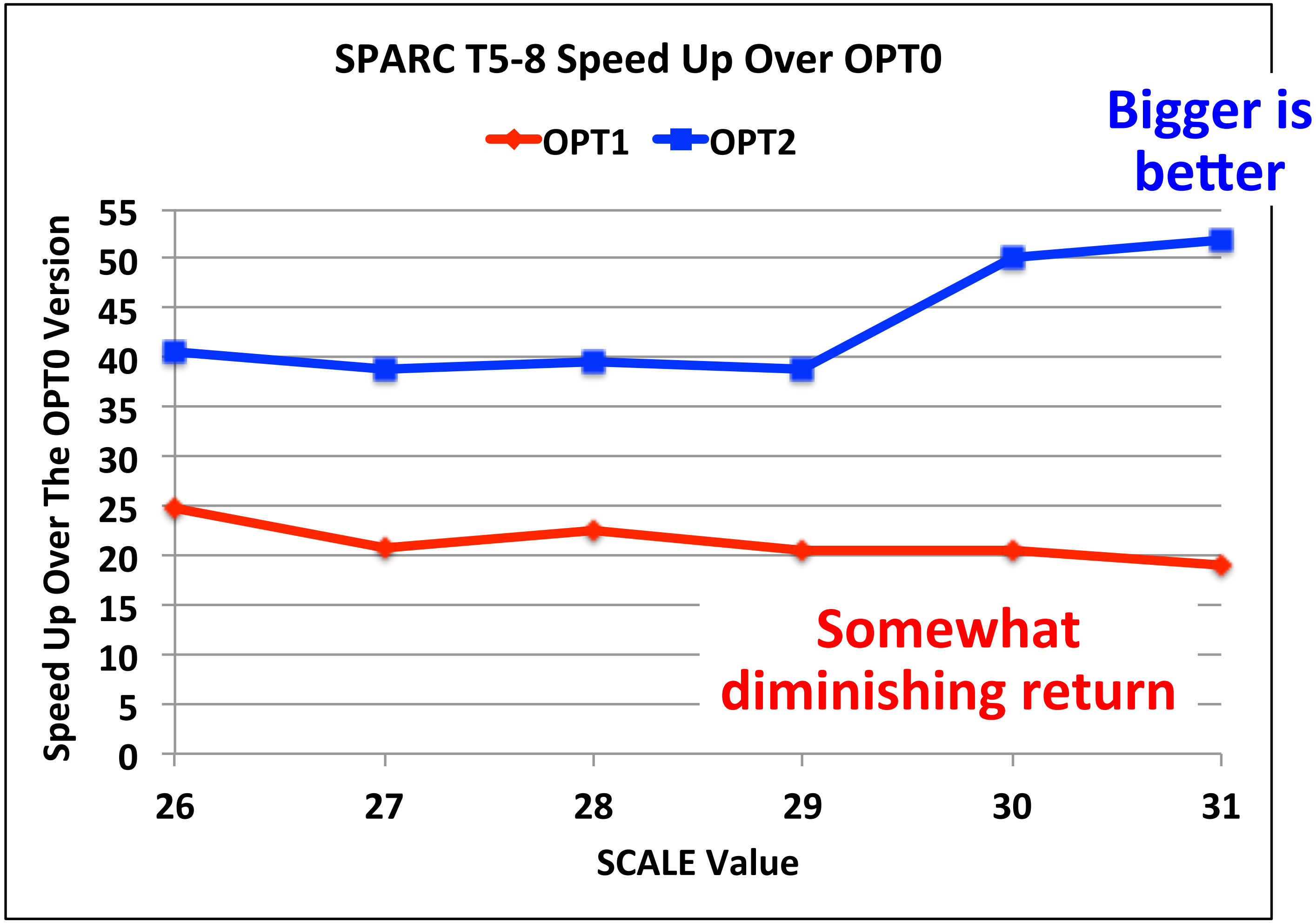
A 2.3 TB Sized Problem

896 Threads !



ORACLE

Tuning Benefit Breakdown

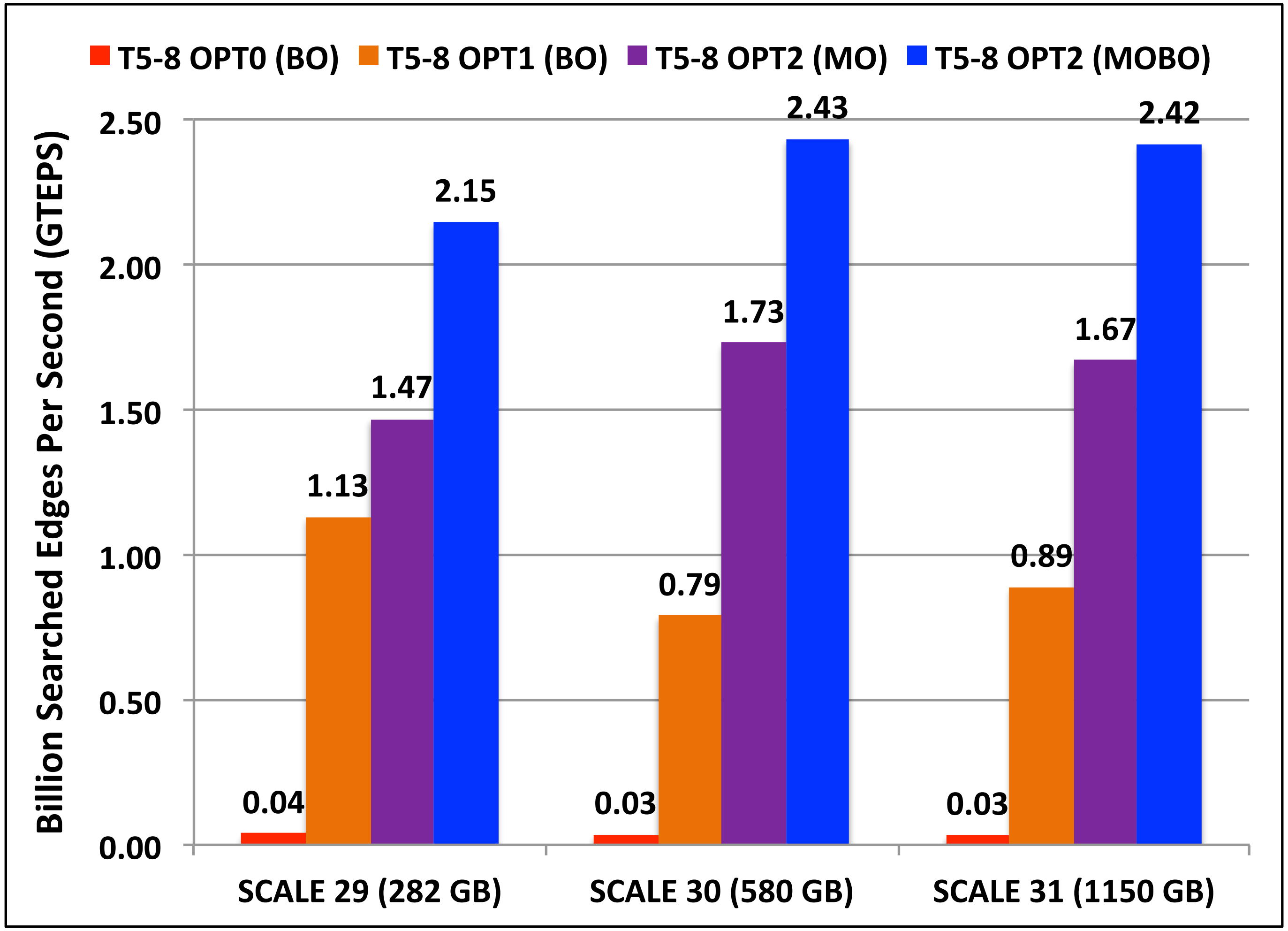


MO → **MOBO**

Different Secret Sauce

A Simple OpenMP Change

57-75x improvement

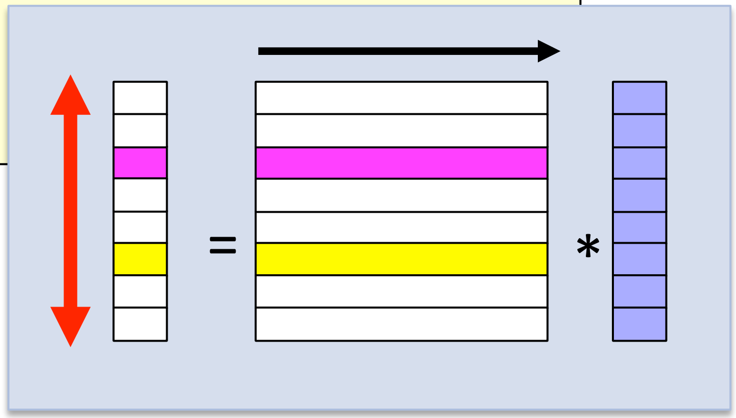


“I Value My Personal Space”

My Favorite Simple Algorithm

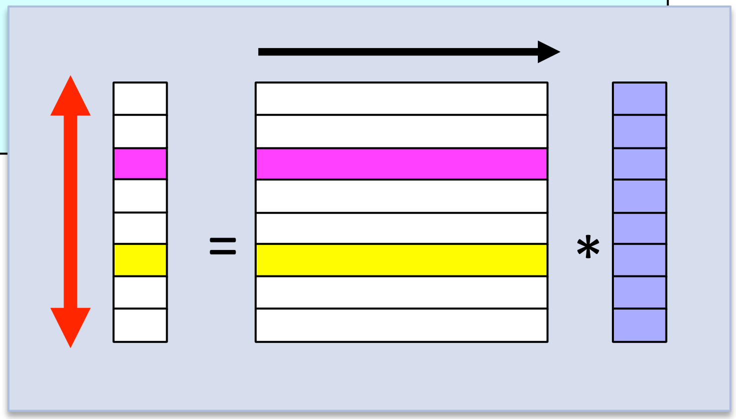
```
void mxv(int m,int n,double *a,double *b[],
         double *c)
{
   for (int i=0; i<m; i++)
   {
      double sum = 0.0;
      for (int j=0; j<n; j++)
          sum += b[i][j]*c[j];
      a[i] = sum;
   }
}
```

parallel loop

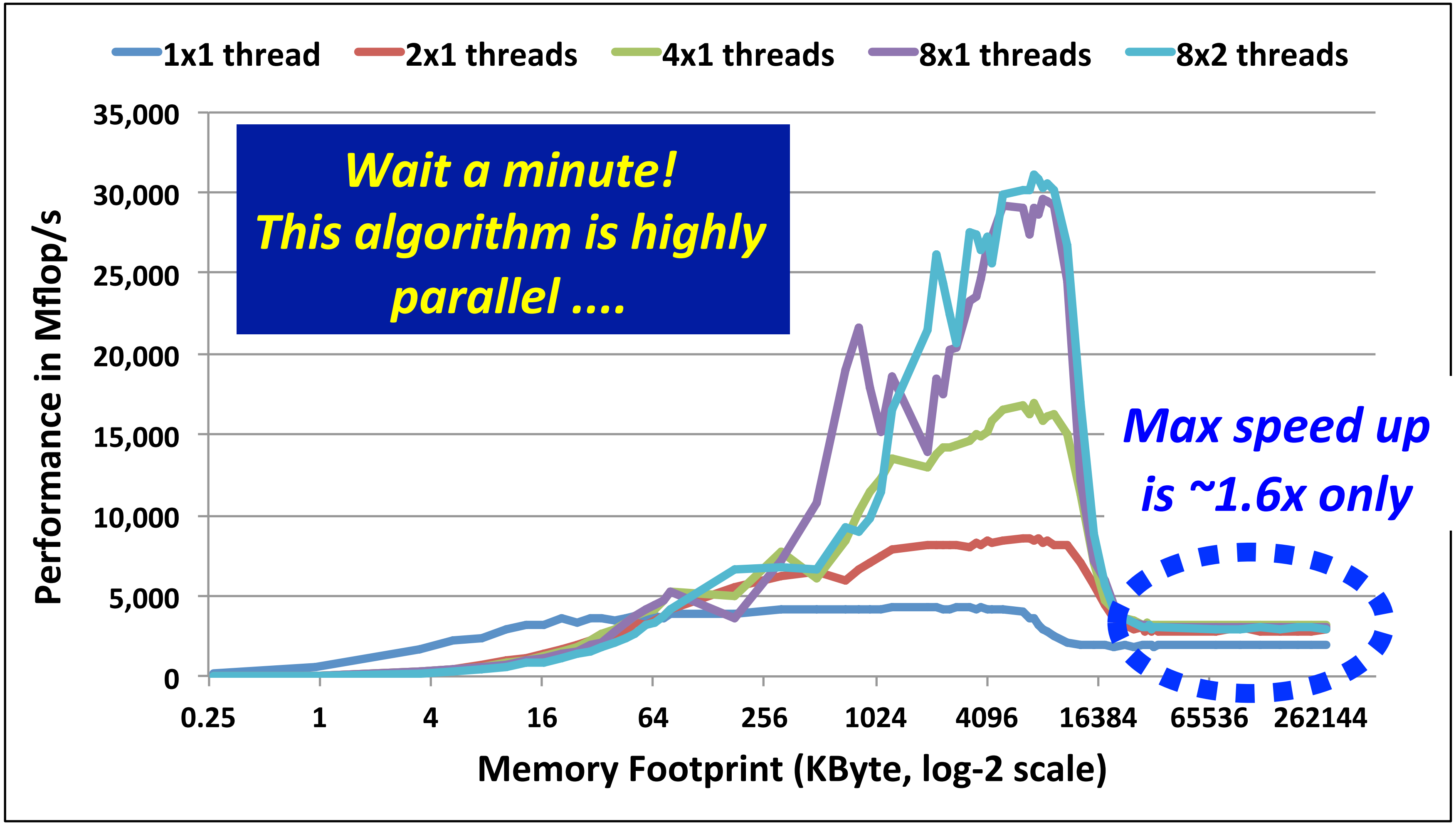


The OpenMP Source

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



Performance On Intel Nehalem



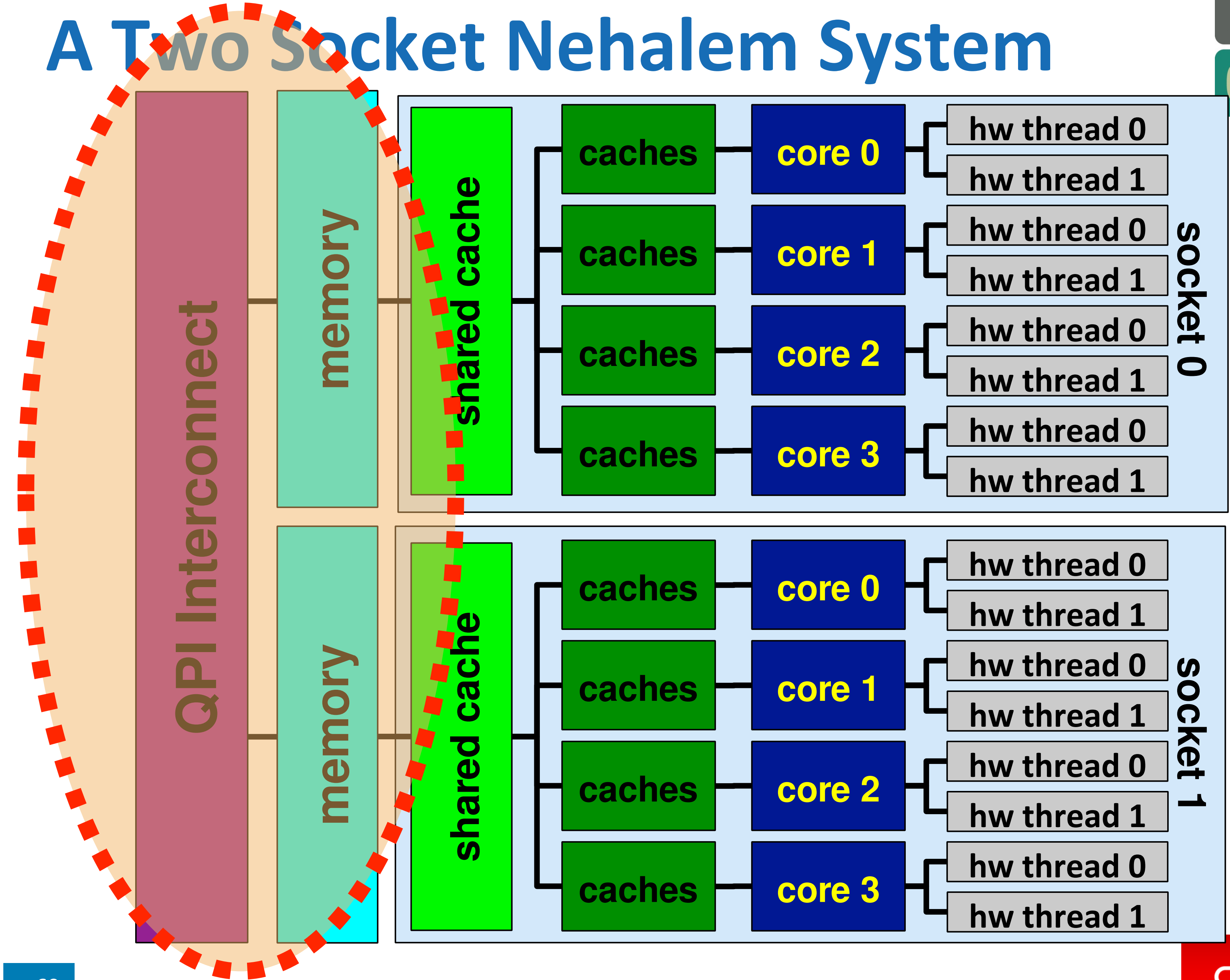
System: Intel X5570 with 2 sockets, 8 cores, 16 threads at 2.93 GHz

Notation: Number of cores x number of threads within core

?

Let's Get Technical

A Two Socket Nehalem System

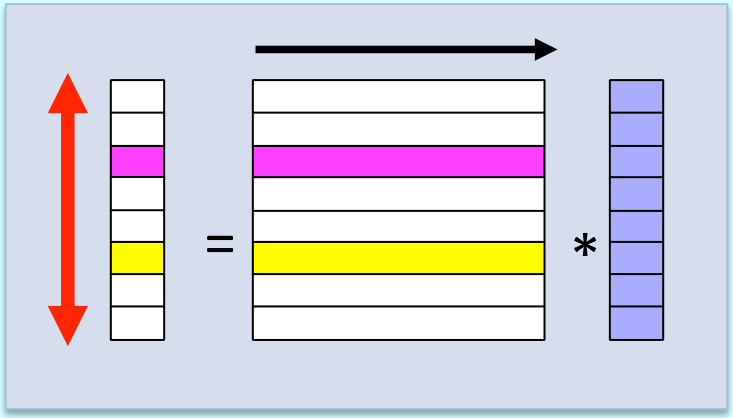


Data Initialization Revisited

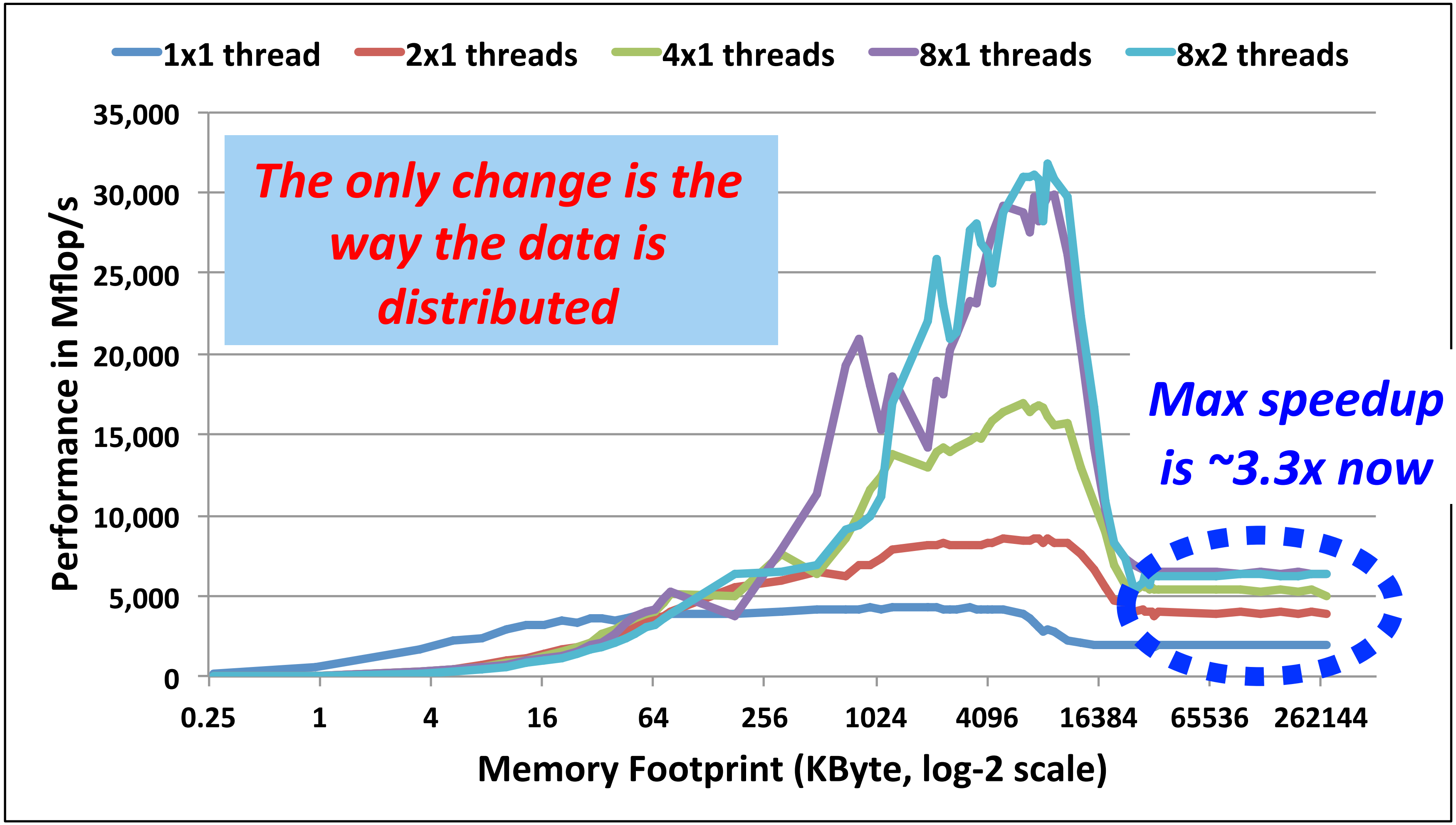
```
#pragma omp parallel default(none) \
        shared(m,n,a,b,c) private(i,j)
{
#pragma omp for
   for (j=0; j<n; j++)
      c[j] = 1.0;

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

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



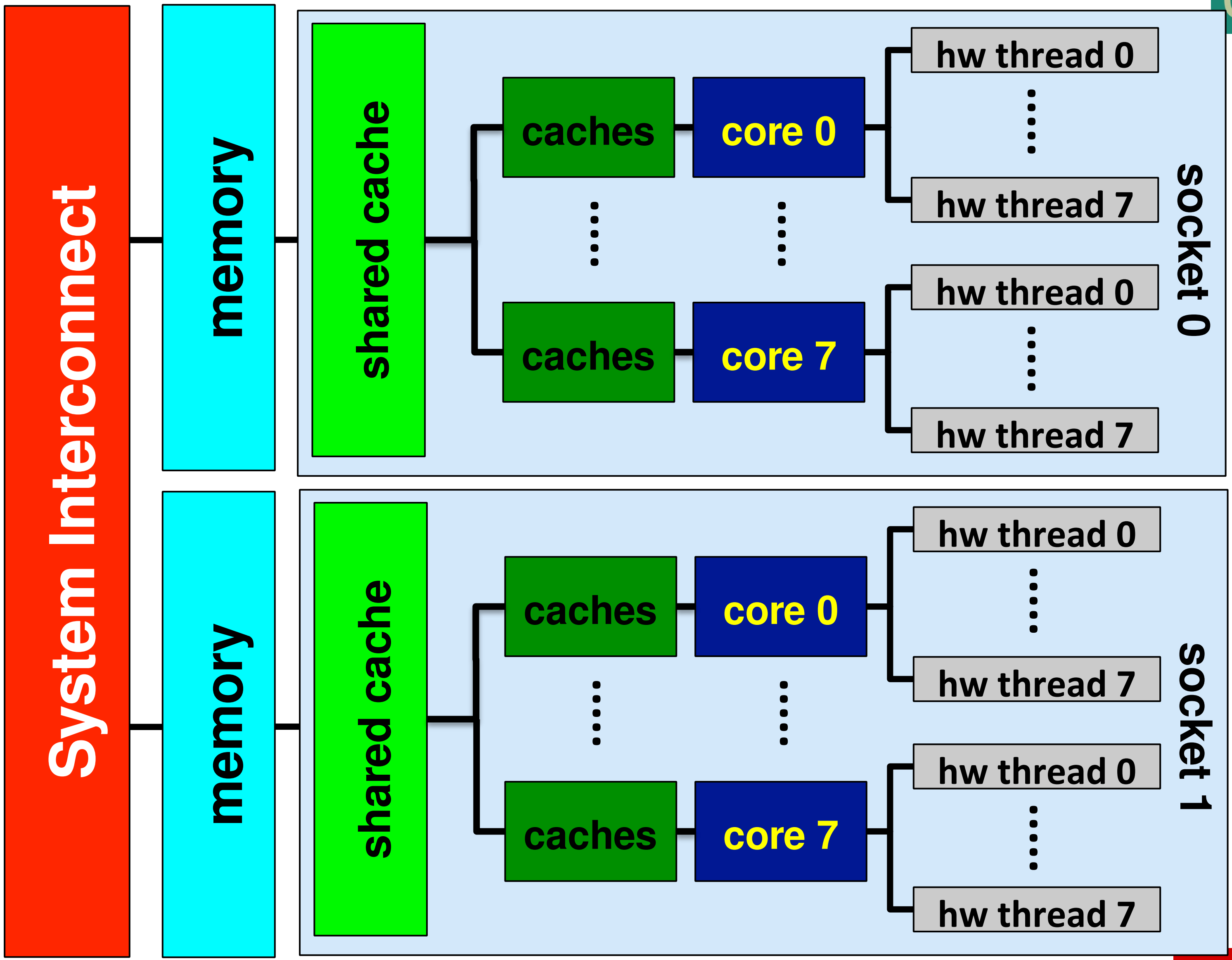
Data Placement Matters!



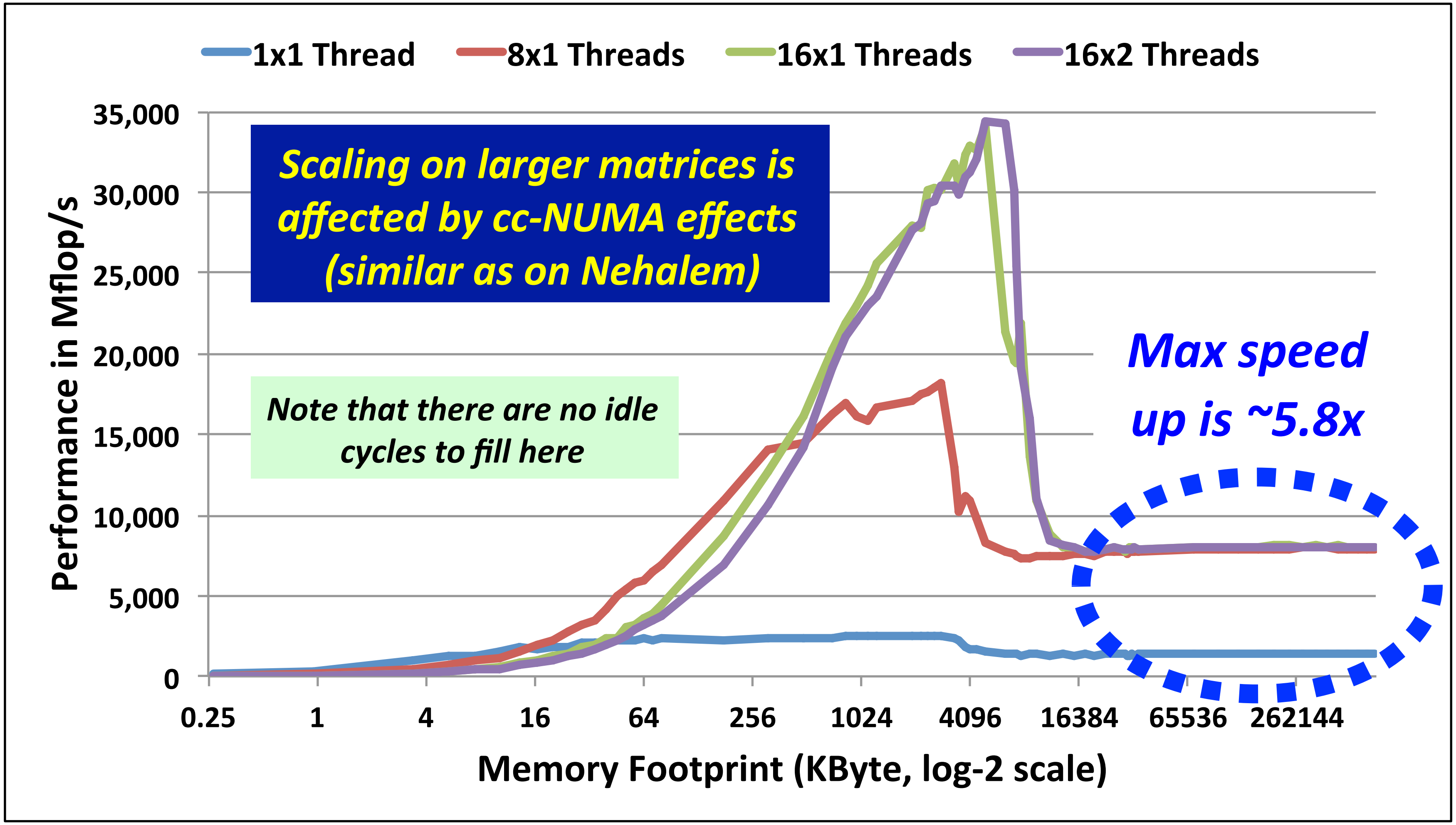
System: Intel X5570 with 2 sockets, 8 cores, 16 threads at 2.93 GHz

__Notation:__ Number of cores x number of threads within core

A Two Socket SPARC T4-2 System



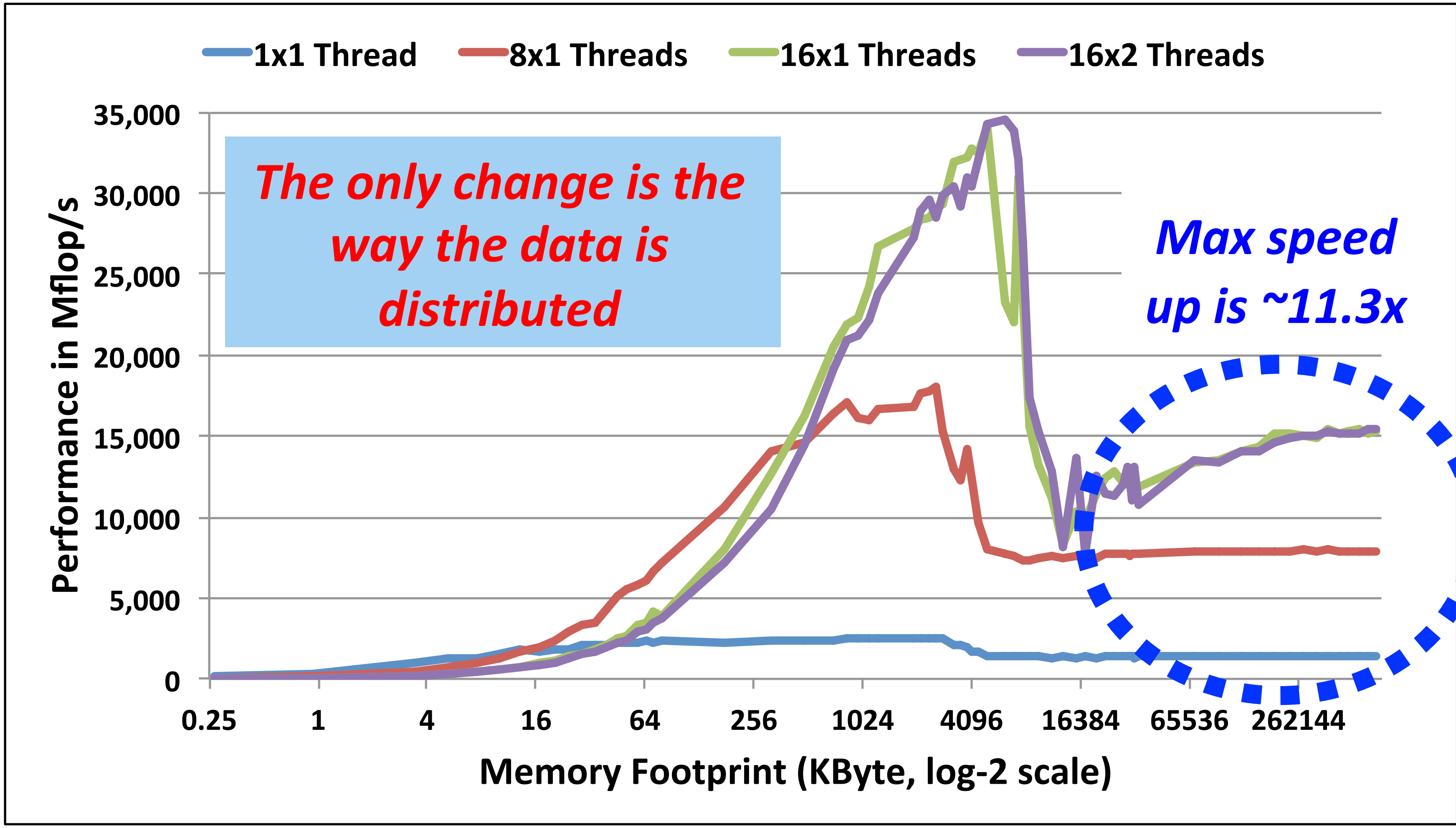
Performance On SPARC T4-2



System: SPARC T4 with 2 sockets, 16 cores, 128 threads at 2.85 GHz

<u>Notation:</u> Number of cores x number of threads within core

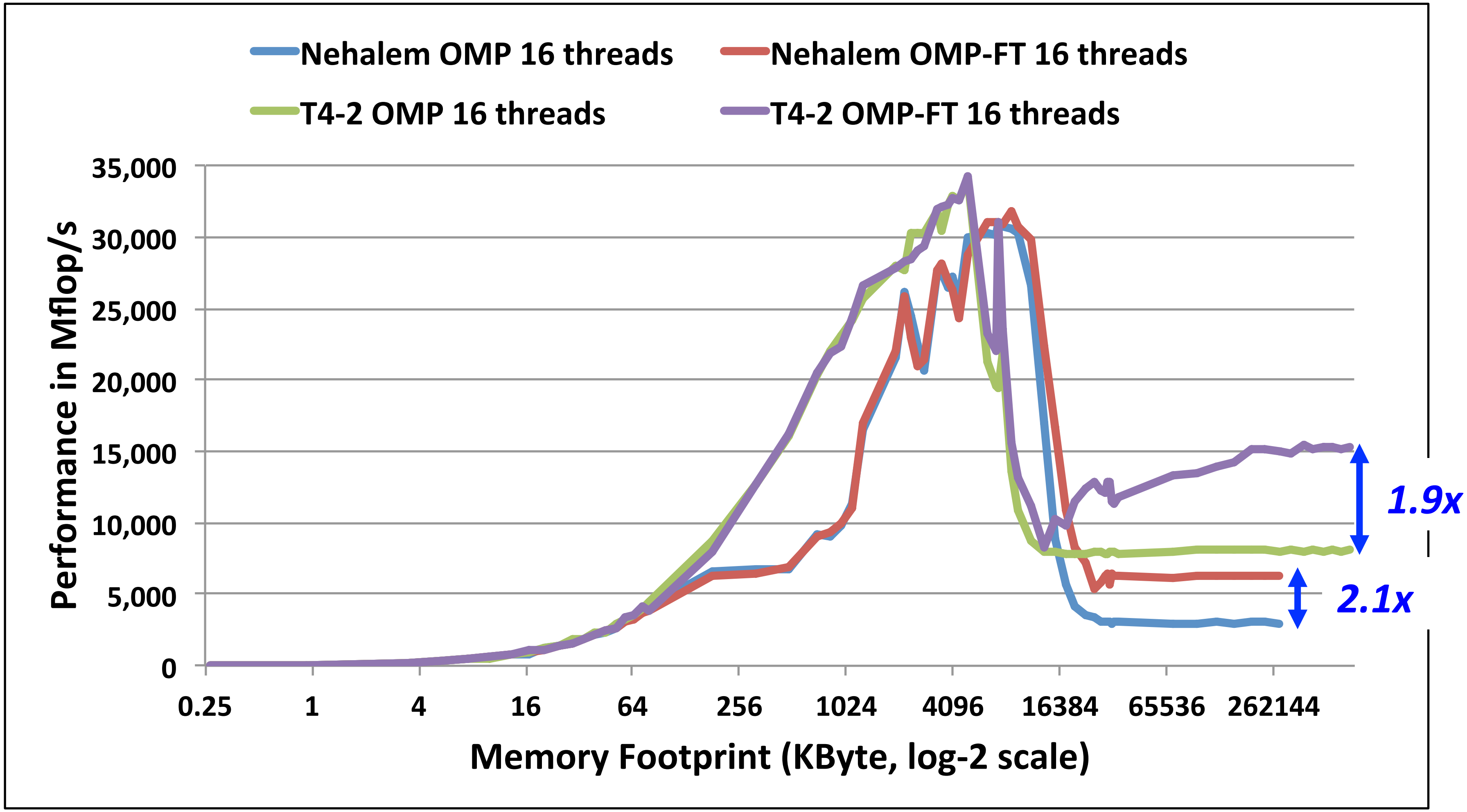
Data Placement Matters!



System: SPARC T4 with 2 sockets, 16 cores, 128 threads at 2.85 GHz

***Notation:** Number of cores x number of threads within core*

Summary Matrix Times Vector



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."

Wrapping Things Up

"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 talk, but I need to get a big and strong coffee first."

"WAIT! What did you just say?"

It Really Never Ends

"Somebody told you WHAT ??"

"You think GPUs and OpenCL will solve all your problems?"

"Let's make that an XL Triple Espresso. I'll buy"

Thank You And ..... Stay Tuned !

ruud.vanderpas@oracle.com

DTrace
Why It Can Be Good For You

Ruud van der Pas

Distinguished Engineer

Architecture and Performance

SPARC Microelectronics, Oracle

Santa Clara, CA, USA

Motivation

- DTrace is a **Dynamic Tracing Tool**
  - → Supported on Solaris, Mac OS X and some Linux flavours
- Monitors the Operating System (OS)
- Through “probes”, the user can see what the OS is doing
- Main target: OS related performance issues
- Surprisingly, it can also greatly help to find out what your application is doing though *

) A regular profiling tool should be used first

How DTrace Works

- A DTrace probe is written by the user

```
provider:module:function:name
```

- When the probe "fires", the code in the probe is executed
- The probes are based on "providers"
- The providers are pre-defined
  - →Example: "sched" provider to get info from the scheduler
  - →You can also instrument your own code to have DTrace probes, but there is little need for that

Example – Thread Affinity/1

```
sched:::off-cpu
/ pid == $target && self->on_cpu == 1 /        <-- "predicate"
{
   self->time_delta = (timestamp - ts_base)/1000;

   @thread_off_cpu [tid-1]      = count();
   @total_thr_state[probename] = count();

   printf("Event %8u %4u %6u %6u   %-16s %8s\n",
           self->time_delta, tid-1, curcpu->cpu_id,
           curcpu->cpu_lgrp, probename, probefunc);

   self->on_cpu     = 0;
   self->time_delta = 0;
}
```

Example – Thread Affinity/2

```
pid$target::processor_bind:entry
/ (processorid_t) arg2 >= 0 /
{
   self->time_delta        = (timestamp - ts_base)/1000;
   self->target_processor = (processorid_t) arg2;

   @proc_bind_info[tid-1, curcpu->cpu_id,
                   self->target_processor] = count();

   printf("Event %8u %4u %6u %6u   %9s/%-6d %8s\n",
          self->time_delta, tid-1, curcpu->cpu_id,
          curcpu->cpu_lgrp, "proc_bind", self->target_processor,
          probename);

   self->time_delta        = 0;
   self->target_processor = 0;
}
```

Example – Example Code

```
#pragma omp parallel
{
   #pragma omp single
   {
     printf("Single region executed by thread %d\n",
            omp_get_thread_num());
   } // End of single region
} // End of parallel region
```

```
export OMP_NUM_THREADS=4
export OMP_PLACES=cores
export OMP_PROC_BIND=close
./affinity.d -c ./omp-par.exe
```

Example – Result

```
============================================================
                    Affinity Statistics
============================================================

Thread   On HW Thread   Lgroup   Created Thread   Count
     0            787        7                1       1
     0            787        7                2       1
     0            787        7                3       1

Thread  Running on HW Thread    Bound to HW Thread
     0                   787                   784
     1                   771                   792
     2                   848                   800
     3                   813                   808
```

Thank You And ..... Stay Tuned !

ruud.vanderpas@oracle.com