Shared Memory Parallel Performance To The Extreme

Ruud van der Pas

Distinguished Engineer

Linux Performance Team, Oracle

Santa Clara, CA, USA

About This Presentation

- Unfortunately not all slides shown in the talk can be shared
- I selected those that I think are most relevant when tuning an OpenMP application
- When you go through these slides, you'll come across a check list of things to verify

  → The imaginary conversation is really a check list

- There is a very high chance one or more things on this list cause your code not to scale
- Good luck and remember, don't give up too soon ☺

Purpose Of This Talk

You have just seen your first OpenMP
With this talk I hope to show you the reward
You can get really good performance
But this is not always easy
Just be patient, and …… show up tomorrow ☺

http://www.openmp.org

OpenMP Is The Best Choice

OpenMP is ideally suited for modern architectures

Portable, and portable, and portable

Memory and threading model map naturally

Tight integration with accelerator support

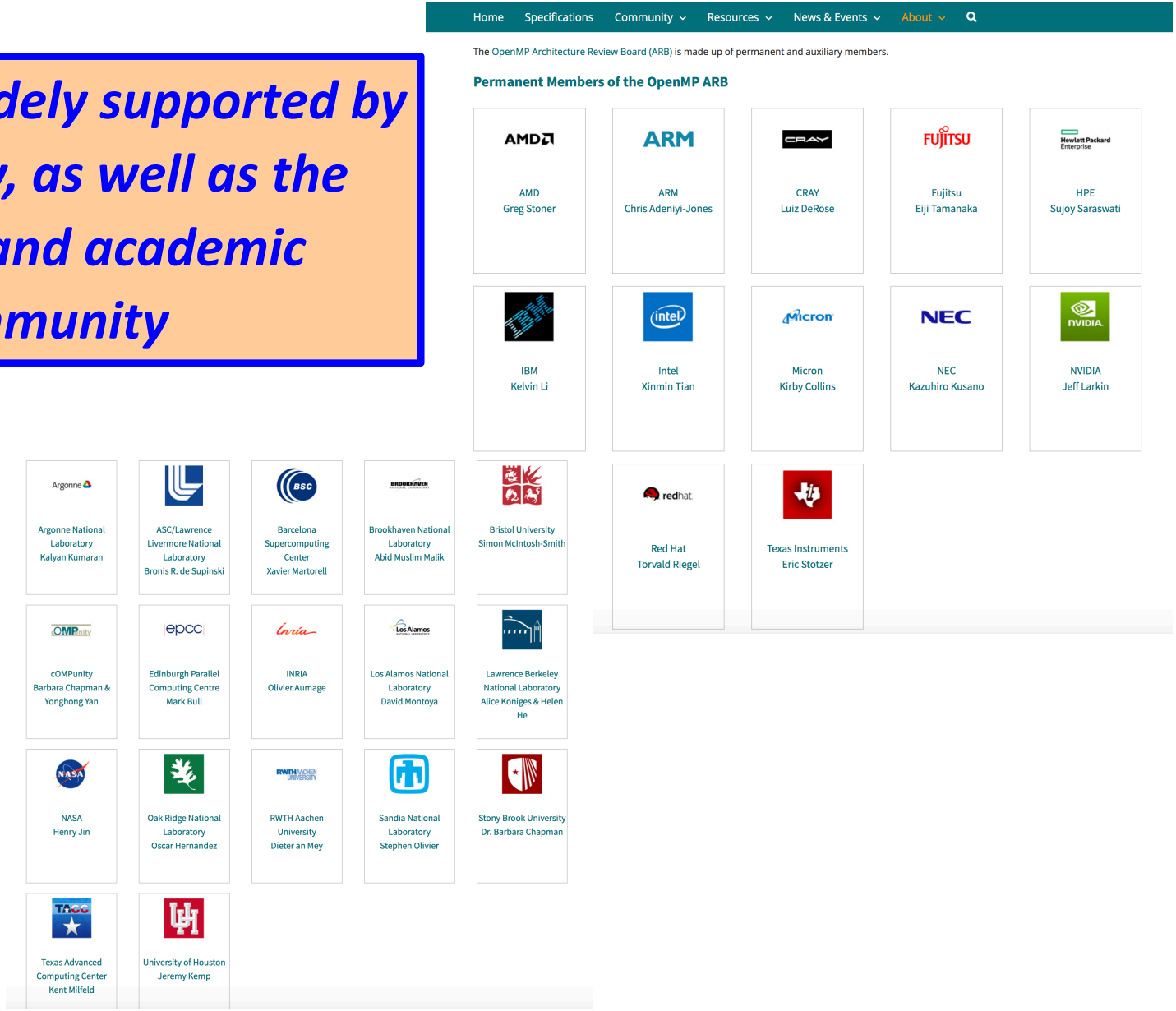
Lightweight

Mature, yet continues to evolve

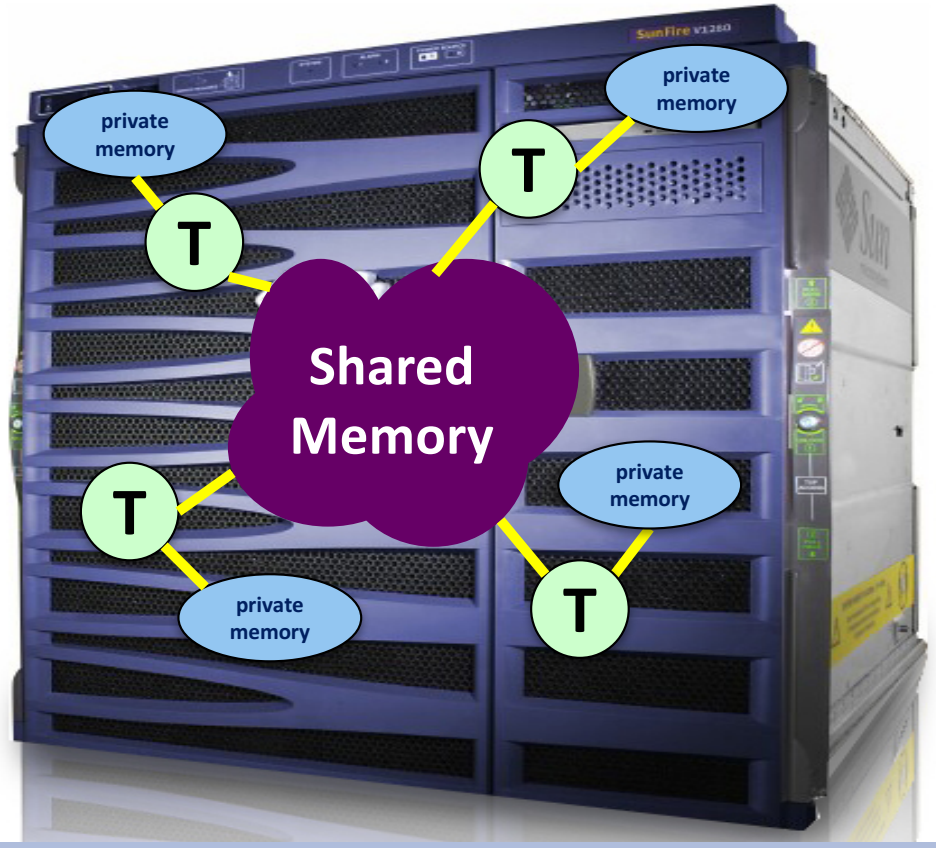
Widespread support and still on the rise

Support for OpenMP still growing

OpenMP is widely supported by the industry, as well as the research and academic community



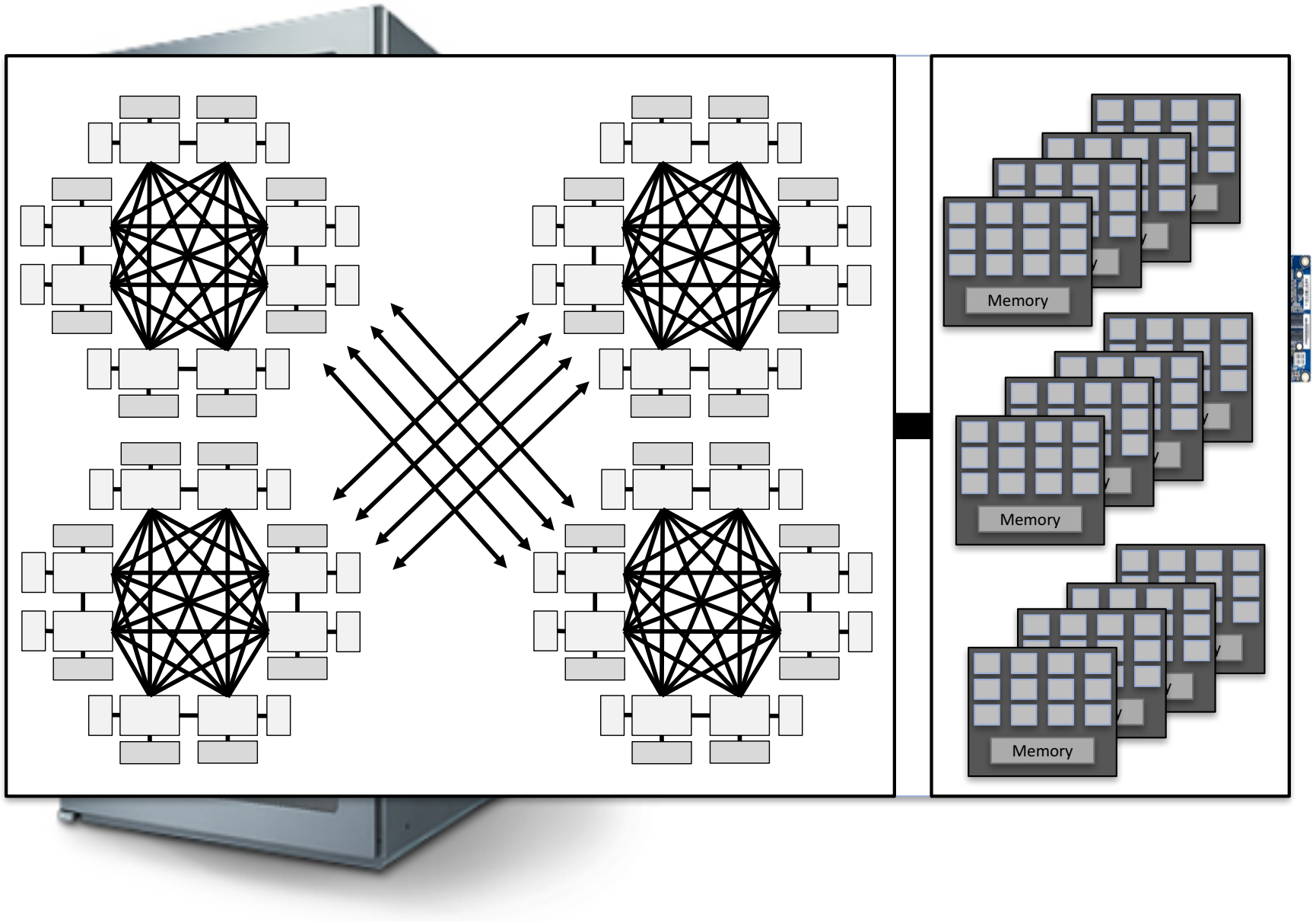
1997 - 2013



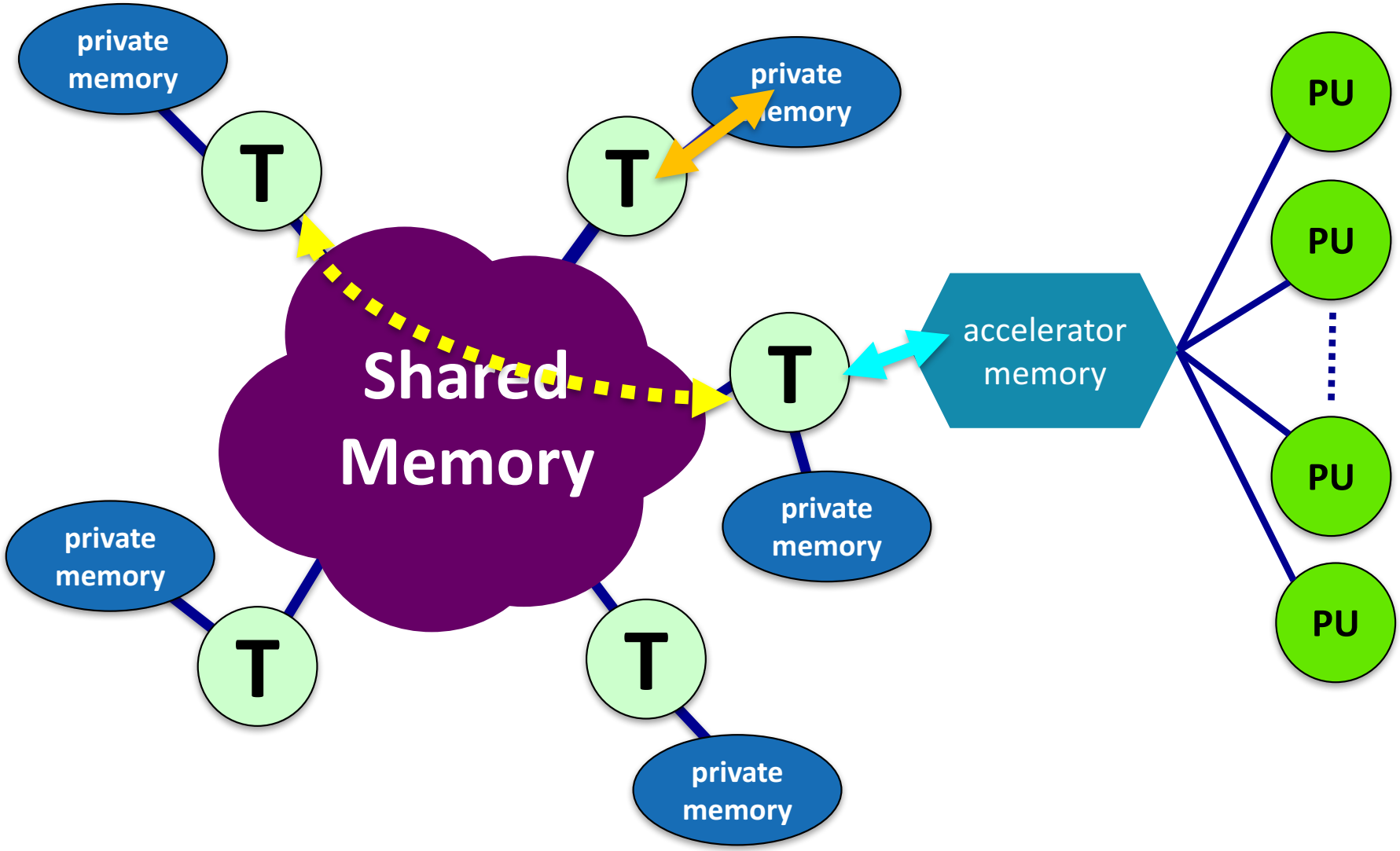
Small number of threads

Single and flat shared memory

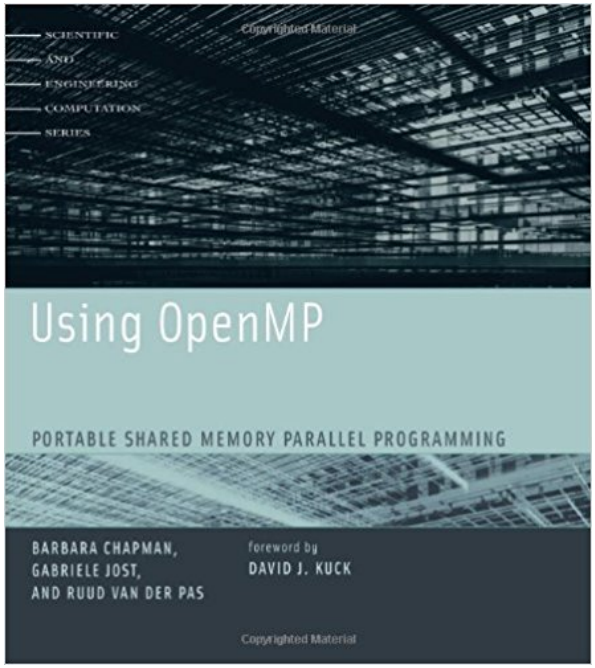
> July 2013



The OpenMP Memory Model



"Getting Started"



Just Out !
Covers all of 4.5 !

OpenMP And Performance

About OpenMP And Performance

Can Get Good Performance And Scalability

If You Do Things Right

Easy =/= Stupid

The Basics For All Users

Do Not Parallelize What Does Not Matter
(never tune a code without profile)

Do Not Share Memory Unless You Have To
(exploit private data as much as you can)

One "Parallel For Is Fine", More Is Evil
(merge them where you can)

Think BIG
(maximize the size of the parallel regions)

The Basics (Yes, Also For You)

Every Barrier Matters
(but no more than necessary, please)

Do Not Lock Yourself Out
(use atomic constructs where possible)

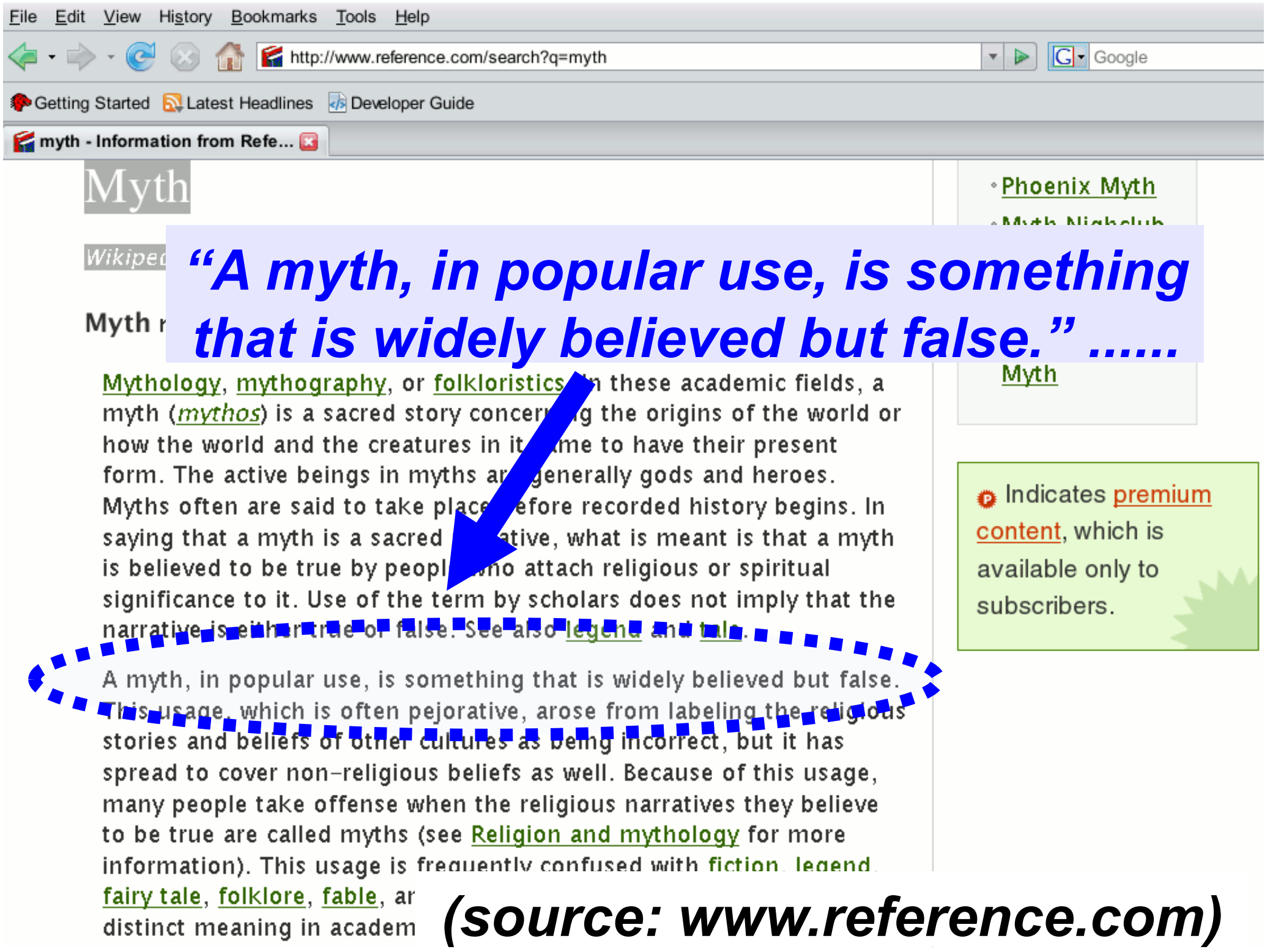
Everything Matters
(minor bottlenecks add up too)

The Basics (Yes, This Too)

Optimize For Memory Access
(about all systems are cc-NUMA)

Do Not Rule Out False Sharing
(may explain what you can't explain)

The Myth


File Edit View History Bookmarks Tools Help
http://www.reference.com/search?q=myth
Google
Getting Started
Latest Headlines
Developer Guide
myth - Information from Refe...
Myth
Wikiped
Myth r
Mythology, mythography, or folkloristics. In these academic fields, a myth (mythos) is a sacred story concer g the origins of the world or how the world and the creatures in it me to have their present form. The active beings in myths ar generally gods and heroes. Myths often are said to take plac efore recorded history begins. In saying that a myth is a sacred tive, what is meant is that a myth is believed to be true by peopl ho 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, ar
distinct meaning in academ
Phoenix Myth
Myth
Indicates premium content, which is available only to subscribers.
"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

Some Questions I Could Ask

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

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

Some More 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 used all the cores in the system."

"Did you estimate the expected speed up, using Amdahl's Law ?"

Even More Questions I Could Ask

"No, Amdahl's Law is not another EU financial bailout program. It is something else."

"Oh, you just think it should scale and used all the cores."

"I understand. You can't know everything."

Patience Is A Virtue

"Have you at least used a tool to identify the most time consuming parts in the program ?"

"Oh, you didn't and just parallelized all loops in the program."

"Did you try to avoid parallelizing the innermost loop in a loop nest ?"

There Are Limits To Anything

"Oh, you didn't. Did you at least minimize the number of parallel regions then ?"

"Oh, you didn't. It just worked fine the way it was."

"Sure."

Does It Ever End ?

"Did you at least use the nowait clause to reduce the impact of the barriers ?"

"Oh, never heard of a barrier. May be worth to read up on."

"Do all threads roughly perform the same amount of work ?"

"Oh, you don't know, but think it is fine. Let's hope you're right."

I Don’t Give Up That Easily

“Did you make optimal use of private data, or shared data in a rather liberal way ?”

“Oh, you didn’t. Sharing is just easier. I see.”

I Hate Giving Up

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

"Oh, never heard of that either. How unfortunate."

"Could false sharing perhaps impact the scalability ?"

"Oh, never heard of that either. May come handy to learn a little more about both."

The Grass Is Always Greener

"Clearly, you're unhappy with the performance. So, what did you do next ?"

"Switched to MPI. I see. Does that perform any better then?"

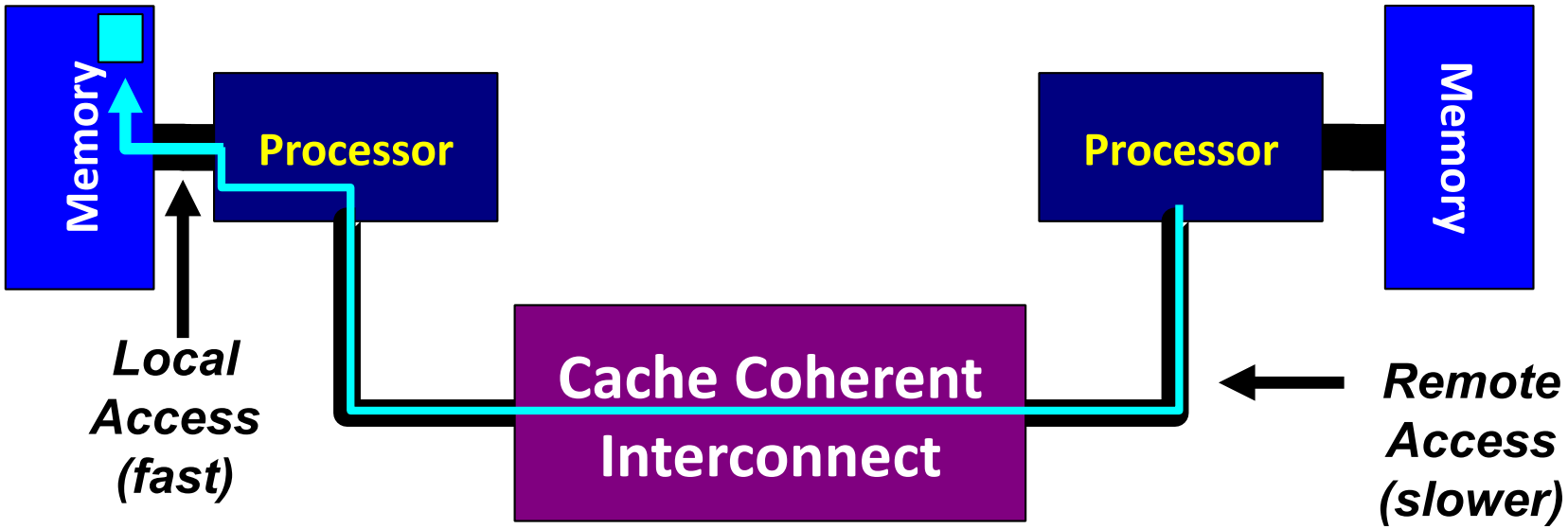
"Oh, you don't know. You're still debugging the code."

Changing To Pedantic Mode

"While waiting for your 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."

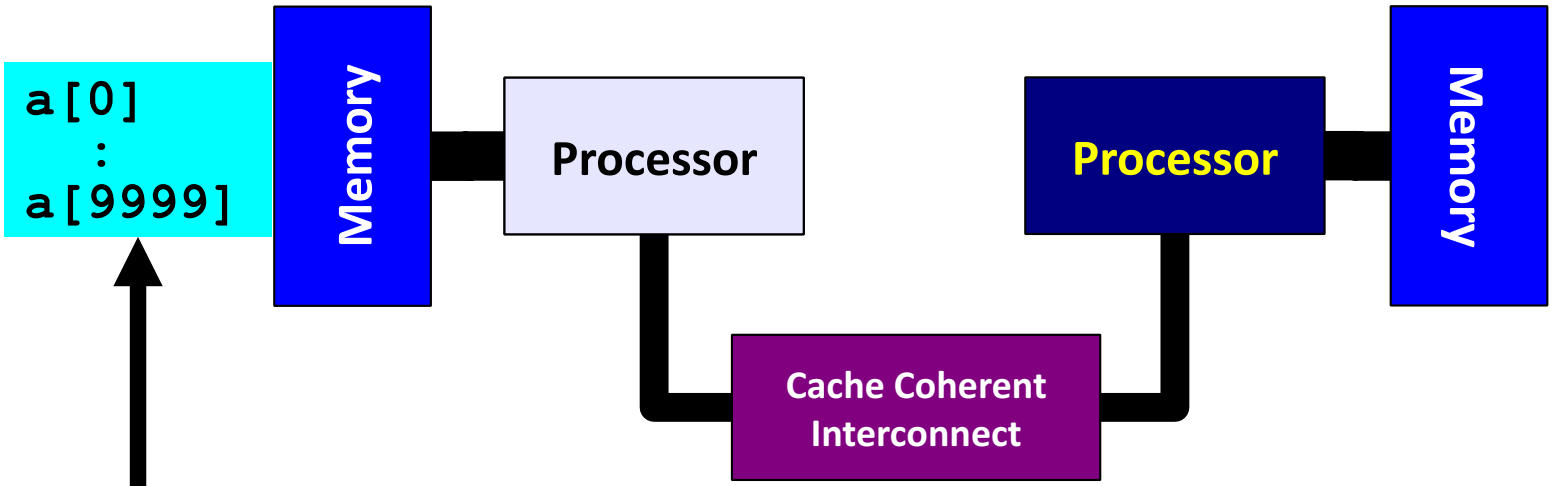
Considerations for cc-NUMA

A Generic cc-NUMA Architecture



Main Issue: How To Distribute The Data ?

Example First Touch Placement/1



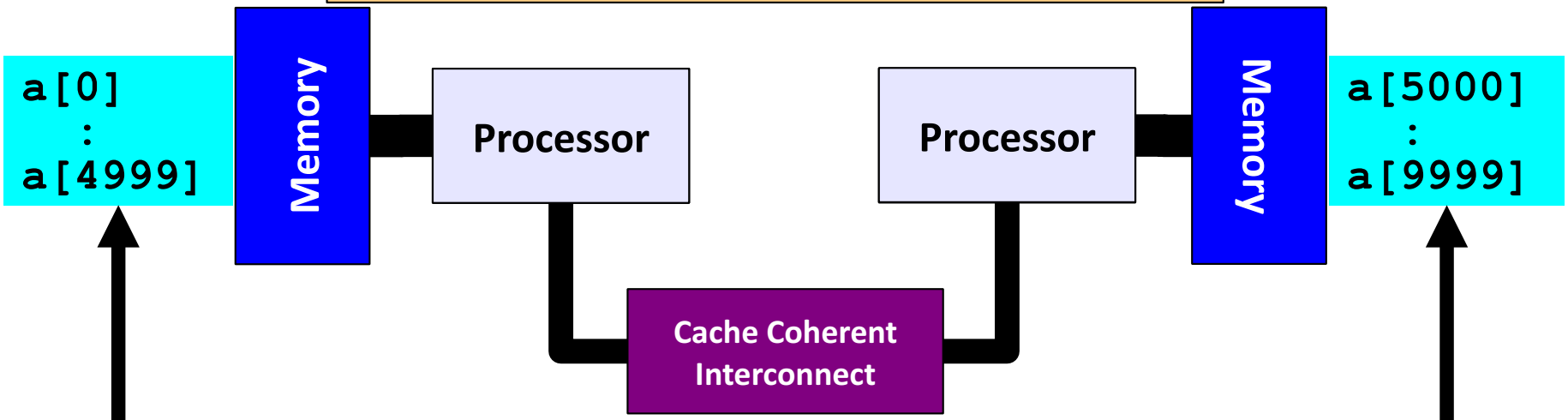

```
for (i=0; i<10000; i++)
    a[i] = 0;
```

First Touch

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

Example First Touch Placement/2

```
$ export OMP_PLACES=sockets
$ export OMP_PROC_BIND=spread
```




```
#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

"Lock Yourself Out"

What Is A Lock ?

A lock is "something" acquired by a thread

While it has the lock, nobody else gets it

This allows the thread to do its work in private, not bothered by other threads

Once finished, it has to release the lock

And allow another thread to do its private business

A Real-World Example

Everybody wants to steal a cookie from the jar

While someone selects a cookie, nobody else is allowed to steal a cookie as well

The Code

```
#include <omp.h>

omp_lock_t myLock;

int main(….)
{
   (void) omp_init_lock(&myLock);

   #pragma omp parallel
   {
       (void) TheCookieJar();

   } // End of parallel region

   (void) omp_destroy_lock(&myLock);
}
```

Inside The Cookie Jar

```
void TheCookieJar()
{

   (void) omp_set_lock(&myLock);     // acquire lock

    (void) grabMyFavouriteCookie(); // get cookie

   (void) omp_unset_lock(&myLock);   // release lock

}
```

Why Bother About Locks ?

In OpenMP, a lock is a special variable

And lives in memory

To check availability, a thread has to read it

That means data has to travel to the thread

This takes some time, or a lot of time

So What ?

A lock serializes execution

The bigger the system, the worse it gets

But don't worry, small systems suffer too

Sooner than later, a lock destroys scalability

The Evil Family Member

```
#include <omp.h>

int64_t myCounter = 0;

int main(….)
{

   #pragma omp parallel
   {
      #pragma omp atomic update
         myCounter++;

   } // End of parallel region

}
```

Deep Inside The Dark Dungeon

(void) my_atomic_add (&a,1);

This is the code in "fake assembly"*

```
   ld    [%o0],%r1       ! Load: Copy value from address in o0 into r1
again:
   add   %r1,1,%r2       ! Compute new value: %r2 = %r1 + 1
   cas   [%o0],%r1,%r2   ! CAS: if [%o0] == %r1 then swap [%o0] and %r2
   cmp   %r1,%r2         ! Compare: if %r2 == %r1, CAS succeeded, values
                         ! are swapped and [%o0] has the new value
   bne   %icc, again     ! Branch: CAS failed, try again

   <register management and return>
```

**) And a very naive implementation*

Why Is It Evil ?

Atomic operations are potentially evil too

Despite instruction level support (e.g. “cas”)

A big part of the cost is in the data transfer

Sooner than later, an atomic operation destroys scalability

The Code

```
#pragma omp parallel shared(job)
{
    int64_t i;
    while(1) {
      #pragma omp atomic capture
      {
         i = job->counter;
         job->counter++;
      }
      if ( i >= job->end ) break;

      do_the_work(i);

    } // End of while-loop
} // End of parallel region
```

Good for load balancing

Bad for scalability

Main Idea

This Can Be Transformed Into A Loop

The Modified Code – Set Up

```
int64_t Nthreads                = … // Number of threads
int64_t total_chunks_of_work    = … // Total size of work
int64_t chunk_size_per_thread = … // Work per thread

for(int k=0; k < Nthreads; k++)
{
    thread_work_chunks_start[k] = …
    thread_work_chunks_end  [k] = …
}
```

The Modified Code – Core Part/1

```
#pragma omp parallel
{

  omp_set_lock(&my_lock);
    if ( index_work >= Nthreads ) {
       done = true;
    } else {
      start_loop = thread_work_chunks_start[index_work];
      end_loop   = thread_work_chunks_end  [index_work];
      index_work++;
    }
  omp_unset_lock(&my_lock);

    <the computational part of the code is on the next slide>

} // End of parallel region
```

The Modified Code – Core Part/2

```
if (!done) {

  for (chunk=start_loop; chunk <= end_loop; chunk++)
  {

      do_the_work(chunk);

  } // End of loop over chunks

}
```

Worse for load balancing

Very good for scalability

Summary

Locks And Atomic Operations Are Like Gold Coins

You Like To Have Them, But Not Use Them

"~~Second Hand~~ Pre-Owned Memory"

Motivation Of This Work

Question: "Why Do You Rob Banks ?"

Answer: "Because That's Where The Money Is"

Willie Sutton – Bank Robber, 1952

Question: "Why Do You Focus On Memory ?"

Answer: "Because That's Where The Bottleneck Is"

Ruud van der Pas – Performance Geek, 2018

The New Challenge

Modern systems have:

- ***Many hardware threads***
- ***Large to very large memories***
- ***A cc-NUMA architecture***

In order to get scalable performance, the memory has to be used in the right way

A Sparse Matrix Summation

```
for (int i=0; i<N; i++)
    sum += data[index[i]];
```

Array "index" is 5-6 GB

Array "data" is much smaller (~300 MB)

Random Memory Access

- Because of the “index” array, memory access to “data” jumps all over the place
- For example if **index[0] = 10** and **index[1] = 5678**:

```
sum += data[10];   // i = 0
sum += data[5678]; // i = 1
```

Distance in memory:

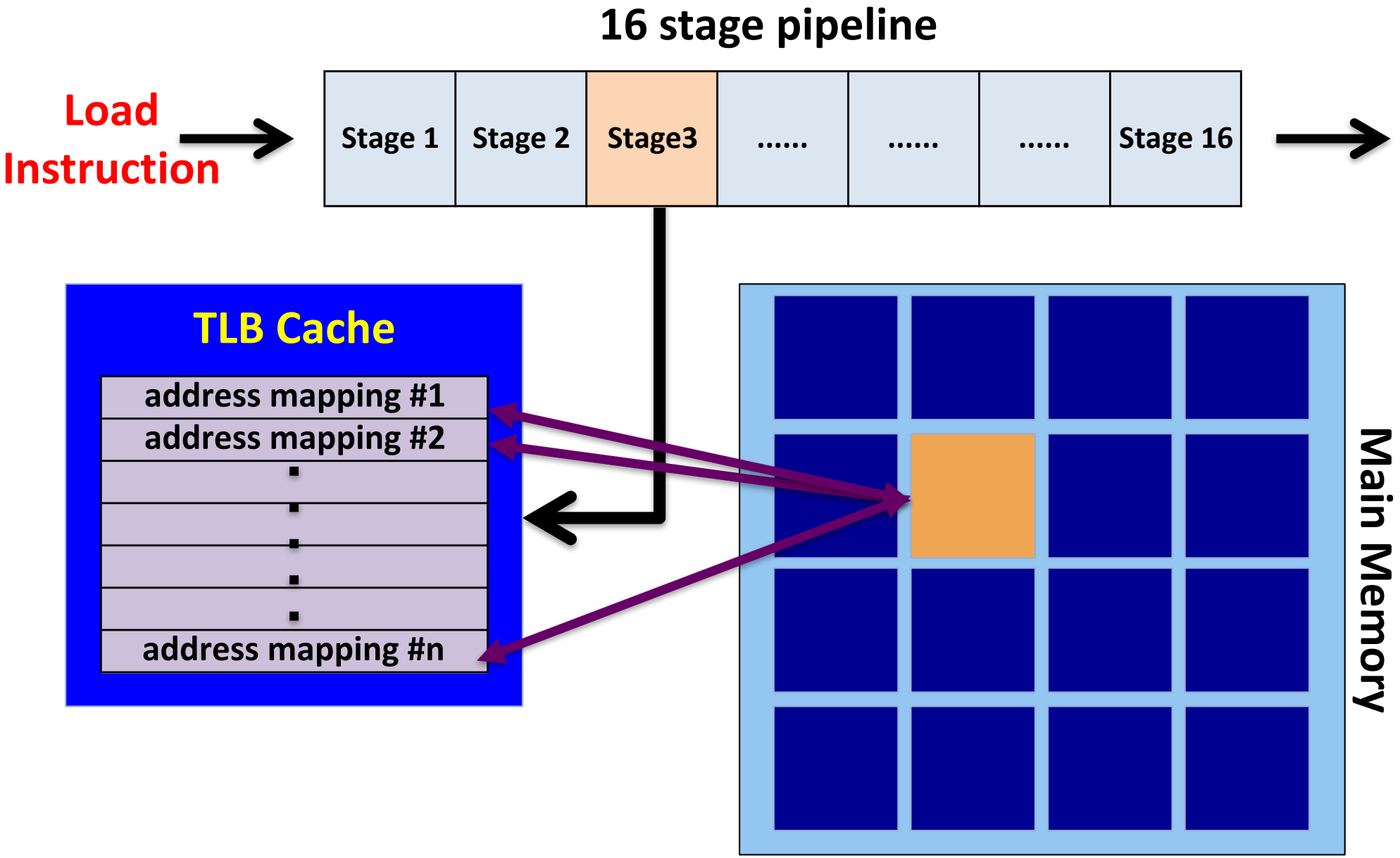
(5678-10+1)*8 = 5669*8 = 45352 bytes

Translation Lookaside Buffer (TLB)

- A load/store instruction uses a **virtual address**
- This address needs to be **translated**

  → Virtual Address (app) -> Physical Address (in memory)
- This is expensive and so information is **cached**
- This cache is called a **Translation Lookaside Buffer**, or "**TLB**" for short

Addresses - How Things Work



About The TLB/1

The requested element is part of a page

The page has to be mapped into the TLB

If this is not the case, tough luck

Need to wait for the mapping

This Is EXPENSIVE

About The TLB/2

Mapping capacity = #Entries * Size Of Page

Can't change the number of entries

Can change the page size

"Use Large Pages"

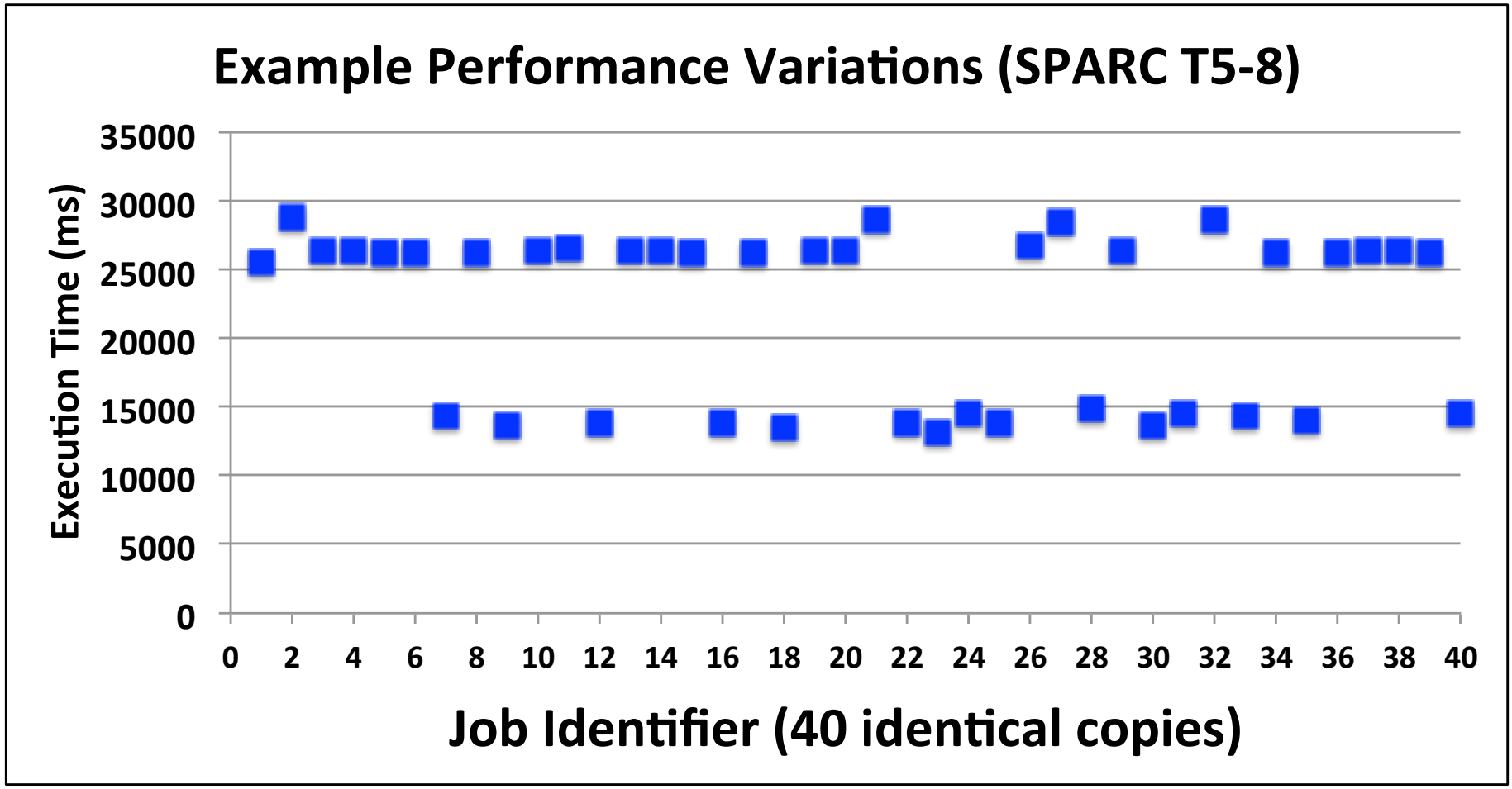
When Can Large Pages Help ?

If memory requirements are large

Data is accessed randomly

Observation – C++ Application

Seemingly random but discrete performance variations



The Analysis

- Using DTrace and other tools, it was found that:
  - → This C++ application does **substantial memory management** before entering the core part
  - → Memory is allocated and released during this initial phase
- Memory allocators try to re-use released memory

Where Is This Memory Placed ?

In Other Words

```
array = (double *) malloc(N * sizeof(double));
```

```
#pragma omp parallel for
   for (int i=0; i<N; i++)
      array[i] = 0.0;
```

What If "First Touch" Is Really "Second Hand Touch" ?

Recall The Algorithm

```
for (int i=0; i<N; i++)
   sum += data[index[i]];
```

"index" is ~5-6 GB and "data" ~300 MB

Maybe we should use Large Pages

But the difference in size is an issue

Example

```
array = (double *) malloc(N * sizeof(double));
```

```
(void) memcntl(array,...,"POLICY",...);
```

```
#pragma omp parallel for
   for (int i=0; i<N; i++)
      array[i] = 0.0;
```

Use "POLICY" to specify page placement method

"What You Write Is What You Get"

What Is A Graph ?

A graph describes relations and consists of two sets

The vertices "V" and the edges "E"

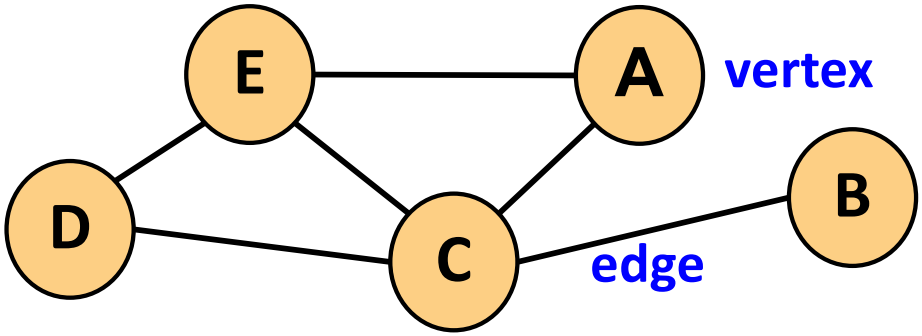
The vertices "V" are the objects of interest (e.g. people)

The edges "E" are defined by the relationship

Example: "Do these two persons know each other ?"

An important graph operation is a search for connections

Example Of A Graph



V(G) := { A, B, C, D, E }

E{G} := { (A,C), (A,E), (B,C), (C,E), (C,D), (D,E) }

Example of a search: "Are A and D directly connected ?"

The Graph 500 Benchmark

Conducts 64 ”Breadth First Searches” (BFS) in a graph

The “scale” parameter controls the size of the graph:

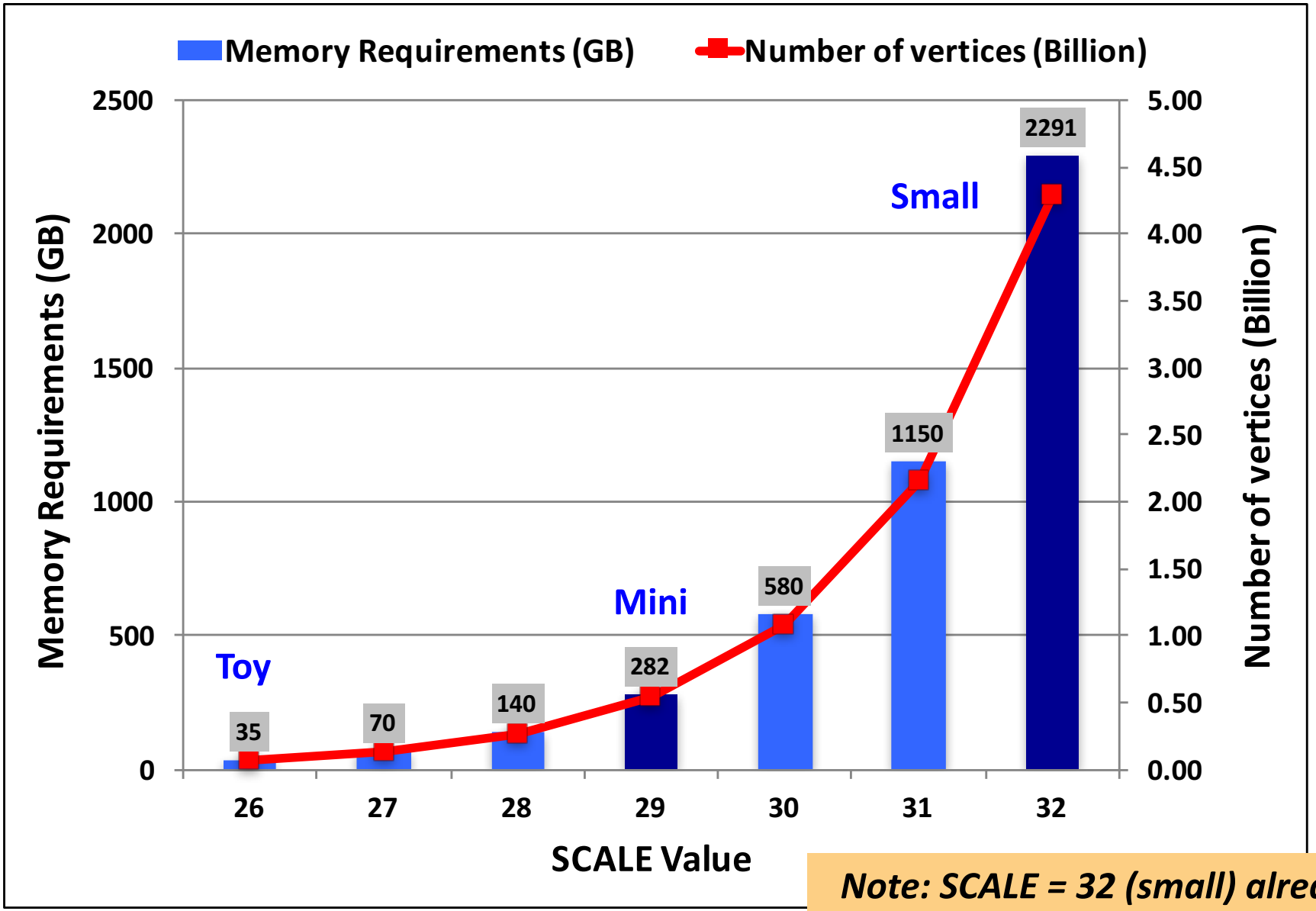
- *Number of vertices is 2^scale*
- *Number of edges is 16 per vertex*

The benchmark score is given in “Giga Traversed Edges Per Second”, or GTEPS for short

This is a speed and the equivalent of GFLOPS

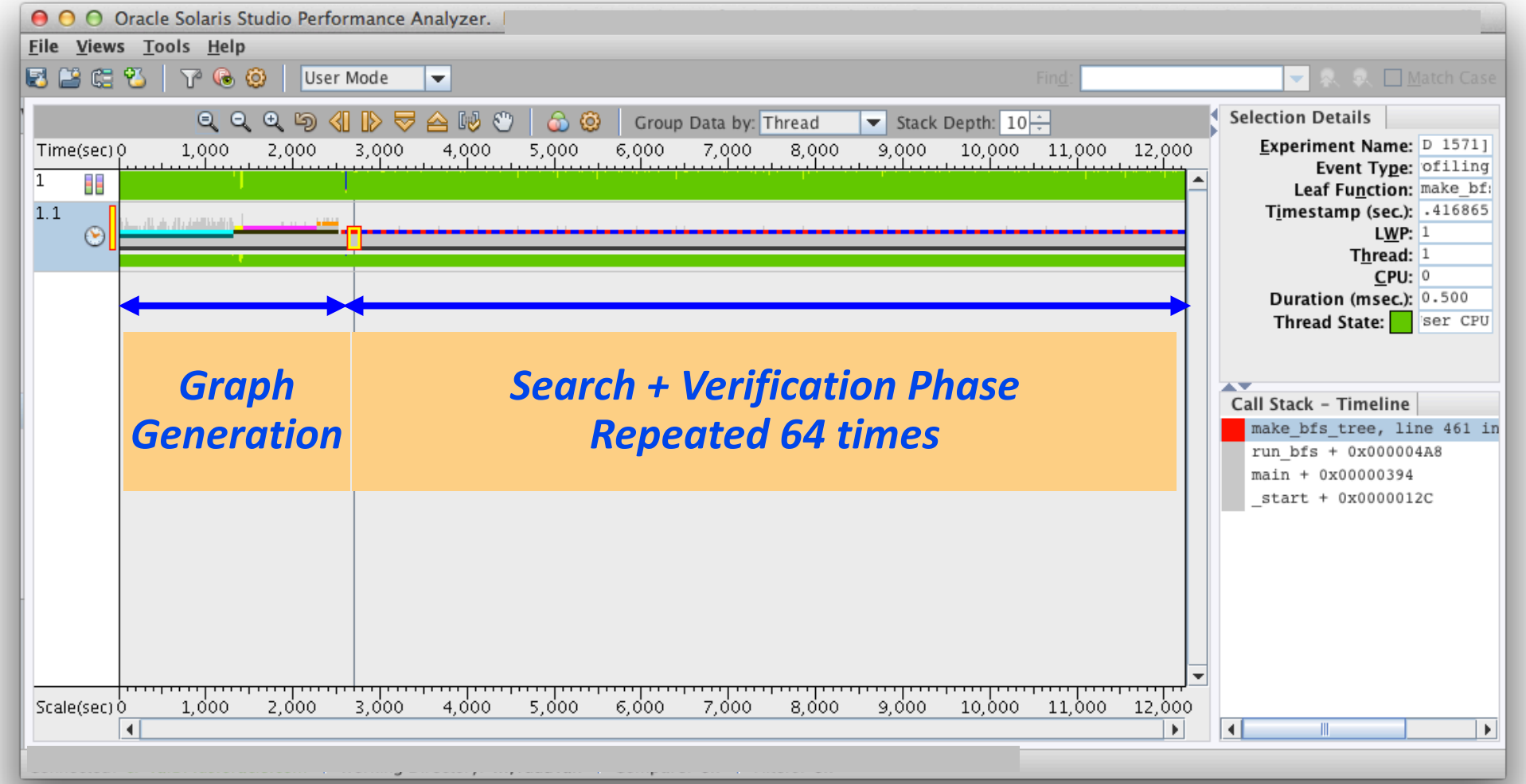
More information at http://www.graph500.org

Graph 500 Resource Requirements



Note: SCALE = 32 (small) already requires 2.3 TB of memory

BFS Graph 500 - Execution Pattern



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

Why Is This Bad For Performance ?

```
#pragma omp parallel for
   for (…)   // Loop 1
#pragma omp parallel for
   for (…)   // Loop 2
#pragma omp pa
   for (…)   //
#pragma omp pa
   for (…)   //
```

There is a barrier for each loop
And you can't get rid of it
Do you know why ?

Each parallel region has overhead, making things worse

The Solution

```
#pragma omp pa
{
   #pragma omp
    for (…)   //
   #pragma omp for [nowait]
    for (…)   // Loop 2
   #pragma omp for [nowait]
    for (…)   // Loop 3
   #pragma omp for nowait
    for (…)   // Loop 4
} // End of parallel region
```

Opportunity for "nowait", at least one barrier less anyhow, and much less parallel overhead

Some Optimizations Performed

Removed OpenMP inefficiencies
Example: redundant barriers
Maximize the size of parallel regions
Brute Force: Serialized a parallel part ….
Exploit "First Touch"

How To Exploit "First Touch" ?

```
my_pointer = (int64_t *) malloc(n*sizeof(64_t));

if (my_pointer == NULL) {<bailout>}

#pragma omp parallel for shared(n,my_pointer)
  for (int64_t i=0 ; i<n; i++)
     my_pointer[i] = 0;
```

Focus on large data allocations

After the allocation, initialize in parallel

Do this as soon as you can

Common Reasons For Such Gaps

- **Load imbalance**
  - → Not all threads need the same amount of time
  - → Threads that finish early, wait in the barrier
- **Only the master thread executes**
  - → Either we're outside of a parallel region, or
  - → We're inside a "single region" within a parallel region

Summary – For The Patient

Think Ahead

Never Forget To Tune For Serial Performance

Really Think About Your Datastructures

Seriously Consider Data Placement

Minimize Communication

Order Sufficient Secret Sauce
(consider a subscription model)