OpenMP: a defacto standard for high-level Parallel Computing on shared memory and more

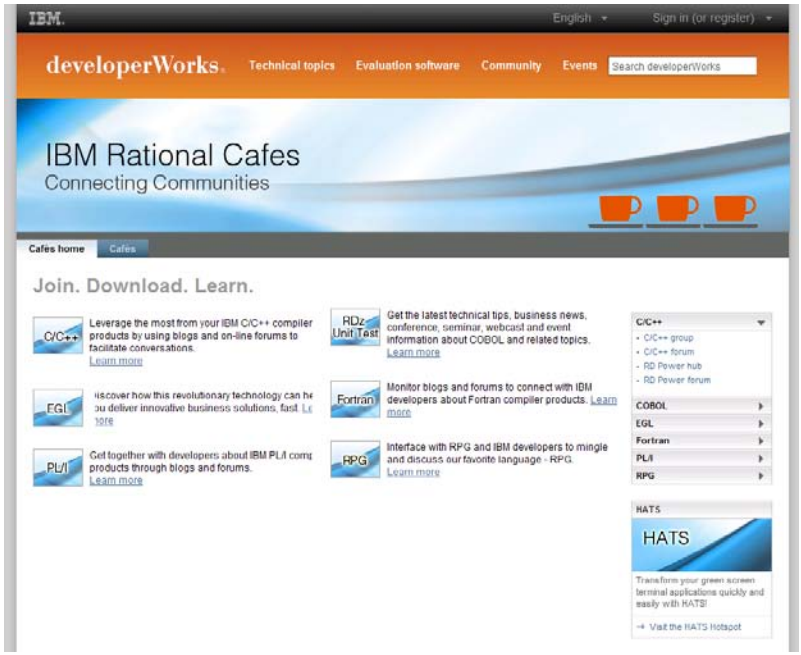
Michael Wong
michaelw@ca.ibm.com
IBM, Canada C++ Standard
Chair of WG21 SG5 Transactional Memory
OpenMP CEO
IBM xlC compiler Senior Technical Lead

IBM Rational Disclaimer

IBM Rational Cafes – Connecting Communities

- Accelerate your enterprise modernization efforts by becoming a member of the Cafe communities
- Ask questions, get free distance learning, browse the resources, attend user group webcasts, read the blogs, download trials, and share with others
- Cafes have forums, blogs, wikis, and more

- **Languages covered:**
  C/C++, COBOL, Fortran, EGL, PL/I, and RPG
- **Products covered:**
  - COBOL for AIX®
  - Enterprise COBOL for z/OS®
  - Enterprise PL/I for z/OS
  - Host Access Transformation Services
  - PL/I for AIX
  - Rational® Business Developer
  - Rational Developer for Power Systems Software™
  - Rational Developer for i for SOA Construction
  - Rational Developer for System z
  - Rational Developer for System z Unit Test
  - Rational Team Concert™
  - XL C for AIX
  - XL C/C++ for AIX/Linux ®
  - XL Fortran for AIX/Linux
  - XL C/C++ for z/VM
  - z/OS XL C/C++



Become a member and join the conversation!
ibm.com/rational/café

Continue the conversation

facebook.com/IBMcompilers

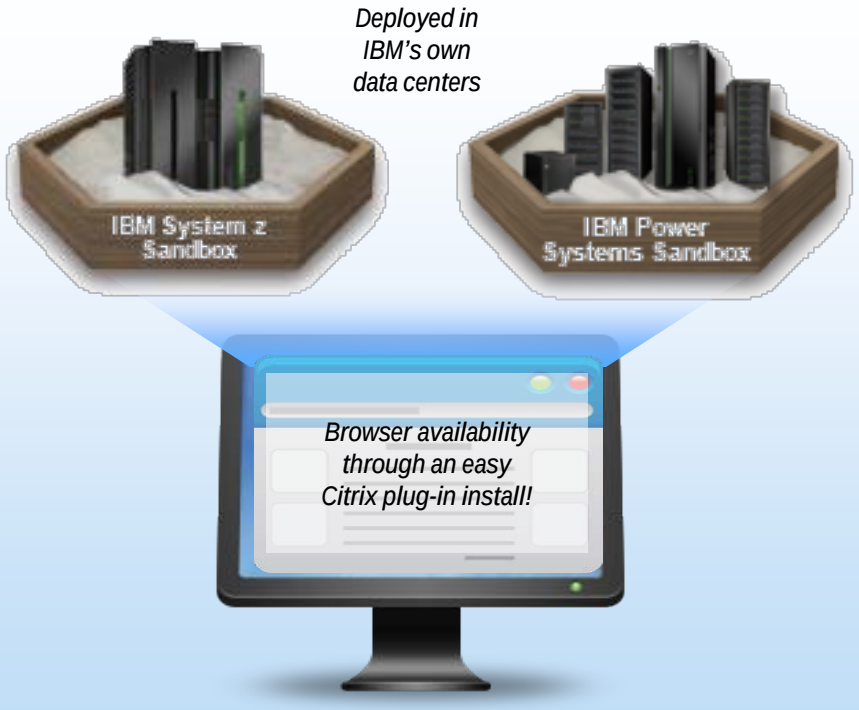
@IBM_Compilers

xl_compiler@ca.ibm.com

IBM Enterprise Modernization Sandbox

Realize the value of your investments in assets, skills and infrastructure within minutes

- Learn how to revitalize applications, empower people, unify teams and exploit infrastructure based on your knowledge and experiences
- See firsthand the tangible value Rational tools can bring to your business
- Get a fast start with scripted scenarios and best practice education materials at no cost available 24x7
- Access a low risk way to try several new offerings and integrated solutions without disturbing your existing environment

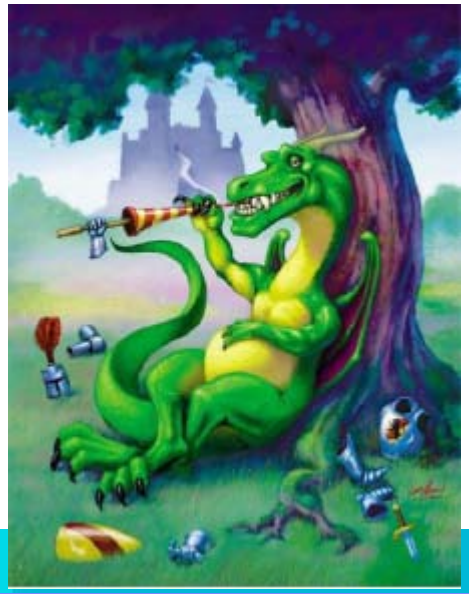


Try it firsthand within minutes today!

ibm.com/playinthesandbox

Agenda

- Is it legal to do tasks in C++ today?
- The OpenMP ARB
- History of OpenMP
- Intro tutorial to OpenMP
- Bonus 1: Towards OpenMP 4.0
- Fragen?



Hello Concurrent World

```
#include <iostream>
#include <thread> //#1
void hello() //#2
{
   std::cout<<"Hello Concurrent World"<<std::endl;
}
int main()
{
   std::thread t(hello); //#3
   t.join(); //#4
}
```

Is this valid C++ today? Are these equivalent?

```
int x = 0;
atomic<int> y = 0;
Thread 1:
  x = 17;
  y.store(1,
  memory_order_release);
  // or:       y.store(1);

Thread 2:
  while
  (y.load(memory_order_acqu
  ire) != 1)
  // or:       while
  (y.load() != 1)
  assert(x == 17);
```

```
int x = 0;
atomic<int> y = 0;
Thread 1:
  x = 17;
  y = 1;
Thread 2:
  while (y != 1)
        continue;
  assert(x == 17);
```

Hello World again

- What will this program print?

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
        printf("Hello ");
        printf("World ");
        printf("\n");
        return(0);
}
```

2-threaded Hello World with OpenMP threads

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
   #pragma omp parallel
   {
      printf("Hello ");
      printf("World ");
   } // End of parallel region
   printf("\n");
   return(0);
}
```

Hello World Hello World

Or

Hello Hello World World

More advanced 2-threaded Hello World

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    #pragma omp parallel
    {
        #pragma omp single
        {
            printf("Hello ");
            printf("World ");
        }
    } // End of parallel region
    printf("\n");
    return(0);
}
```

```
Hello World
```

Hello World with OpenMP tasks now run 3 times

```
int main(int argc, char *argv[]) {
        #pragma omp parallel
        {
                #pragma omp single
                {
                        #pragma omp task
                                {printf("Hello ");}
                        #pragma omp task
                                {printf("World ");}
                        }
                } // End of parallel region
        printf("\n");
        return(0);
}
```

```
Hello World
Hello World
World Hello
```

Tasks are executed at a task execution point

```
int main(int argc, char *argv[]) {
        #pragma omp parallel
        {
                #pragma omp single
                {
                        #pragma omp task
                                {printf("Hello ");}
                        #pragma omp task
                                {printf("World ");}
                        printf("\nThank You ");
                }
                } // End of parallel region
        printf("\n");
        return(0);
}
```

```
Thank You Hello World
Thank You Hello World
Thank You World Hello
```

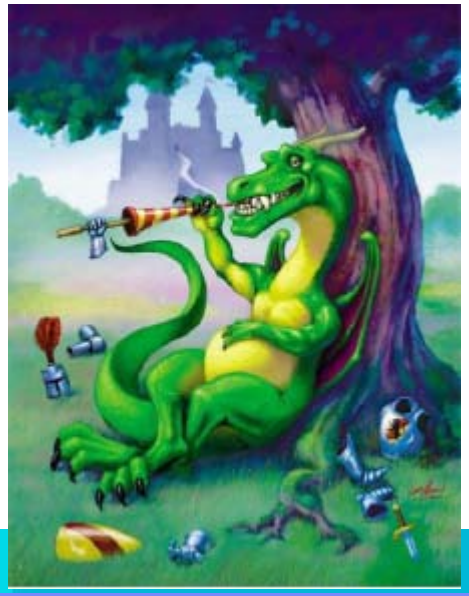
Execute Tasks First

```
int main(int argc, char *argv[]) {
        #pragma omp parallel
        {
                #pragma omp single
                {
                        #pragma omp task
                                {printf("Hello ");}
                        #pragma omp task
                                {printf("World ");}
                        #pragma omp taskwait
                        printf(“Thank You “);
                }
        } // End of parallel region
        printf("\n");return(0);
}
```

```
Hello World Thank You
Hello World Thank You
World Hello Thank You
```

Agenda

- Is it legal to do tasks in C++ today?
- The OpenMP ARB
- History of OpenMP
- Intro tutorial to OpenMP
- Bonus 1: Towards OpenMP 4.0
- Fragen?



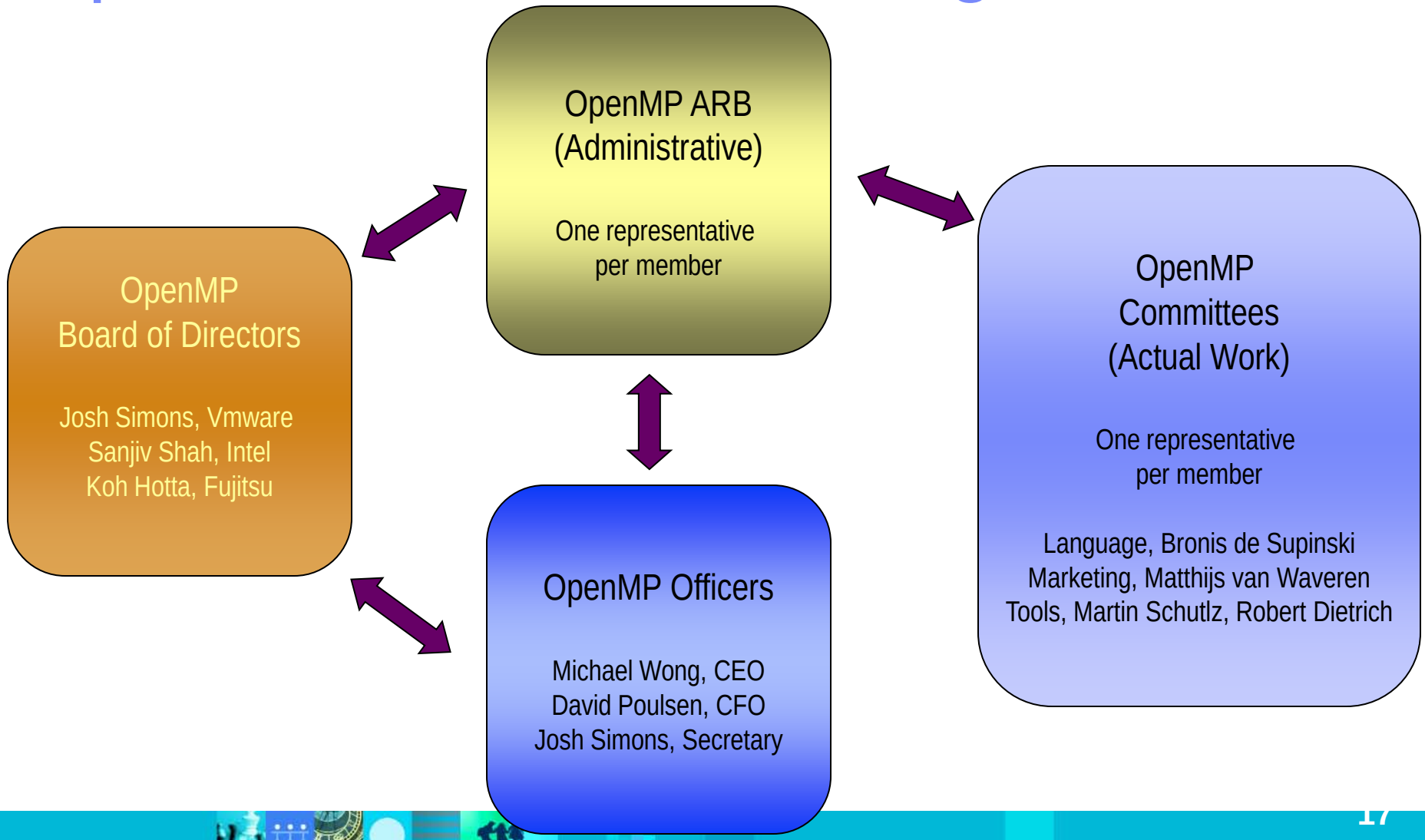
The OpenMP ARB

- The care of OpenMP is in the hands of the OpenMP Architecture Review Board (the ARB).
- The ARB:
  - Interprets OpenMP
  - Writes new specifications - keeps OpenMP relevant.
  - Works to increase the impact of OpenMP.
- Organizations join the ARB - not individuals
  - Current members
    - Permanent: AMD, Cray, Fujitsu, HP, IBM, Intel, Microsoft, NEC, SGI, Oracle/Sun, STMicroelectronics/PGI
    - Auxiliary: ASCI, cOMPunity, EPCC, LLNL, NASA, RWTH Aachen

New Members

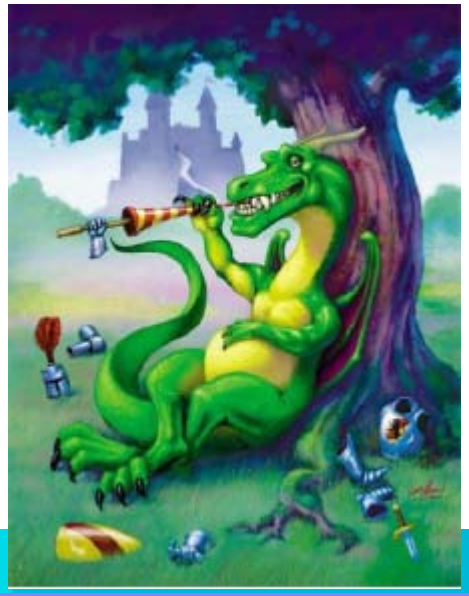
- Rapid growth in membership
  - NVIDIA – U.S.
  - Texas Instrument – U.S.
  - CAPS Enterprise – France
  - Argonne National Laboratory – U.S.
  - Oak Ridge National Laboratory – U.S.
  - Los Alamos National Laboratory – U.S.
  - Texas Advanced Computing Centre – U.S.
  - Convey Computers – U.S.

OpenMP ARB Current Organization

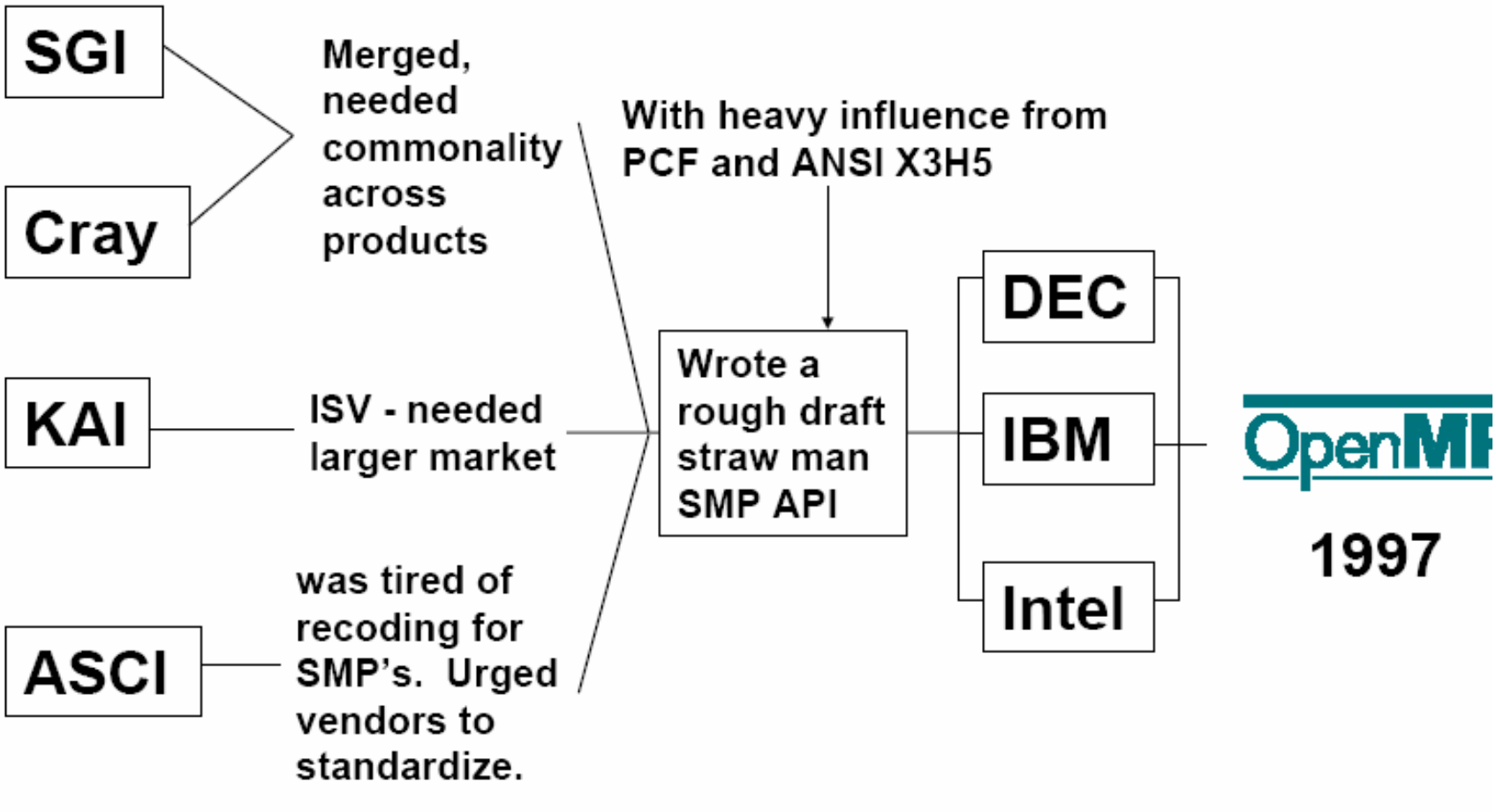


Agenda

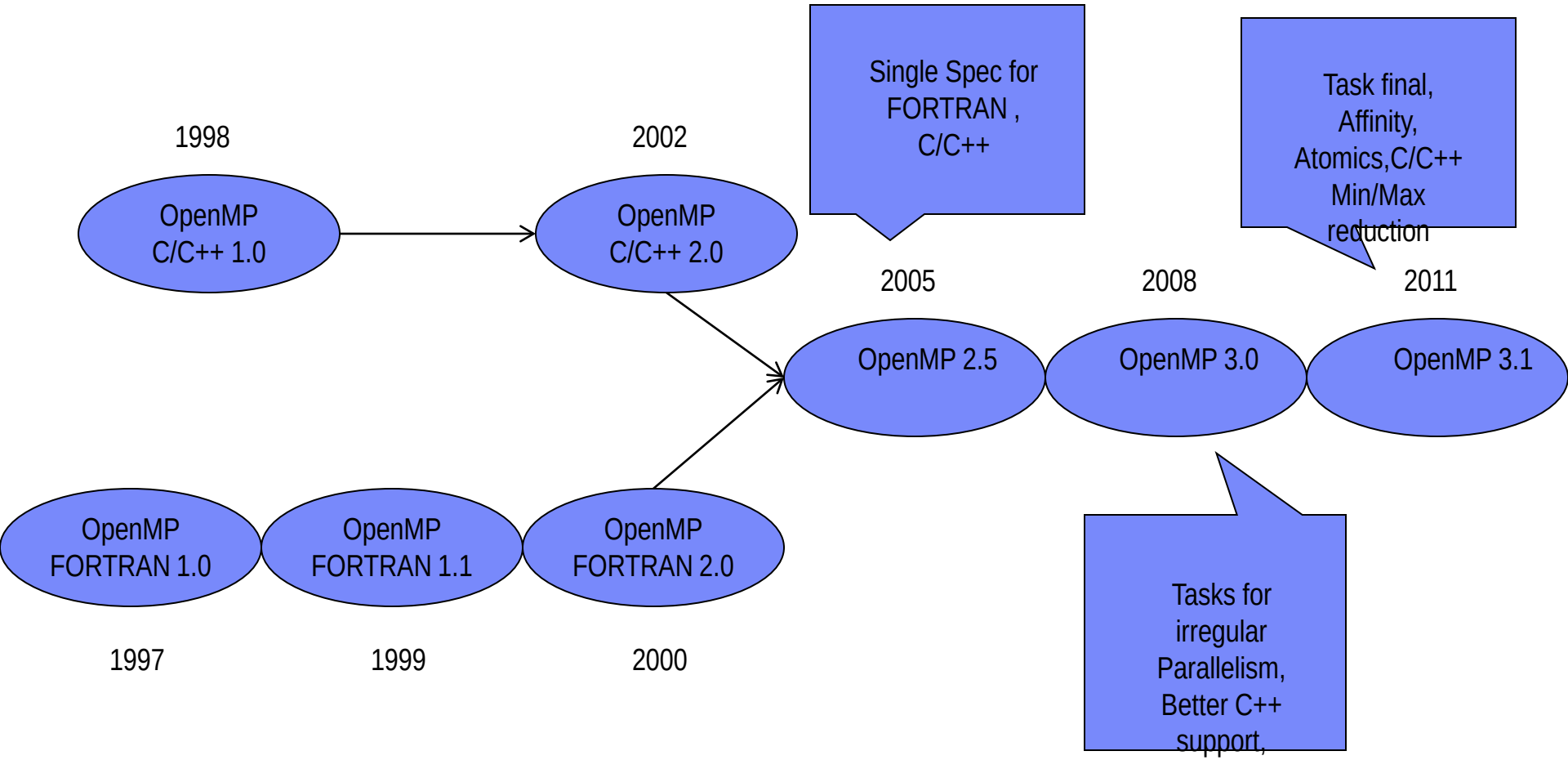
- Is it legal to do tasks in C++ today?
- The OpenMP ARB
- History of OpenMP
- Intro tutorial to OpenMP
- Bonus 1: Towards OpenMP 4.0
- Fragen?



Pre- 1997: Ancient times

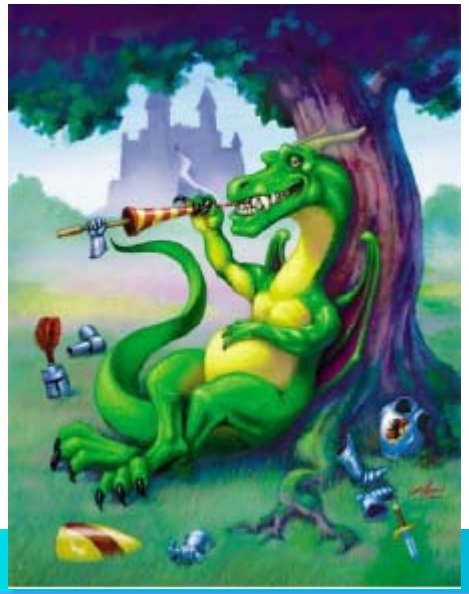


1997-2011: A New Era



Agenda

- Is it legal to do tasks in C++ today?
- The OpenMP ARB
- History of OpenMP
- Intro tutorial to OpenMP
- Bonus 1: Towards OpenMP 4.0
- Fragen?



Intro to OpenMP

- ***De-facto standard Application Programming Interface (API) to write shared memory parallel applications in C, C++, and Fortran***
- ***Consists of:***
  - ***Compiler directives***
  - ***Run time routines***
  - ***Environment variables***
- ***Specification maintained by the OpenMP***
- ***Architecture Review Board (http://www.openmp.org)***
  - ***Version 3.1 has been released 2011***

When do you want to use OpenMP?

- If the compiler cannot parallelize the way you like it even with auto-parallelization
  - a loop is not parallelized
    - Data dependency analyses is not able to determine whether it is safe to parallelize or not
  - Compiler finds a low level of parallelism
    - But your know there is a high level, but compiler lacks information to parallelize at the highest possible level
- No Auto-parallelizing compiler, then you have to do it yourself
  - Need explicit parallelization using directives

Coming in from the jungle

- It was a jungle out there, with lots of vendor directives, missing functionalities, can't port programs
  - Ours was called –qsmp
  - So if you just say –qsmp, then you get:
    - auto|opt, explicit, noomp, norec_locks, nonested_par, schedule=runtime
- Not a standard but a specification from a consortium, like MPI
- Fortran, C/C++, getting more support for C++

Advantages of OpenMP

- ***Good performance and scalability***
  - ***If you do it right ....***
- ***De-facto and mature standard***
- ***An OpenMP program is portable***
  - ***Supported by a large number of compilers***
- ***Requires little programming effort***
- ***Allows the program to be parallelized incrementally***

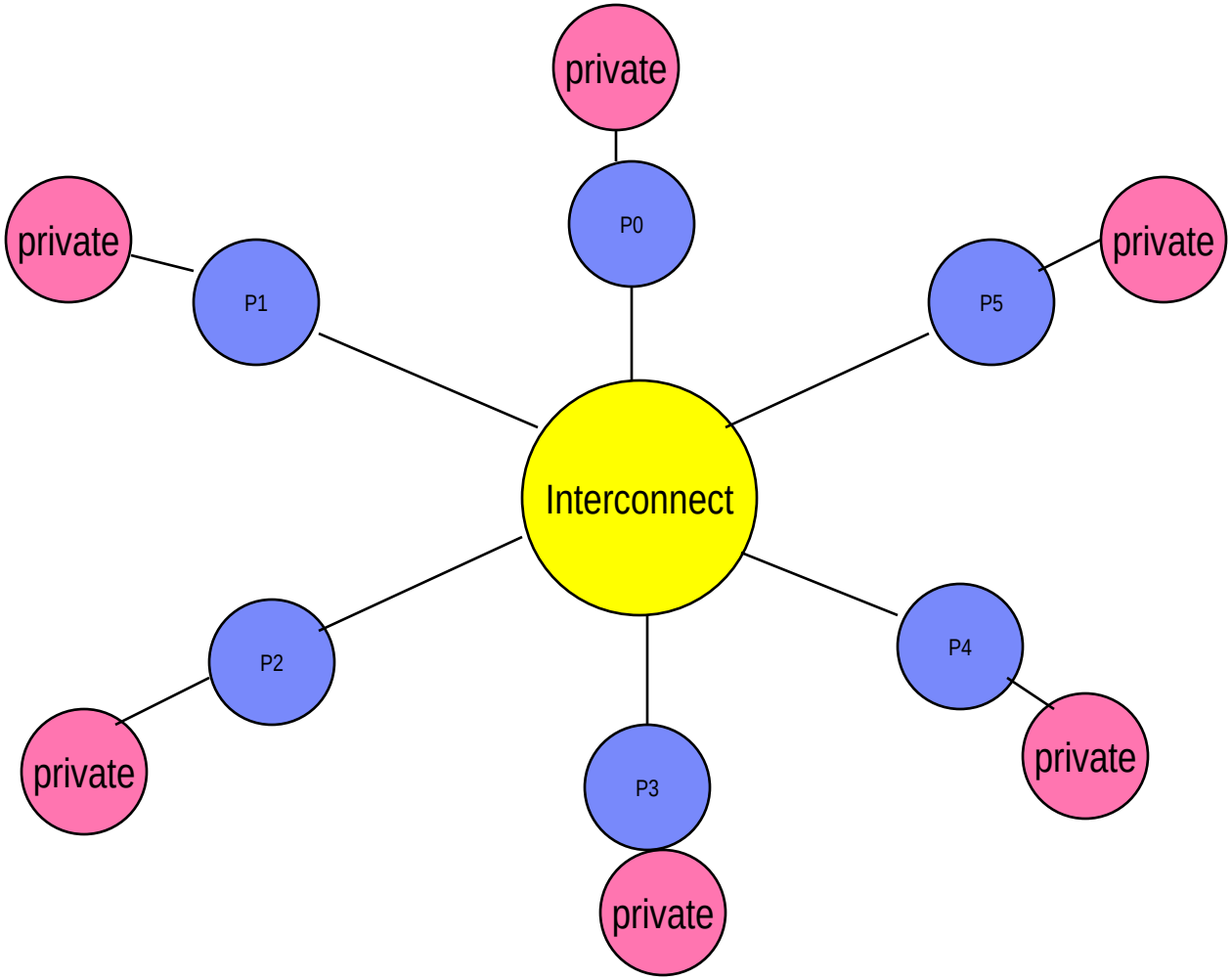
Can OpenMP work with MultiCore, Heterogeneous

- ***OpenMP is ideally suited for multicore architectures***
  - ***Memory and threading model map naturally***
  - ***Lightweight***
  - ***Mature***
  - ***Widely available and used***

Parallel programming models

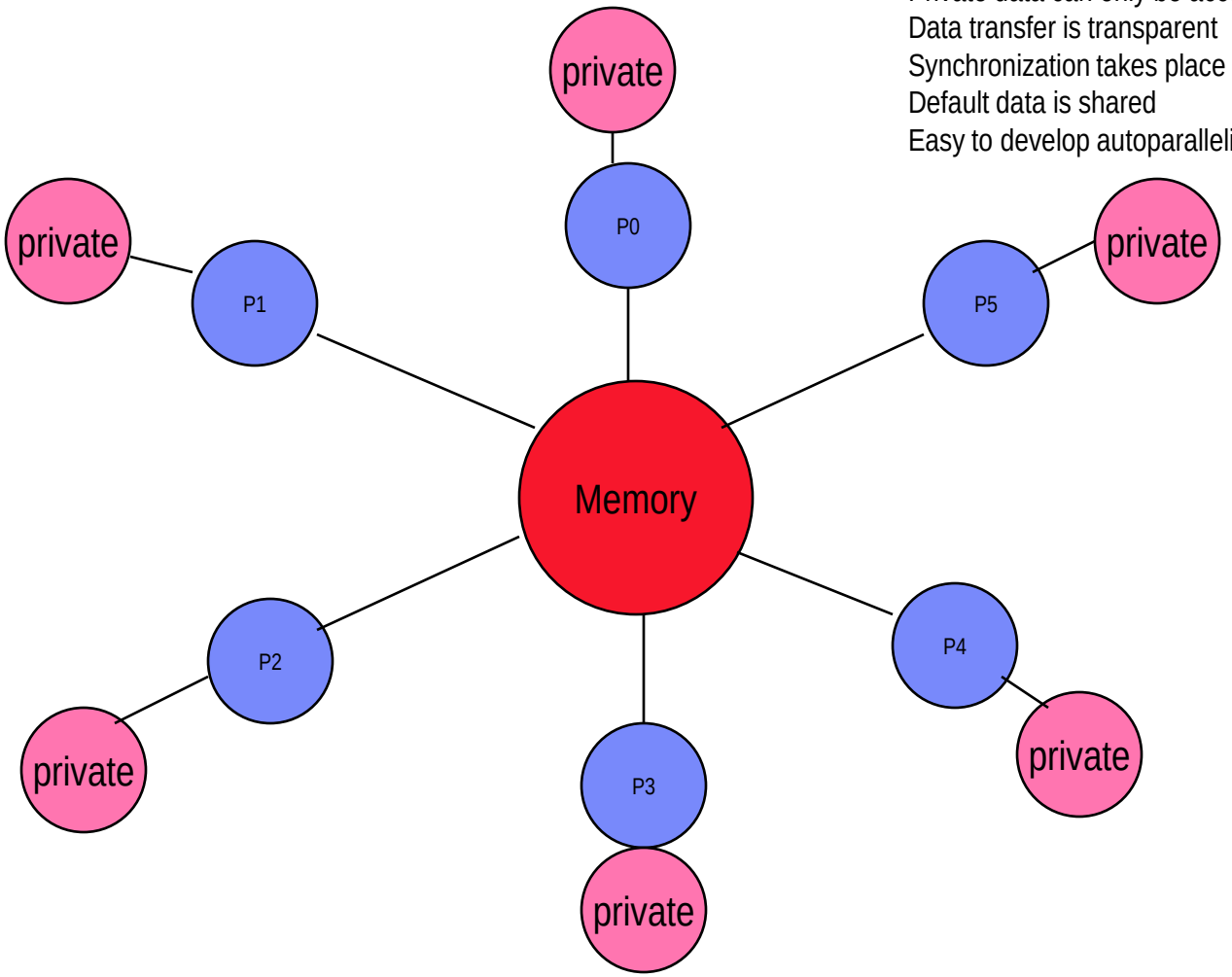
- A parallel programming model is a set of software technologies to express parallel algorithms and match applications with the underlying parallel systems. It encloses the areas of applications, programming languages, compilers, libraries, communications systems, and parallel I/O.
- Distributed Memory (any cluster and/or single SMP)
  - PVM (parallel Virtual machine)
  - MPI
- Shared Memory (SMP only)
  - Posix (standard but low level)
  - OpenMP not a standard, but everyone sticks to it
  - Automatic parallelization
- Partitioned Global Address Space
  - UPC
- Others
  - HPF, Global Arrays, Co-Array Fortran, Stream_processing, SHMEM

Distributed Memory model

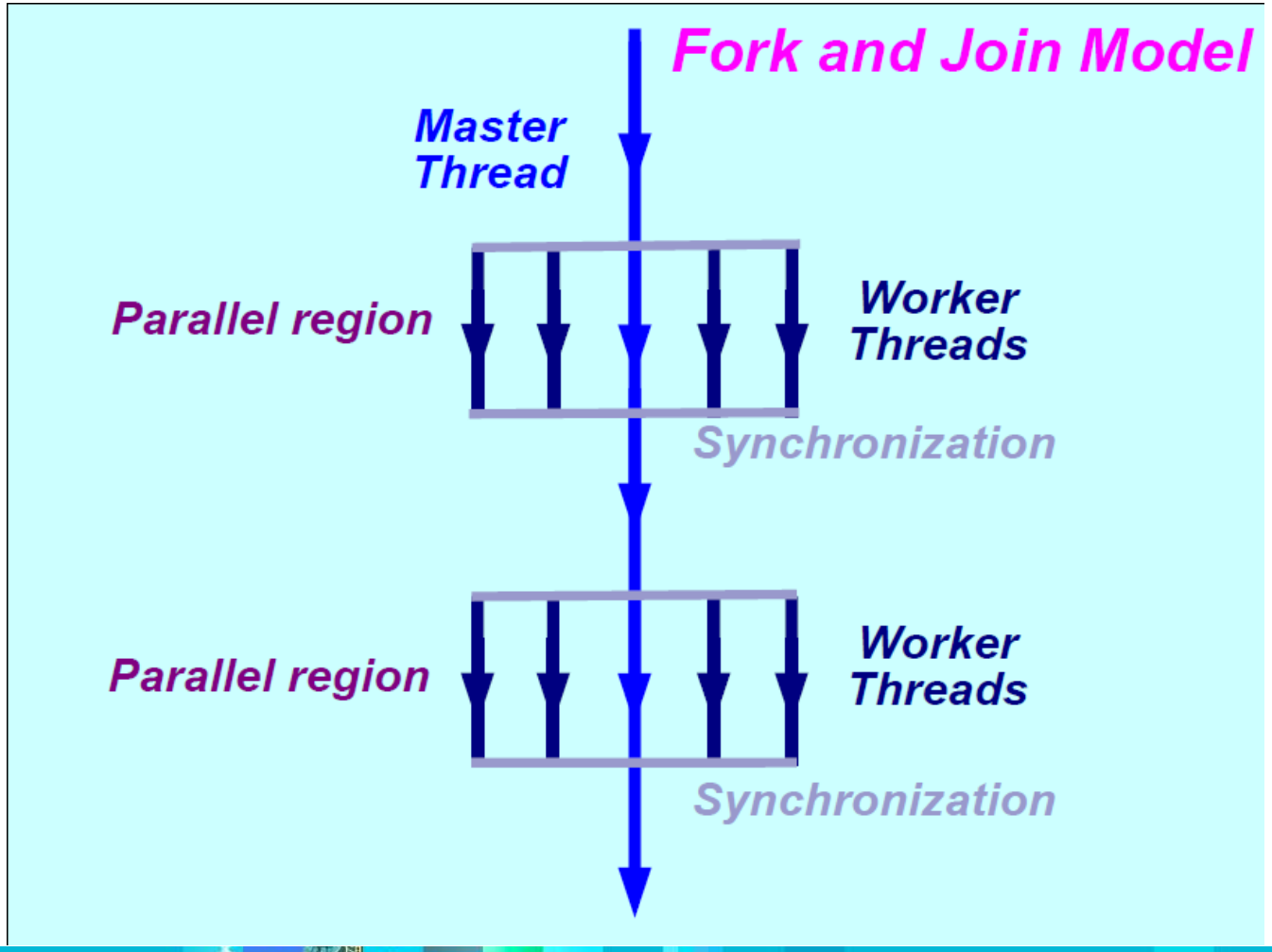


Shared Memory Model

All threads have access to same global shared memory
Data can be shared or private
Shared data is accessible by all
Private data can only be accessed by the owning thread
Data transfer is transparent
Synchronization takes place but is mostly implicit
Default data is shared
Easy to develop autoparallelizing compiler



The OpenMP Execution Model



Data Sharing Attribute

- ***In an OpenMP program, data needs to be "labeled"***
- ***there are two basic types:***
  - ***Shared - There is only one instance of the data***
    - ***✔ All threads can read and write the data simultaneously, unless protected through a specific OpenMP construct***
    - ***✔ All changes made are visible to all threads***
      - ***But not necessarily immediately, unless enforced .....***
  - ***Private - Each thread has a copy of the data***
    - ***✔ No other thread can access this data***
    - ***✔ Changes only visible to the thread owning the data***

The private and shared clauses

- **private (list)**
  - ✔ ***No storage association with original object***
  - ✔ ***All references are to the local object***
  - ✔ ***Values are undefined on entry and exit***
- **shared (list)**
  - ✔ ***Data is accessible by all threads in the team***
  - ✔ ***All threads access the same address space***

Loop parallelized with openMP

- 1st thing to do is set up a parallel region
- Every thread will execute all the code in the region until they reach the work sharing construct.
- The rest are clauses

```
#pragma omp parallel default(none) shared(n,x,y) private (i)
{
   #pragma omp for
            for (int i=0;i<n;++i)
                     x[i]+=y[i];
} /* end of parallel region */
```

Defining Parallelism in OpenMP

- ***OpenMP Team := Master + Workers***
- ***A Parallel Region is a block of code executed by all threads simultaneously***
  - ☞ ***The master thread always has thread ID 0***
  - ☞ ***Thread adjustment (if enabled) is only done before entering a parallel region***
  - ☞ ***Parallel regions can be nested, but support for this is implementation dependent***
  - ☞ ***An "if" clause can be used to guard the parallel region; in case the condition evaluates to "false", the code is executed serially***
- ***A work-sharing construct divides the execution of the enclosed code region among the members of the team; in other words: they split the work***

Directive Format

- C/C++
  - #pragma omp directive [clause [clause] …]
  - Continuation: \
  - Conditional compilation: _OPENMP macro is set
- Fortran:
  - *Fortran: directives are case insensitive*
    - *Syntax: sentinel directive [clause [[,] clause]...]*
    - *The sentinel is one of the following:*
      - ✔ *!$OMP or C$OMP or *$OMP (fixed format)*
      - ✔ *!$OMP (free format)*
  - *Continuation: follows the language syntax*
  - *Conditional compilation**: !$ or C$ -> 2 spaces***

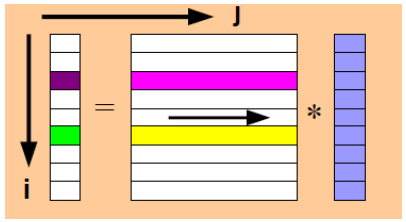
OpenMP clauses

- ***Many OpenMP directives support clauses***
  - ***These clauses are used to provide additional information with the directive***
- ***For example, private(a) is a clause to the "for" directive:***
  - **#pragma omp for private(a)**
- ***The specific clause(s) that can be used, depend on the directive***

Matrix times vector example on 2 threads

```
#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[0][j]*c[j])
  - A[0]=sum;

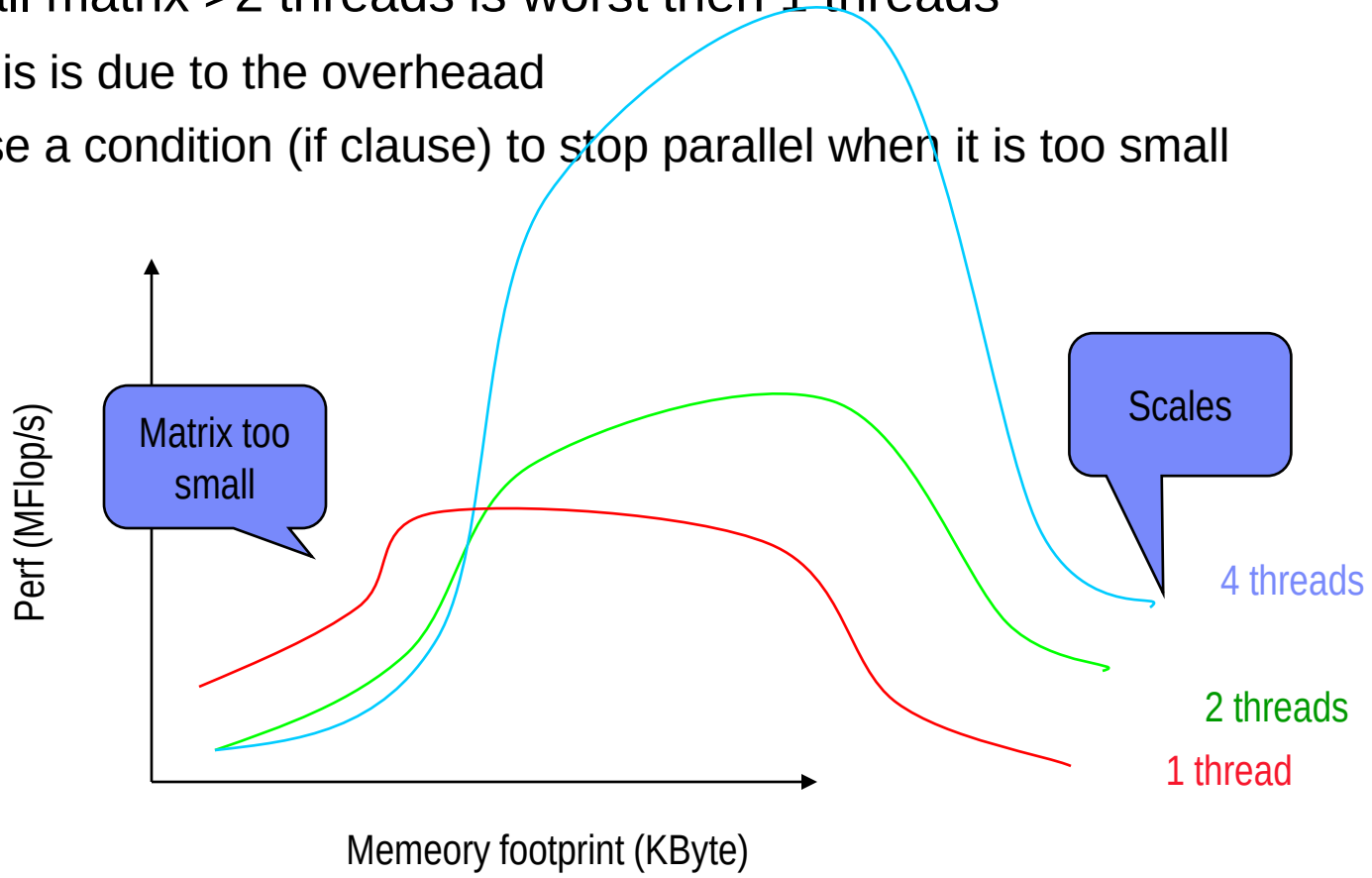
- TID=1

For (i=5,6,7,8,9)

- i=7
  - Sum=Sum(b[7][j]*c[j])
  - A[7]=sum;

OpenMP Performance

- Small matrix >2 threads is worst then 1 threads
  - This is due to the overheaad
  - Use a condition (if clause) to stop parallel when it is too small



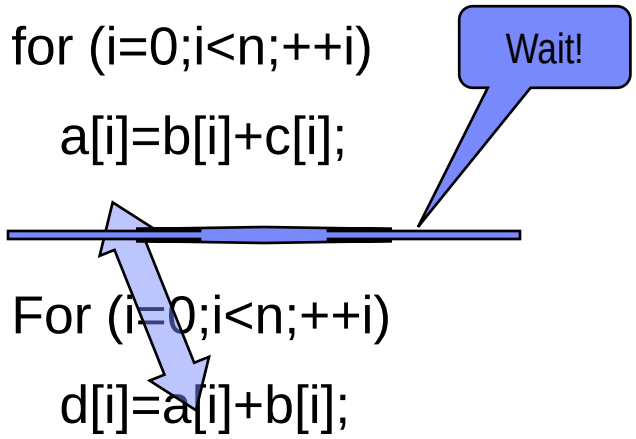
The if clause

- **if (scalar expression)**
  - ✔ ***Only execute in parallel if expression evaluates to true***
  - ✔ ***Otherwise, execute serially***

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

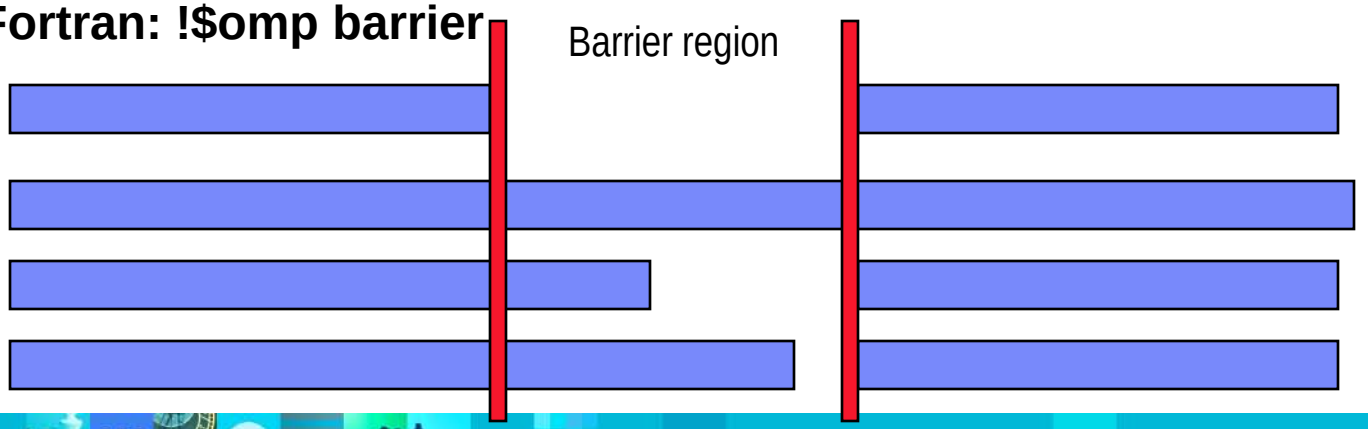
Common problems and parallel construct solutions

- Suppose we run each of these two loops in parallel over i
  - This will fail one day. Why?
  - Use a barrier when all threads will wait until all of them are done before continuing
  - Very expensive,may not scale, causes idle, what to do with that idle time?



When to use barrier

- When data is updated asynchronously and the data integrity is at risk
- E.g.:
  - Between parts in the code that read and write the same section of memory
  - After one timestep/iteration in a solver
- Expensive, won't scale with more cores, use with care
- Syntax:
  - **C/C++: #pragma omp barrier**
  - **Fortran: !$omp barrier**



Nowait clause

- Very conveneient, leave out the barrier in many OpenMP clause which has hidden barriers
- Thread Will not synchronize/ wait at the end of that construct
- It is a clause on the pragma, good when computation between workshare is independent

```
$pragma omp for nowait
{
}
```

- In C, it is one of the clause on the pragma
- In Fortran, nowait clause is appended at the closing part

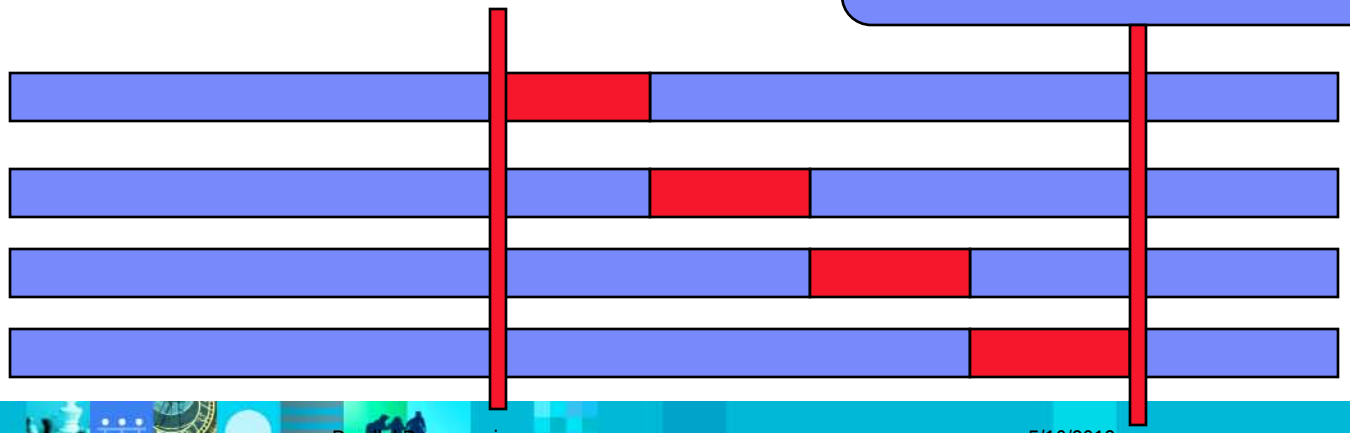
Critical region

- Prevent simultaneous update of shared variables
- Can cause race conditions
- Force only one thread at a time through

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

One at a time please

If sum is a shared variable, this loop can not run parallel



Components of OpenMP

- **Directives**
  - Tasking
  - Parallel region
  - Work sharing
  - Synchronization
  - Data scope attributes
    - Private
    - Firstprivate
    - Lastprivate
    - Shared
    - reduction
  - Orphaning
- **Environment Variables**
  - Number of threads
  - Scheduling type
  - Dynamic thread adjustment
  - Nested parallelism
  - Stacksize
  - Idle threads
  - Active levels
  - Thread limit
- **Runtime Variables**
  - Number of threads
  - Thread id
  - Dynamic thread adjustment
  - Nested Parallelism
  - Schedule
  - Active Levels
  - Thread limit
  - Nesting Level
  - Ancestor thread
  - Team size
  - Wallclock Timer
  - locking

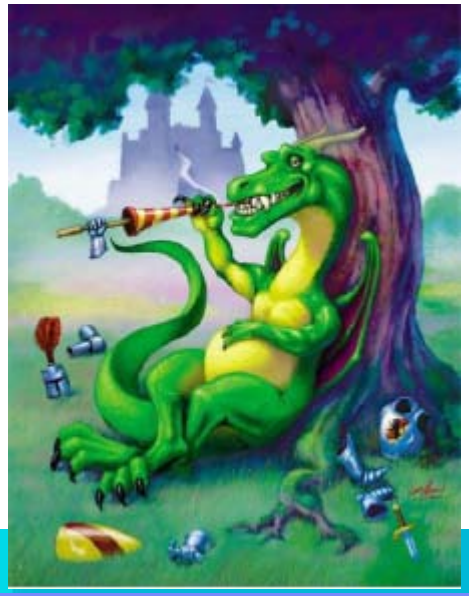
Parallel Region

- A block of code where all threads execute until you get to a worksharing construct
- There is always a barrier at the end of the parallel region, you can't get rid of it

```
#pragma omp parallel [clause[[,] clause] … ]
{
} //implied barrier
```

Agenda

- Is it legal to do tasks in C++ today?
- The OpenMP ARB
- History of OpenMP
- Intro tutorial to OpenMP
- Bonus 1: Towards OpenMP 4.0
- Fragen?



Drive to 4.0

- Initial work to support Fortran 2003
- Development of an error model
  - The `done` directive
  - Callbacks for integrated error handling
- Interoperability and composability
  - Interactions between thread models
  - Interfaces to support interactions with distributed models
- Refinements to the OpenMP tasking model
  - Specifying task dependencies (think data flow)
  - Task reductions, task-only threads, `omp while`
- Affinity (previous slide)
- Sequentially consistent atomic operations
- How to specify subarrays in C

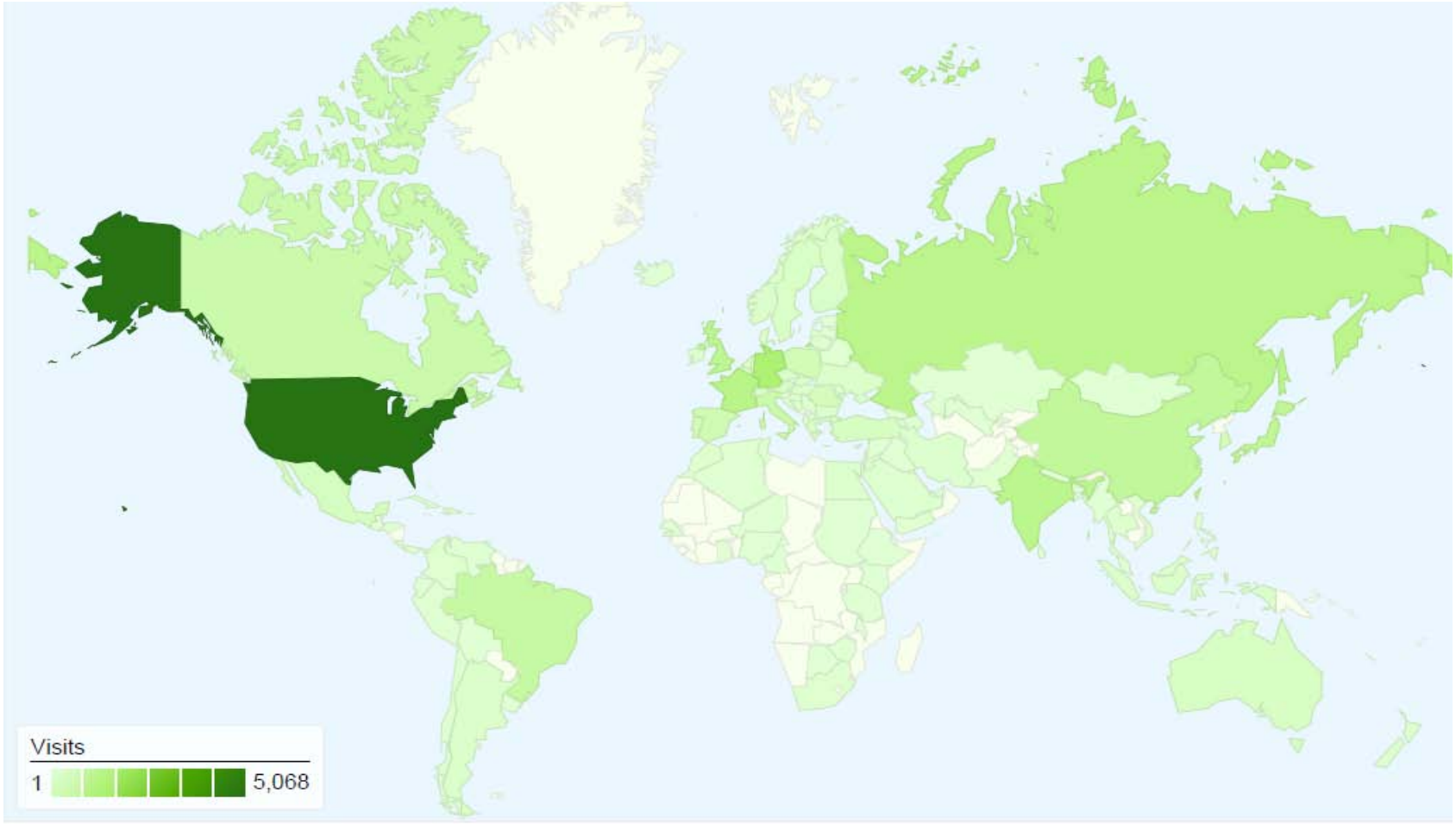
Beyond 4.0

- Other topics being considered beyond OpenMP 4.0
  - Transactional memory and thread level speculation
  - Additional task/thread synchronization mechanisms
  - Support for FPGA
  - Extending OpenMP to C/C++11
  - Extending OpenMP to additional languages
  - Incorporating tools support
  - Further support Embedded Chips
- How can you help shape the future of OpenMP?
  - Attend IWOMP, become a cOMPunity member
  - Lobby your institution to join the OpenMP ARB

Connect OpenMP with Users

- Hold our leadership in HPC community
- Aggressive expansion into non-traditional OpenMP areas
  - Oil&Gas, Financial, Biotechs, graphics, games, consumer electronics
  - To be discussed in OpenMP Lang
  - And ARB F2F meetings

Where are OpenMP Users?



Lang Meeting coming near you

- Lang Committee Telecons weekly
  - Active Subgroup Telecons weekly
- Move Language meetings close to OpenMP vendors/users (in NA, Europe/Asia/SA)
  - Solicit hosts for OpenMP Language meetings
- Upcoming in 2012
  - San Francisco - Silicon Valley Companies
  - Rome (IWOMP SC chair report) - CASPUR
    - Bjarne Stroustrup will keynote
  - Toronto - IBM

Do more with openmp.org

- Active News update and Discussion Forum
- Tutorial Material
- Google calendar schedule
  - http://openmp.org/calendar.html
- Gather info on Who Uses OpenMP / How?
  - http://openmp.org/wp/whos-using-openmp/
- More frequent, scheduled press releases
  - http://eon.businesswire.com/news/eon/20120327006830/en/accelerator/multicore/automotive
- Extend beyond HPC Tradeshows
  - Multicore/Embedded Expo, ISC? SIGGRAPH? Oil and Gas, Financial, BioTech, CES? E3/GDC

Food for thought and Q/A

- This is the chance to get a copy before you have to pay for it:
  - C++ : http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3291.pdf
  - C++ (last free version): http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3242.pdf
  - C: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
- Participate and feedback to Compiler
  - What features/libraries interest you or your customers?
  - What problem/annoyance you would like the Std to resolve?
  - Is Special Math important to you?
  - Do you expect 0x features to be used quickly by your customers?
- Talk to me at my blog:
  - http://www.ibm.com/software/rational/cafe/blogs/cpp-standard

My blogs and email address

- OpenMP CEO: http://openmp.org/wp/about-openmp/
  My Blogs: http://ibm.co/pCvPHR
  C++11 status:
  http://www.ibm.com/software/awdtools/xlcpp/aix/features/?S_CMP=rnav
  Boost test results
  http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911
  C/C++ Compilers Support/Feature Request Page
  http://www.ibm.com/software/awdtools/ccompilers/support/
  http://www.ibm.com/support/docview.wss?uid=swg27005811
  STM:
  https://sites.google.com/site/tmforcplusplus/

- Tell us how you use OpenMP:
- http://openmp.org/wp/whos-using-openmp/

Acknowledgement

- Some slides are borrowed from committee presentations by various committee members, their proposals, and private communication
- Special Thanks to Ruud van de Pas, and Bronis de Supinski for some of the slides