GPU/Accelerator programming with OpenMP 4.0: yet another Significant Paradigm Shift in High-level Parallel Computing

Michael Wong, Senior Compiler Technical Lead/Architect
michaelw@ca.ibm.com
OpenMP CEO
Chair of WG21 SG5 Transactional Memory
ISOCPP.org, Director, VP
Vice Chair of Programming Languages, Standards Council of Canada
WG21 C++ Standard, Head of Delegation for Canada and IBM

CPPCON 2014

Acknowledgement and Disclaimer

- Numerous people internal and external to the OpenMP WG, in industry and academia, have made contributions, influenced ideas, written part of this presentations, and offered feedbacks to form part of this talk.
- I even lifted this acknowledgement and disclaimer from some of them.
- But I claim all credit for errors, and stupid mistakes. **These are mine, all mine!**
- Any opinions expressed in this presentation are my opinions and do not necessarily reflect the opinions of IBM or OpenMP or ISO C++.

Legal Disclaimer

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM, PowerPC and the IBM logo are trademarks or registered trademarks of IBM or its subsidiaries in the United States and other countries.
- The OpenMP_Timeline files here are licensed under the three clause BSD license, http://opensource.org/licenses/BSD-3-Clause
- Other company, product, and service names may be trademarks or service marks of others..

What is OpenMP about?

And how does it fit with C++?

Common-vendor Specification Parallel Programming model on Multiple compilers

AMD, Convey, Cray, Fujitsu, HP, IBM, Intel, NEC, NVIDIA, Oracle, RedHat (GNU), ST Mircoelectronics, TI, clang/llvm

A de-facto Standard: Across 3 Major General Purpose Languages

C++, C, Fortran

A de-facto Standard: One High-Level Accelerator Language

One High-Level Vector SIMD language too!

Support Multiple Devices and let the local compiler generate the best code

Xeon Phi, NVIDIA, GPU, GPGPU, DSP, MIC, ARM and FPGA

So how does it fit with other GPU/Accelerator efforts?

ISO C++ WG21 SG1 Parallelism TS

C++AMP

OpenCL

Cuda?

WG21 SG1 Parallelism TS

```
std::vector<int> v = ...
// standard sequential sort
std::sort(vec.begin(), vec.end());
using namespace
    std::experimental::parallel;
// explicitly sequential sort
sort(seq, v.begin(), v.end());
// permitting parallel execution
sort(par, v.begin(), v.end());
// permitting vectorization as well
sort(vecpar_vec, v.begin(), v.end());
// sort with dynamically-selected
    execution
```

```
size_t threshold = ...
execution_policy exec = seq;
if (v.size() > threshold) {
  exec = par;
}
sort(exec, v.begin(), v.end());
```

C++AMP

```
void AddArrays(int n, int m, int * pA, int * pB, int * pSum) {
 concurrency::array_view<int,2> a(n, m, pA), b(n, m, pB),
   sum(n, m, pSum);
 concurrency::parallel_for_each(sum.extent,
   [=](concurrency::index<2> i) restrict(amp)
 {
   sum[i] = a[i] + b[i];
 });
}
```

CUDA

```
texture<float, 2, cudaReadModeElementType> tex;
void foo() {
  cudaArray* cu_array;
  // Allocate array
  cudaChannelFormatDesc description = cudaCreateChannelDesc<float>();
  cudaMallocArray(&cu_array, &description, width, height);
  // Copy image data to array
  …
  // Set texture parameters (default)
  …
  // Bind the array to the texture
  …
  // Run kernel
  …
  // Unbind the array from the texture
}
```



Its like the difference between:

An Aircraft Carrier Battle Group (ISO)

And a Cruiser (Consortium: OpenMP)

And a Destroyer (Company Specific language)

Agenda

- **What Now?**
- OpenMP ARB Corporation
- A Quick Tutorial
- A few key features in 4.0
- Accelerators and GPU programming
- Implementation status and Design in clang/llvm
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

What now?

- Nearly every C, C++ features makes for beautiful, elegant code for developers (Disclaimer: I love C++)
  - Please insert your beautiful code here:
  - Elegance is efficiency, or is it? Or
  - What we lack in beauty, we gain in efficiency; Or do we?
- The new C++11 Std is
  - 1353 pages compared to 817 pages in C++03
- The new C++14 Std is
  - 1373 pages (N3937), vs the free n3972
- The new C11 is
  - 701 pages compared to 550 pages in C99
- OpenMP 3.1 is
  - 354 pages and growing
- OpenMP 4.0 is
  - 520 pages

Beautiful and elegant Lambdas

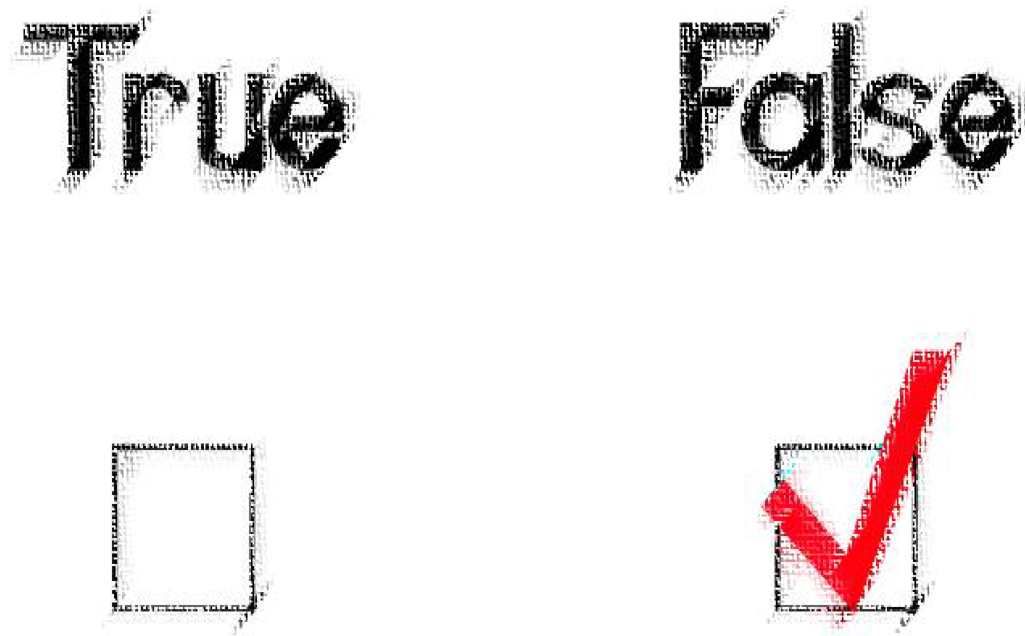
C++98	C++11

vector<int>::iterator i = v.begin();
for(; i != v.end(); ++i) {
   if(*i > x && *i < y)
      break;
}
	auto i = **find_if**(begin(v), end(v),
	[=](int i) {
	   return i > x && i < y;
	});

- "Lambdas, Lambdas Everywhere" *http://vimeo.com/23975522*
- *Full Disclosure: I love C++ and have for many years*
- *But … What is wrong here?*

The Truth

- Q: Does your language allow you to access all the GFLOPS of your machine?



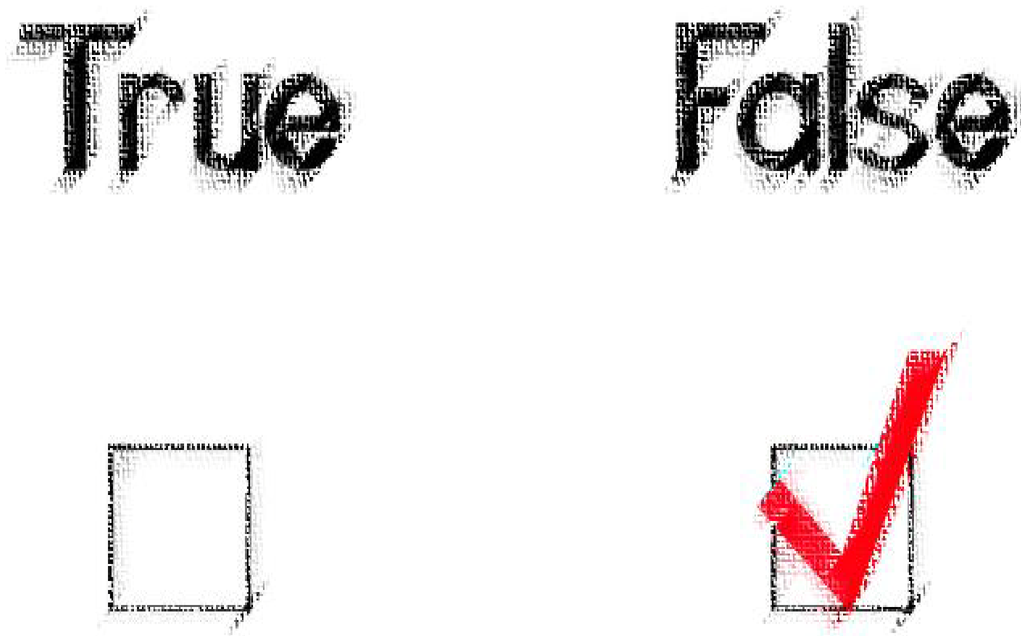
“Is there in Truth No Beauty?”

from *Jordan by George Herbert*

- Q: Does your language allow you to access all the GFLOPS of your machine?
- A: What a quaint concept!
  - I thought its natural to drop out into OpenCL, CUDA, OpenGL, DirectX, C++AMP, Assembler …. to get at my GPU
  - Why? I just use my language as a cool driver, it’s a great scripting language too. But for real kernel computation, I just use Fortran
  - I write vectorized code, so my vendor offers me intrinsics, they also tell me they can auto-vectorize, though I am not sure how much they really do, so I am looking into OpenCL
  - Well, I used to use one thread, but now that I use multiple threads, I can get at it with C++11, OpenMP, TBB, GCD, PPL, MS then continuation, Cilk
  - I know I may have a TM core somewhere, so my vendor offers me intrinsics
  - No I like using a single thread, so I just use C, or C++

The Question

- Q: Is it true that there is a language that allows you to access all the GFLOPS of your machine?



Power of Computing

- 1998, when C++ 98 was released
  - Intel Pentium II: 0.45 GFLOPS
  - No SIMD: SSE came in Pentium III
  - No GPUs: GPU came out a year later
- 2011: when C++11 was released
  - Intel Core-i7: 80 GFLOPS
  - AVX: 8 DP flops/HZ*4 cores *4.4 GHz= 140 GFlops
  - GTX 670: 2500 GFLOPS
- Computers have gotten so much faster, how come software have not?
  - Data structures and algorithms
  - latency

In 1998, a typical machine had the following flops

- .45 GFLOP, 1 core

- Single threaded C++98/C99 dominated this picture

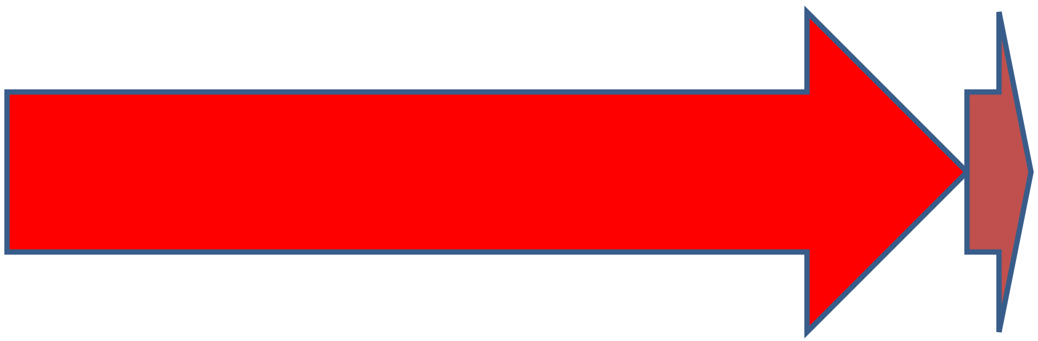
In 2011, a typical machine had the following flops

- 2500 GFLOP GPU

- To program the GPU, you use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP

In 2011, a typical machine had the following flops

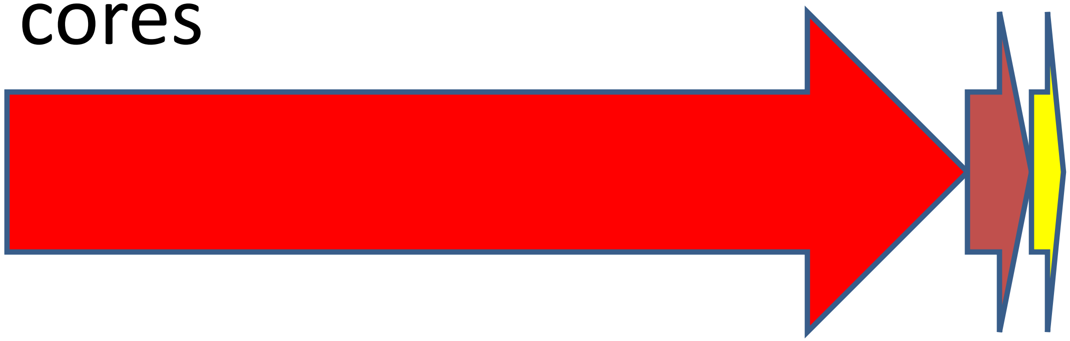
- 2500 GFLOP GPU+140GFLOP AVX



- To program the GPU, you use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP
- To program the vector unit, you use Intrinsics, OpenCL, or auto-vectorization

In 2011, a typical machine had the following flops

- 2500 GFLOP GPU+140GFLOP AVX+80GFLOP 4 cores



- To program the GPU, you use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP
- To program the vector unit, you use Intrinsics, OpenCL, or auto-vectorization
- To program the CPU, you use C/C++11, OpenMP, TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors

In 2011, a typical machine had the following flops

- 2500 GFLOP GPU+140GFLOP AVX+80GFLOP 4 cores+HTM

- To program the GPU, you use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP
- To program the vector unit, you use Intrinsics, OpenCL, or auto-vectorization
- To program the CPU, you use C/C++11, OpenMP, TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors
- To program HTM, you have?

In 2014, a typical machine had the following flops

- 2500 GFLOP GPU+140GFLOP AVX+80GFLOP 4 cores+HTM

- To program the GPU, you use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP, **OpenMP**
- To program the vector unit, you use Intrinsics, OpenCL, or auto-vectorization, **OpenMP**
- To program the CPU, you might use C/C++11/14, OpenMP, TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors
- To program HTM, you have the upcoming C++ TM TS

OpenMP 4.0: A Significant Paradigm S Parallelism

THE OPENMP® API SPECIFICATION FOR PARALLEL PROGRAMMING

Subscribe to the News Feed

»OpenMP Specifications

»About the OpenMP ARB
»Frequently Asked Questions
»Compilers
»Resources
»Who's Using OpenMP?
»Press Releases

»Discussion Forums

Events
»*Public OpenMP Calendar*

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

Follow @OpenMP_ARB

Search OpenMP.org
Google™ Custom Search
Search

OpenMP 4.0 Specifications Released

The OpenMP 4.0 API Specification is released with Significant New Standard Features

The OpenMP 4.0 API supports the programming of accelerators, SIMD programming, and better optimization using thread affinity

The OpenMP Consortium has released OpenMP API 4.0, a major upgrade of the OpenMP API standard language specifications. Besides several major enhancements, this release provides a new mechanism to describe regions of code where data and/or computation should be moved to another computing device.

Bronis R. de Supinski, Chair of the OpenMP Language Committee, stated that "*OpenMP 4.0 API is a major advance that adds two new forms of parallelism in the form of device constructs and SIMD constructs. It also includes several significant extensions for the loop-based and task-based forms of parallelism already supported in the OpenMP 3.1 API.*"

The 4.0 specification is now available on the »**OpenMP Specifications page**.

Standard for parallel programming extends its reach

With this release, the OpenMP API specifications, the de-facto standard for parallel programming on shared memory systems, continues to extend its reach beyond pure HPC to include DSPs, real time systems, and accelerators. The OpenMP API aims to provide high-level parallel language support for a wide range of applications, from automotive and aeronautics to biotech, automation, robotics and financial analysis.

New features in the OpenMP 4.0 API include:

· **Support for accelerators.** The OpenMP 4.0 API specification effort included significant participation by all the major vendors in order to support a wide variety of compute devices. OpenMP API provides mechanisms to describe regions of code where data and/or computation should be moved to another computing device. Several prototypes for the accelerator proposal have already been implemented.

· **SIMD constructs to vectorize both serial as well as parallelized loops.** With the advent of SIMD units in all major processor chips, portable support for accessing them is essential. OpenMP 4.0 API provides mechanisms to describe when multiple iterations of the loop can be executed concurrently using SIMD instructions and to describe how to create versions of functions that can be invoked across SIMD lanes.

Get
»OpenMP specs

Use
»OpenMP Compilers

Learn

»*Using OpenMP* -- the book
»*Using OpenMP* -- the examples
»*Using OpenMP* -- the forum
»Wikipedia
»OpenMP Tutorial
»More Resources

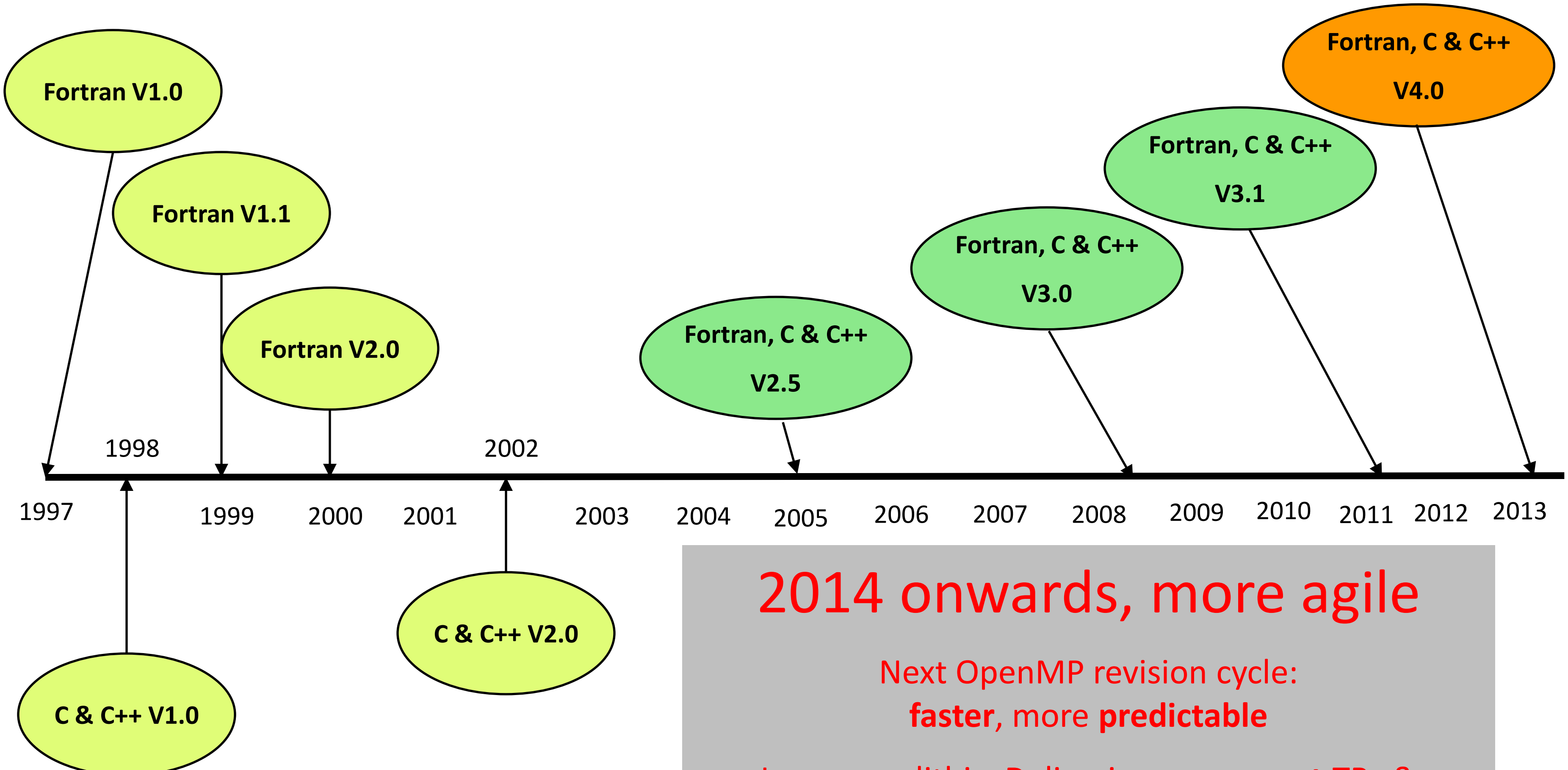
Discuss
»User Forum
Ask the experts and get answers to questions about OpenMP

Recent News
- OpenMP at SC13 Denver

Agenda

- What Now?
- **OpenMP ARB Corporation**
- A Quick Tutorial
- A few key features in 4.0
- Accelerators and GPU programming
- Implementation status and Design in clang/llvm
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

A brief history of OpenMP API by Kelvin Li



2014 onwards, more agile

Next OpenMP revision cycle:
faster, more **predictable**

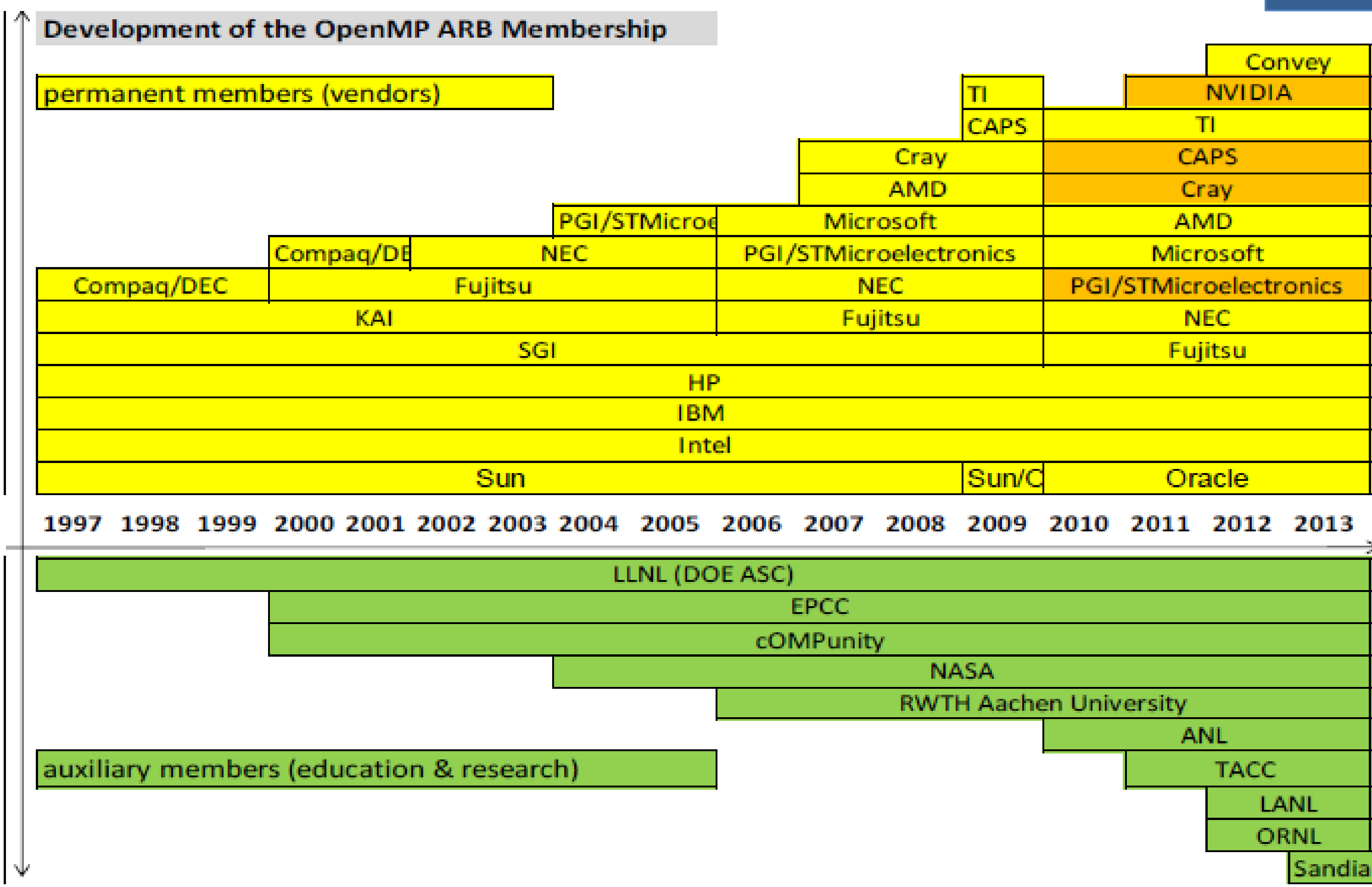
Less monolithic: Delivering **concurrent** TRs & language extensions

OpenMP is a living language

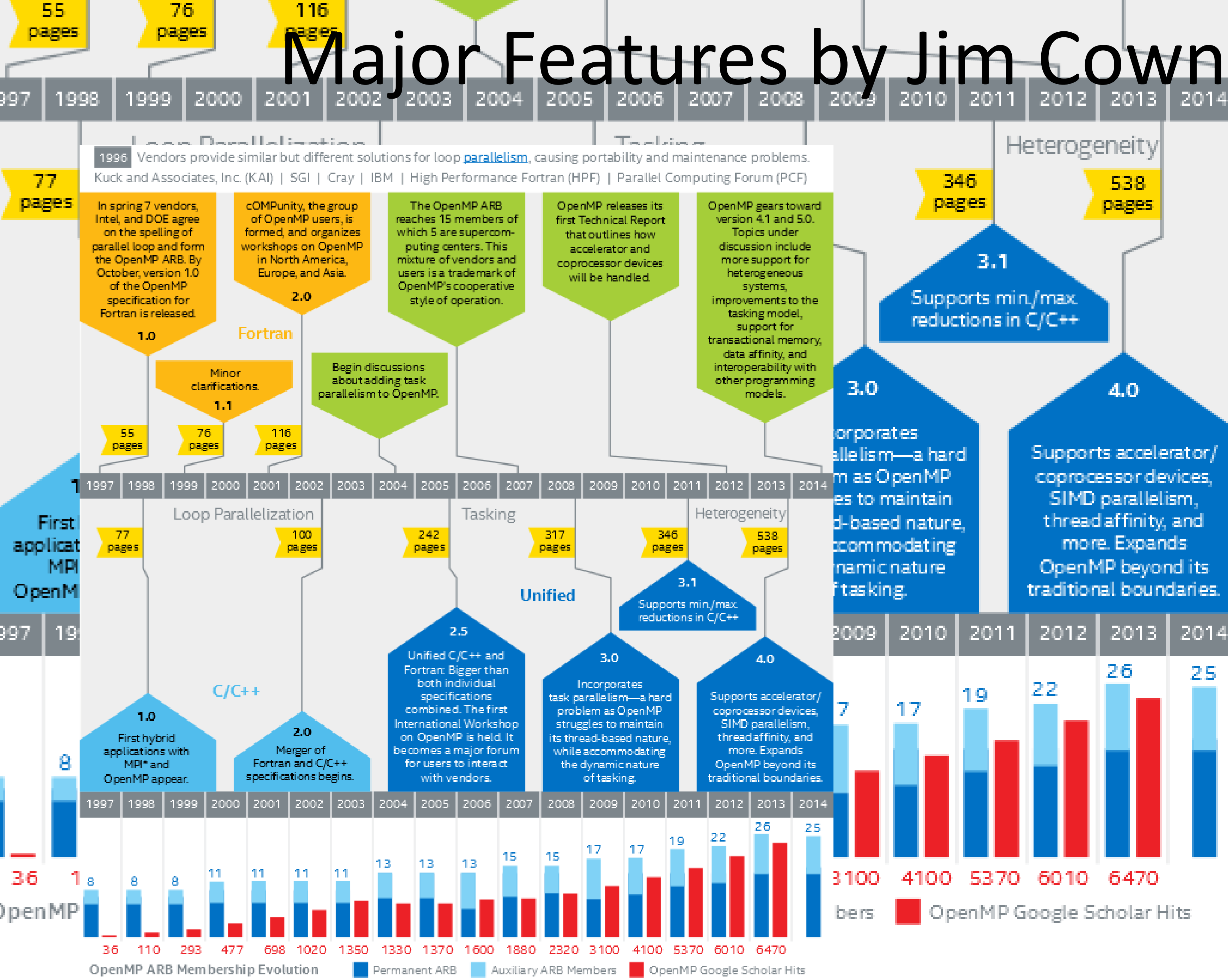
OpenMP Members growth

26 members and growing

- From Dieter An Mey, RWTH Aachen 2012, since 2012 added
  - Red Hat/GCC
  - Barcelona SuperComputing Centre
  - University of Houston



Major Features by Jim Cownie



OpenMP internal Organization

OpenMP ARB

Language WG

Marketing WG

Today	Accel	Error	Task	Tools	Affinity	Fortran 2003
Future	TM	Async/Eve nt	Interop	C++11	C11	Memory Model, Loops, Object oriented

The New Mission Statement of OpenMP

- OpenMP's new mission statement
  - "Standardize directive-based multi-language high-level parallelism that is performant, productive and portable"
  - Updated from
    - "Standardize and unify shared memory, thread-level parallelism for HPC"

Agenda

- What Now?
- OpenMP ARB Corporation
- **A Quick Tutorial**
- A few key features in 4.0
- Accelerators and GPU programming
- VectorSIMD Programming
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

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_acq
   uire) != 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
```

Execute Tasks First with Dependencies

- OpenMP 4.0 only

```
int main(int argc, char *argv[]) {
        #pragma omp parallel
        {
                #pragma omp single
                {
                        int x = 1;
                        #pragma omp task shared (x) depend (out:x)
                                {printf("Hello ");}
                        #pragma omp task shared (x) depend (in:x)
                                {printf("World ");}
                        #pragma omp taskwait
                        printf("Thank You ");
                }
        } // End of parallel region
        printf("\n");return(0);
}
Hello World Thank You
Hello World Thank You
Hello World Thank You
```

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 4.0 was released 2013***

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 are 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

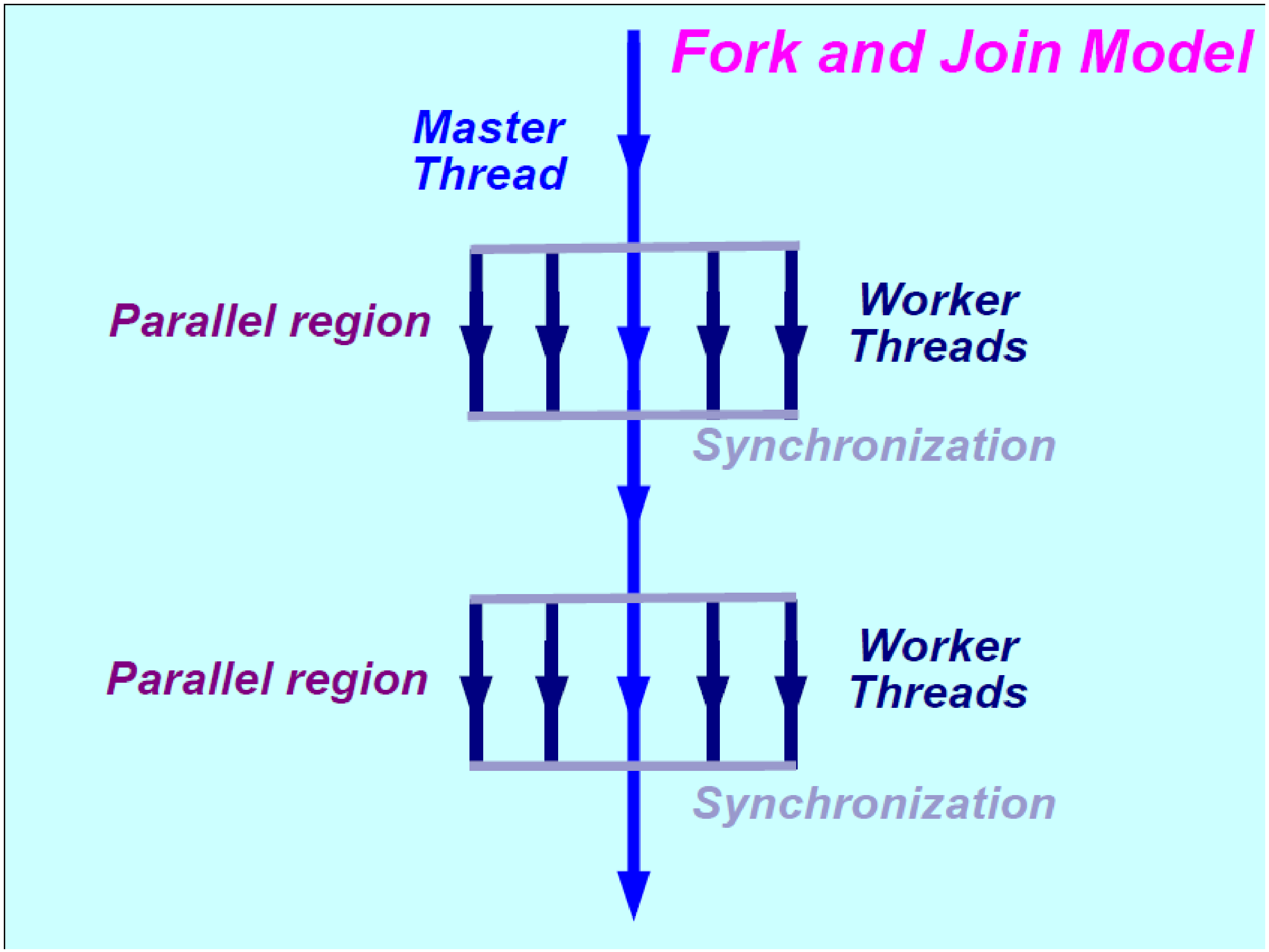
Advantages of OpenMP

- ***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***
- ***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***

The OpenMP Execution Model



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***

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

But why does OpenMP use pragmas?

It is an intentional design …

Pragmas can support 3 General Purpose Programming Languages and maintain the same style

C++

C

Fortran

And National Labs, weather research, nuclear simulations

Still have substantial kernels written in mix of Fortran and C driven by C++

Agenda

- What Now?
- OpenMP ARB Corporation
- A Quick Tutorial
- **A few key features in 4.0**
- Accelerators and GPU programming
- Implementation status and Design in clang/llvm
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

Goals

- Thread-rich computing environments are becoming more prevalent
  - more computing power, more threads
  - less memory relative to compute
- There is parallelism, it comes in many forms
  - hybrid MPI - OpenMP parallelism
  - mixed mode OpenMP / Pthread parallelism
  - nested OpenMP parallelism
- Have to exploit parallelism efficiently
  - providing ease of use for casual programmers
  - providing full control for power programmers
  - providing timing feedback

What did we accomplish in OpenMP 4.0?

- Broad form of accelerator support
- SIMD
- Cancellation (start of a full error model)
- Task dependencies and task groups
- Thread Affinity
- User-defined reductions
- Initial Fortran 2003
- C/C++ array sections
- Sequentially Consistent Atomics
- Display initial OpenMP internal control variable state

Compilers are here!

- Intel 13.1 compiler supports Accelerators/SIMD
- Oracle/Sun Studio 12.4 Beta just announced full OpenMP 4.0
- GCC 4.9 shipped April 9, 2014 supports 4.0
- Clang support for OpenMP injecting into trunk, first appears in 3.5 last week
- Cray, TI, IBM coming online

In 2014, a typical machine had the following flops

- 2500 GFLOP GPU+140GFLOP AVX+80GFLOP 4 cores+HTM

- To program the GPU, you have to use CUDA, OpenCL, OpenGL, DirectX, Intrinsics, C++AMP, **OpenMP**
- To program the vector unit, you have to use Intrinsics, OpenCL, or auto-vectorization, **OpenMP**
- To program the CPU, you might use C/C++11/14, OpenMP, TBB, Cilk, MS Async/then continuation, Apple GCD, Google executors
- To program HTM, you have the upcoming C++ TM TS

Agenda

- What Now?
- OpenMP ARB Corporation
- A Quick Tutorial
- A few key features in 4.0
- **Accelerators and GPU Programming**
- Implementation status and Design in clang/llvm
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

OpenMP Accelerator Subcommittee

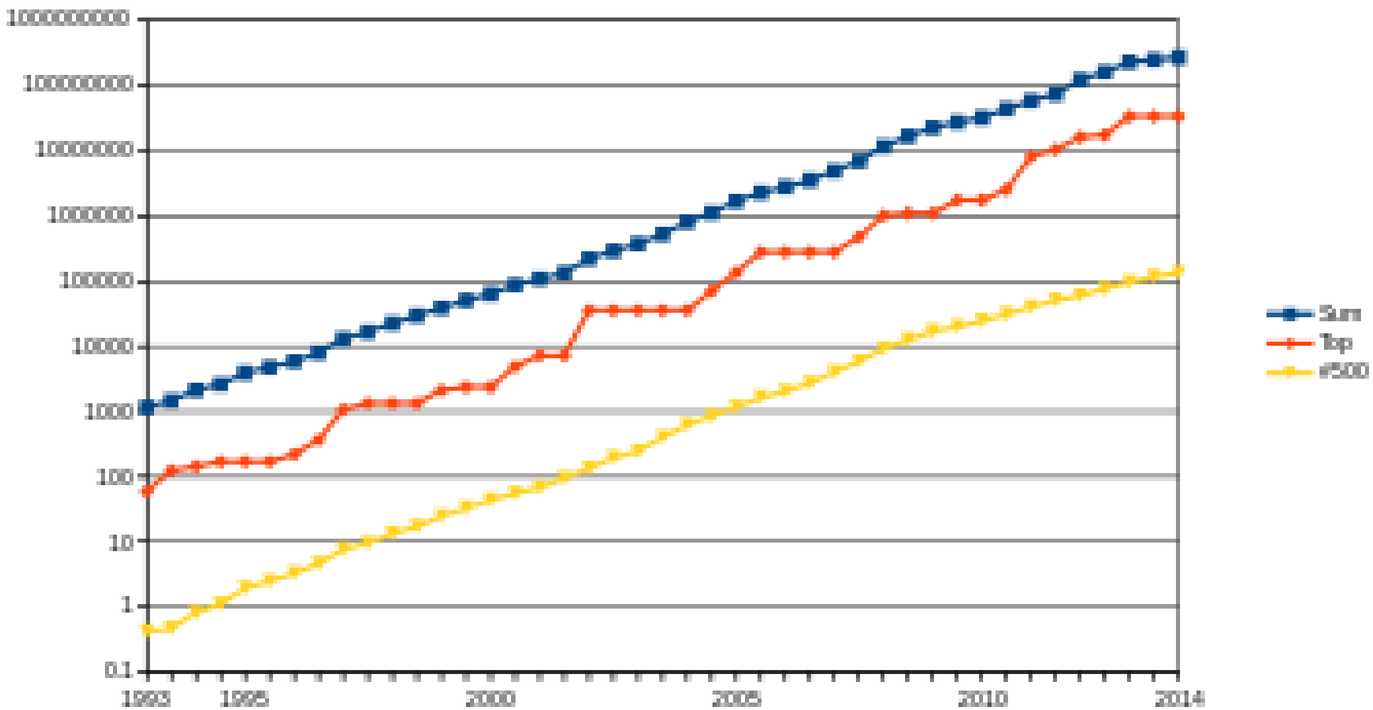
- Co-chairs Technical leads
  - Jame Beyers- Cray (courtesy for slides)
  - Eric Stotzer – TI (courtesy for slides)
- Active subcommittee members
  - Xinmin Tian – Intel (courtesy for slides)
  - Ravi Narayanaswamy – Intel (courtesy for slides)
  - Jeff Larkin – Nvidia
  - Kent Milfeld – TACC
  - Henry Jin – NASA
  - Kevin O'Brien, Kelvin Li, Alexandre Eichenberger, IBM
  - Christian Terboven– RWTH Aachen (courtesy for slides)
  - Michael Klemm – Intel (courtesy for slides)
  - Stephane Cheveau – CAPS
  - Convey, AMD, ORNL, TU Dresden,

So, how do you program GPU?

OpenMP
OpenMPCon USERS CONVENTION

Why is GPU important now?

- Or is it a flash in the pan?
- The race to exascale computing .. 10 [18] flops
- Vertical scale is in GFLOPS



Top500 contenders

What is OpenMP Model's aim?

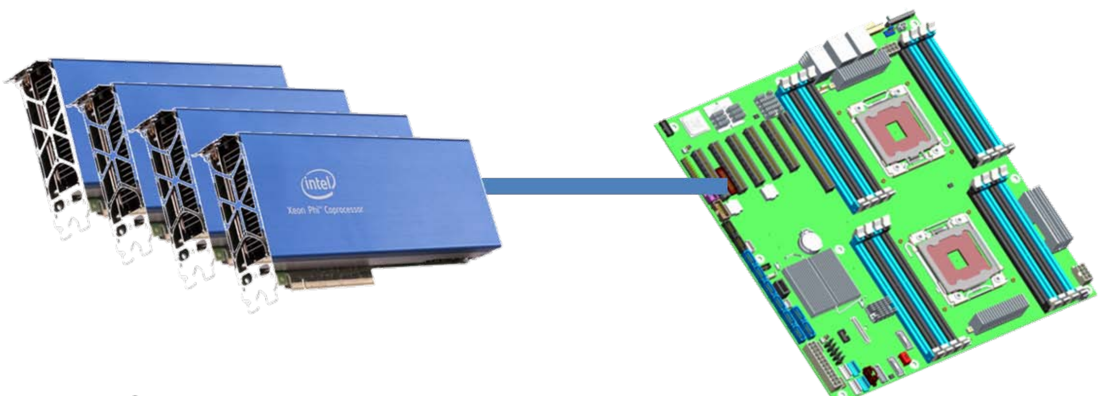
- All forms of accelerators, DSP, GPU, APU, GPGPU
- Network heterogenous consumer devices
  - Kitchen appliances, drones, signal processors, medical imaging, auto, telecom, automation, not just graphics





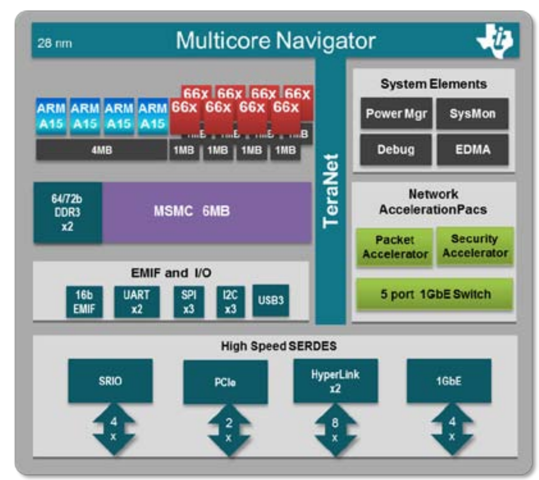

Heterogeneous Device model

- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  - One host
  - Multiple accelerators/coprocessors of the same kind



Coprocessors

Host



Heterogeneous SoC

Glossary

- Device:
  an implementation-defined (logical) execution unit
- League:
  the set of threads teams created by a `teams` construct
- Contention group:
  threads of a team in a league and their descendant threads
- Device data environment:
  Data environment as defined by `target data` or `target` constructs
- Mapped variable:
  An *original variable* in a (host) data environment with a *corresponding variable* in a device data environment
- Mapable type:
  A type that is amenable for mapped variables. (Bitwise copyable plus additional restrictions.)

OpenMP 4.0 Device Constructs

- Execute code on a target device
  - **omp target** *[clause[[,] clause],…]* *structured-block*
  - **omp declare target** *[function-definitions-or-declarations]*
- Map variables to a target device
  - **map** *([map-type:] list) // map clause*
    map-type := **alloc** | **tofrom** | **to** | **from**
  - **omp target data** *[clause[[,] clause],…]* *structured-block*
  - **omp target update** *[clause[[,] clause],…]*
  - **omp declare target** *[variable-definitions-or-declarations]*
- Workshare for acceleration
  - **omp teams** *[clause[[,] clause],…]* *structured-block*
  - **omp distribute** *[clause[[,] clause],…]* *for-loops*

target Construct

- Transfer control from the host to the device
- Syntax (C/C++)

```
#pragma omp target [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target [clause[[,] clause],…]
structured-block
```

- Clauses

```
device(scalar-integer-expression)
map(alloc | to | from | tofrom: list)
if(scalar-expr)
```

target data Construct

- Create a device data environment
- Syntax (C/C++)

```
#pragma omp target data [clause[[,] clause],…]
structured-block
```

- Syntax (Fortran)

```
!$omp target data [clause[[,] clause],…]
structured-block
```

- Clauses

```
device(scalar-integer-expression)
map(alloc | to | from | tofrom: list)
if(scalar-expr)
```

target update Construct

- Issue data transfers between host and devices
- Syntax (C/C++)

```
#pragma omp target update [clause[[,] clause],…]
```

- Syntax (Fortran)

```
!$omp target data update [clause[[,] clause],…]
```

- Clauses

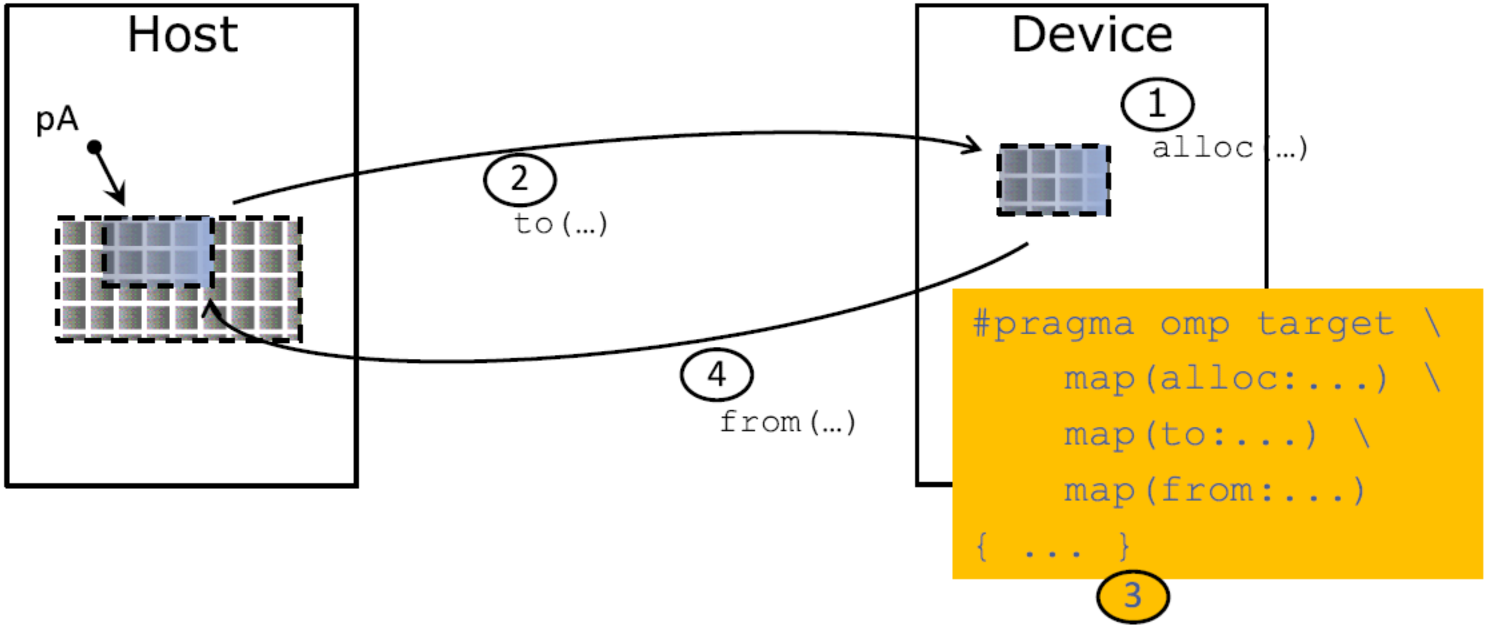
```
device(scalar-integer-expression)
to(list)
from(list)
if(scalar-expr)
```

Execution Model

- The `target construct` transfers the control flow to the target device
  - → The transfer clauses control direction of data flow
  - → Array notation is used to describe array length
- The `target data` construct creates a scoped device data environment
  - → The transfer clauses control direction of data flow
  - → The device data environment is valid through the lifetime of the target data region
- Use `target update` to request data transfers from within a target data region

Execution Model and Data Environment

- Data environment is lexically scoped
  - → Data environment is destroyed at closing curly brace
  - → Allocated buffers/data are automatically released



map Clause

```
extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);

   #pragma omp target map(to:v1[0:N],v2[:N]) \\
                      map(from:p[0:N])
   #pragma omp parallel for
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];

   output(p, N);
}
```

- The `target` construct creates a new *device data environment* and explicitly **maps** the array sections v1[0:N], v2[:N] and p[0:N] to the new device data environment.
- The variable N implicitly mapped into the new device data environment from the encountering task's data environment.

Map-types:

- **`alloc:`** allocate storage for corresponding variable
- **`to:`** alloc and assign value of original variable to corresponding variable on entry
- **`from:`** alloc and assign value of corresponding variable to original variable on exit
- **`tofrom:`** default, both to and form

`target` Construct Example

- Use target construct to
  - Transfer control from the host to the device
  - Establish a device data environment (if not yet done)
- Host thread waits until offloaded region completed
  - Use other OpenMP constructs for asynchronicity

```
#pragma omp target map(to:b[0:count]) map(to:c,d) map(from:a[0:count])
  {
#pragma omp parallel for
    for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
    }
  }
```

Data Environments

- Create a data environment to keep data on devices
  - → Avoid frequent transfers or overlap computation/comm.
  - → Pre-allocate temporary fields

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    do_some_other_stuff_on_host();

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(tmp[i], i)
  }
```

host
target
host
target
host

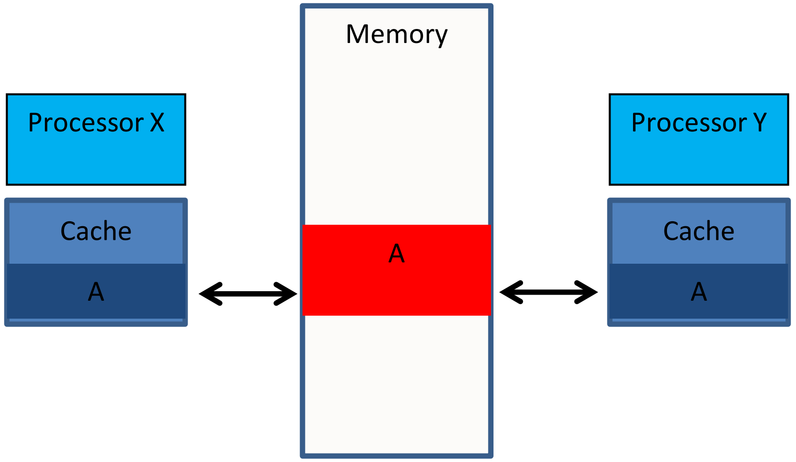
`target data` Construct Example

```
extern void init(float*, float*, int);
extern void init_again(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;

   init(v1, v2, N);

   #pragma omp target data map(from: p[0:N])
   {
      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
          p[i] = v1[i] * v2[i];

      init_again(v1, v2, N);

      #pragma omp target map(to: v1[:N], v2[:N])
      #pragma omp parallel for
      for (i=0; i<N; i++)
          p[i] = p[i] + (v1[i] * v2[i]);
   }

   output(p, N);
}
```

- The `target data` construct creates a *device data environment* and encloses `target` regions, which have their own device data environments.
- The device data environment of the `target data` region is inherited by the device data environment of an enclosed `target` region.
- The `target data` construct is used to create variables that will persist throughout the `target data` region.
- v1 and v2 are mapped at each `target` construct.
- Instead of mapping the variable p twice, once at each `target` construct, p is mapped once by the `target data` construct.

Data mapping: shared or distributed memory

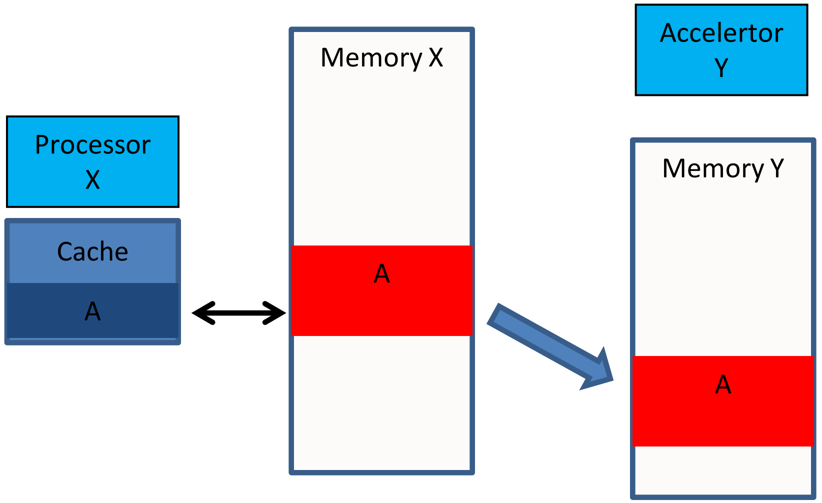
Shared memory



- The corresponding variable in the device data environment *may* share storage with the original variable.

- Writes to the corresponding variable may alter the value of the original variable.

Distributed memory



if Clause Example

```
#define THRESHOLD1 1000000
#define THRESHOLD2 1000

extern void init(float*, float*, int);
extern void output(float*, int);

void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);

   #pragma omp target if(N>THRESHOLD1) \\
          map(to: v1[0:N], v2[:N]) map(from: p[0:N])
   #pragma omp parallel for if(N>THRESHOLD2)
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];
   output(p, N);
}
```

- The `if` clause on the `target` construct indicates that if the variable N is smaller than a given threshold, then the `target` region will be executed by the host device.
- The `if` clause on the `parallel` construct indicates that if the variable N is smaller than a second threshold then the parallel region is inactive.

declare target Constrtuct

- Declare one or more functions to also be compiled for the target device

- Syntax (C/C++):

```
#pragma omp declare target
    [function-definitions-or-declarations]
#pragma omp end declare target
```

- Syntax (Fortran):

```
!$omp declare target [(proc-name-list | list)]
```

Host and device functions

- The tagged functions will be compiled for
  - → Host execution (as usual)
  - → Target execution (to be invoked from offloaded code)

```
#pragma omp declare target
float some_computation(float fl, int in) {
  // ... code ...
}

float final_computation(float fl, int in) {
  // ... code ...
}
#pragma omp end declare target
```

```
some_computation:
   ...
   movups %xmm2, (%r15)
   movups %xmm3, (%rbx)
   ...
final_computation:
   ...
```

host functions

```
some_computation_device:
   ...
   vprefetch0 64(%r15)
   vaddps %zmm7, %zmm6, %zmm9
   ...
final_computation_device:
   ...
```

device functions

Explicit Data Transfers:Target update Construct Example

```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    update_input_array_on_the_host(input);

#pragma omp target update device(0) to(input[:N])

#pragma omp target device(0)
#pragma omp parallel for reduction(+:res)
    for (i=0; i<N; i++)
      res += final_computation(input[i], tmp[i], i)
  }
```

host

target

host

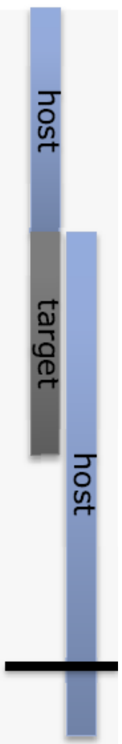
target

host

Asynchronous Offloading

- Use existing OpenMP features to implement asynchronous offloads.

```
#pragma omp parallel sections
{
#pragma omp task
  {
#pragma omp target map(to:input[:N]) map(from:result[:N])
#pragma omp parallel for
    for (i=0; i<N; i++) {
      result[i] = some_computation(input[i], i);
    }
  }
#pragma omp task
  {
    do_something_important_on_host();
  }
#pragma omp taskwait
}
```



Teams Constructs

C/C++

```
#pragma omp teams [clause[[,] clause],...] new-line
    structured-block
```

Fortran

```
!$omp teams [clause[[,] clause],...]
    structured-block
!$omp end teams
```

Clauses:

- **num_teams(** *integer-expression* **)**
- **num_threads(** *integer-expression* **)**
- **default(shared | none)**
- **private(** *list* **)**
- **firstprivate(** *list* **)**
- **shared(** *list* **)**
- **reduction(** *operator* **:** *list* **)**

Restrictions on teams

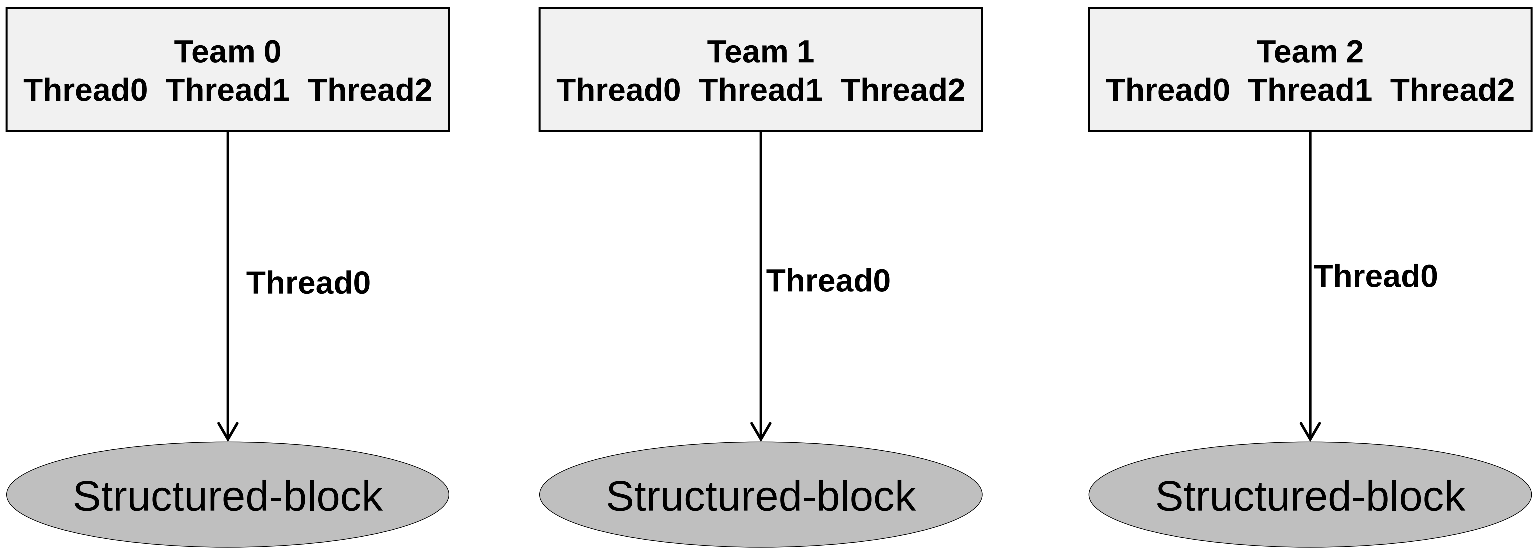
- Creates a league of thread teams
  - → The master thread of each team executes the `teams` region
  - → Number of teams is specified with `num_teams()`
  - → Each team executes `num_threads()` threads
- A `teams` constructs must be "perfectly" nested in a `target` construct:
  - → No statements or directives outside the `teams` construct
- Only special OpenMP constructs can be nested inside a `teams` construct:
  - → `distribute` (see next slides)
  - → `parallel`
  - → `parallel for` (C/C++), `parallel do` (Fortran)
  - → `parallel sections`

Teams Execution Model

Teams Constructs

#pragma omp teams num_teams(3), num_threads(3)

structured-block



SAXPY: Serial (host)

```c
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

SAXPY: Serial (host)

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {









  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
  }
  free(x); free(y); return 0;
}
```

SAXPY:

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
  {
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(nthreads)
```


```
  for (int i = 0; i < n; i += num_blocks){
    for (int j = i; j < i + num_blocks; j++) {
        y[j] = a*x[j] + y[j];
  } }
  }
  free(x); free(y); return 0;
}
```

Distribute Constructs

C/C++

#**pragma omp distribute** *[clause[[,] clause],...] new-line*

for-loops

Fortran

!$omp distribute *[clause[[,] clause],...]*

do-loops

[**!$omp end distribute** *]*

Clauses: **private(** *list* **)**
firstprivate(*list* **)**
collapse(*n* **)**
dist_schedule(*kind[, chunk_size]* **)**

A **distribute** construct must be closely nested in a **teams** region.

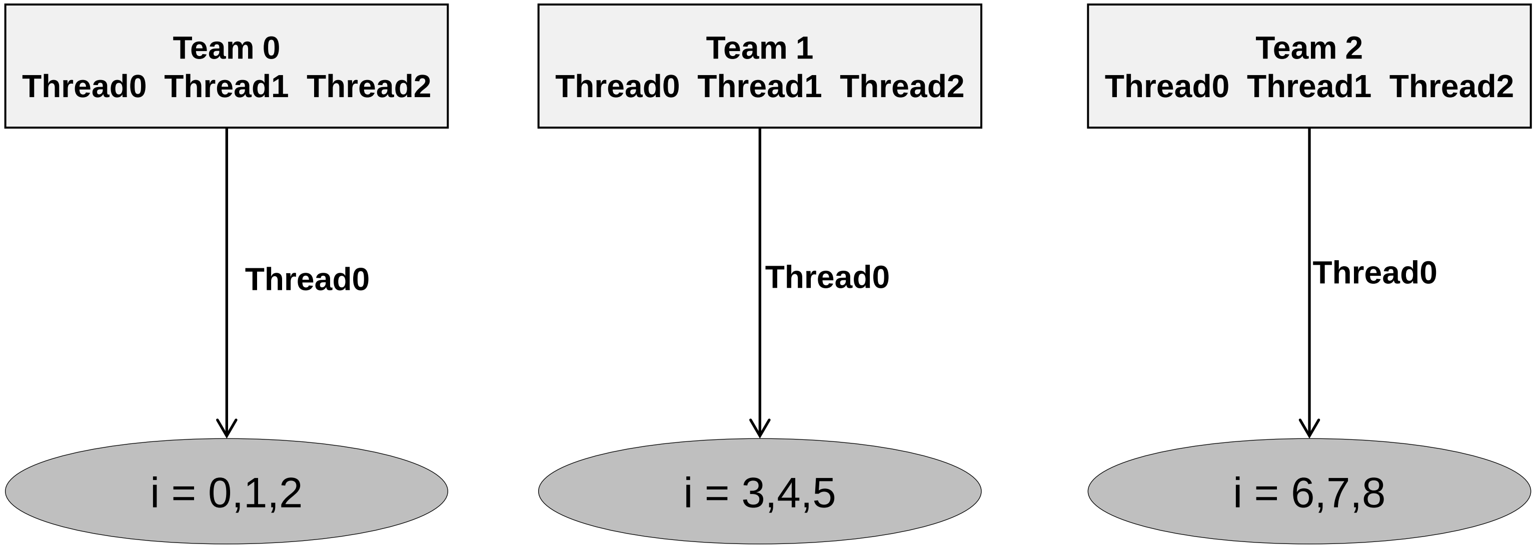
distribute Construct

- New kind of worksharing construct
  - → Distribute the iterations of the associated loops across the master threads of a `teams` construct
  - → No implicit barrier at the end of the construct

- `dist_schedule(`*`kind[, chunk_size]`*`)`
  - → If specified scheduling kind must be static
  - → Chunks are distributed in round-robin fashion of chunks with size *`chunk_size`*
  - → If no chunk size specified, chunks are of (almost) equal size; each team receives at least one chunk

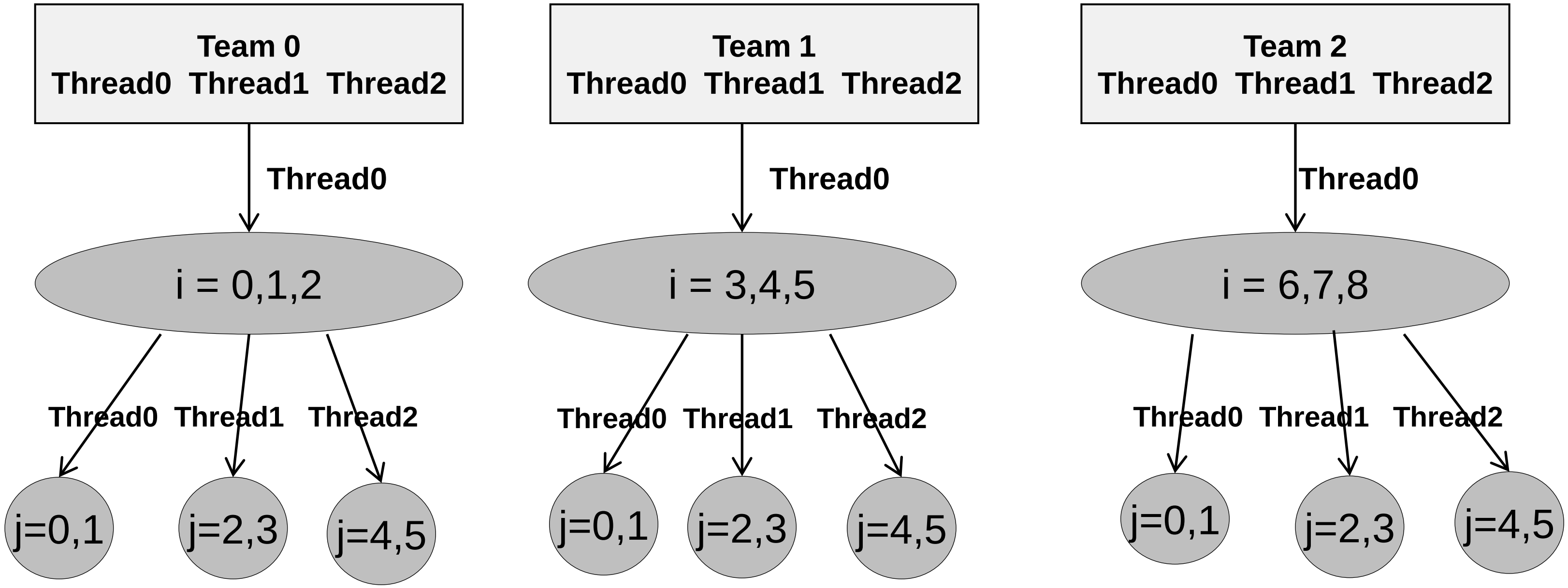
Teams + Distribute Execution Model

```
#pragma omp teams num_teams(3), num_threads(3)
 #pragma omp distribute
    for (int i=0; i<9; i++) {
```



Teams + Distribute Constructs

```
#pragma omp teams num_teams(3), num_threads(3)
 #pragma omp distribute
     for (int i=0; i<9; i++) {
       #   pragma omp parallel for
             for (int j=0;j<6; j++)  {
```

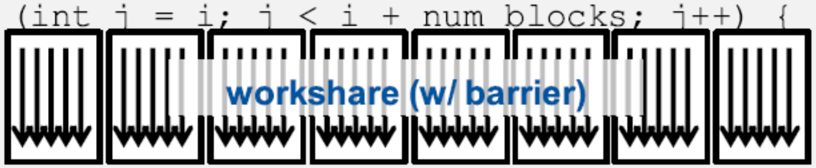


SAXPY: Coprocessor/Accelerator

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
```


```
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
```


```
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
```




```
        y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

Combined Constructs

- The distribution patterns can be cumbersome
- OpenMP 4.0 defines combined constructs for typical code patterns
  - → `distribute simd`
  - → `distribute parallel for` (C/C++)
  - → `distribute parallel for simd` (C/C++)
  - → `distribute parallel do` (Fortran)
  - → `distribute parallel do simd` (Fortran)
  - → ... plus additional combinations for `teams` and `target`
- Avoids the need to do manual loop blocking

SAXPY: Combined Constructs

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target map(to:x[0:n]) map(tofrom:y)
  {
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
#pragma omp distribute parallel for
    for (int i = 0; i < n; ++i){
      y[i] = a*x[i] + y[i];
    }
  }

  free(x); free(y); return 0;
}
```

SAXPY: Combined Constructs

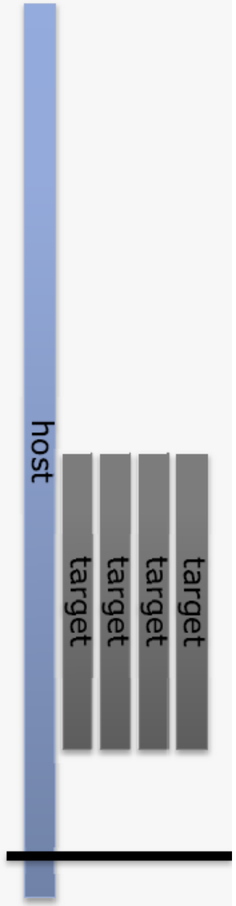
```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target map(to:x[0:n]) map(tofrom:y)
  {
#pragma omp teams distribute parallel for \
       num_teams(num_blocks) num_threads(bsize)
    for (int i = 0; i < n; ++i){
      y[i] = a*x[i] + y[i];
    }
  }

  free(x); free(y); return 0;
}
```

Additional Runtime Support

- Runtime support routines
  - → `void omp_set_default_device(int dev_num )`
  - → `int omp_get_default_device(void)`
  - → `int omp_get_num_devices(void);`
  - → `int omp_get_num_teams(void)`
  - → `int omp_get_team_num(void);`
- Environment variable
  - → Control default device through `OMP_DEFAULT_DEVICE`
  - → Accepts a a non-negative integer value

Multi-device Example

```
int num_dev = omp_get_num_devices();
int chunksz = length / num_dev;
assert((length % num_dev) == 0);
#pragma omp parallel sections firstprivate(chunksz,num_dev)
{
  for (int dev = 0; dev < NUM_DEVICES; dev++) {
#pragma omp task firstprivate(dev)
    {
      int lb = dev * chunksz;
      int ub = (dev+1) * chunksz;
#pragma omp target device(dev) map(in:y[lb:chunksz]) map(out:x[lb:chunksz])
      {
#pragma omp parallel for
        for (int i = lb; i < ub; i++) {
          x[i] = a * y[i];
        }
      }
    }
  }
}
```



OpenACC1 compared to OpenMP 4.0 (by Dr. James Beyer)

OpenACC1

- Parallel (offload)
  - Parallel (multiple "threads")
- Kernels
- Data
- Loop
- Host data
- Cache
- Update
- Wait
- Declare

OpenMP 4.0

- Target
- Team/Parallel
-
- Target Data
- Distribute/Do/for/Simd
-
-
- Target Update
-
- Declare Target

Future OpenACC vs future OpenMP (by Dr. James Beyer)

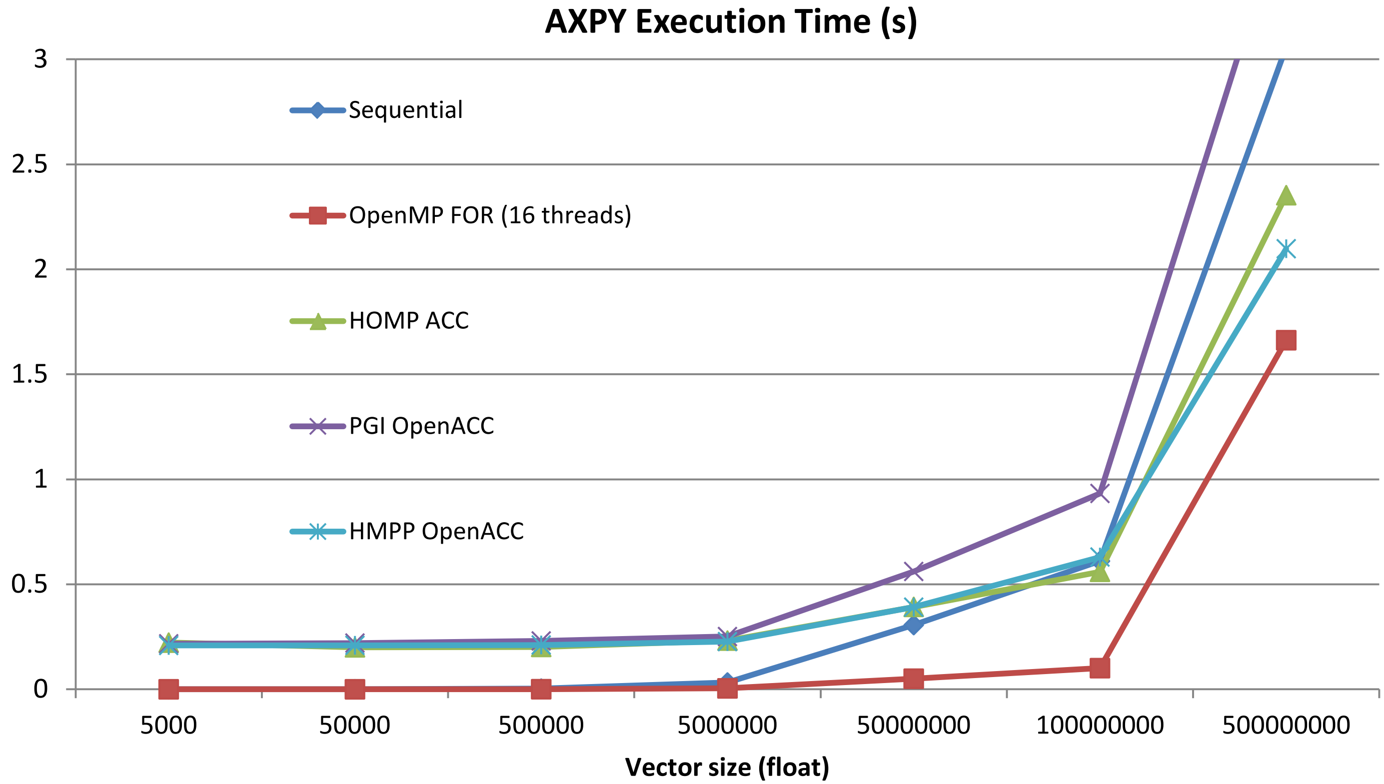
OpenACC2

- enter data
- exit data
- data api
- routine
- async wait
- parallel in parallel
- tile
- Linkable
- Device_type

OpenMP future

- Unstructured data environment
- declare target
-
- Parallel in parallel or team
- tile
- Linkable or Deferred_map
- Device_type

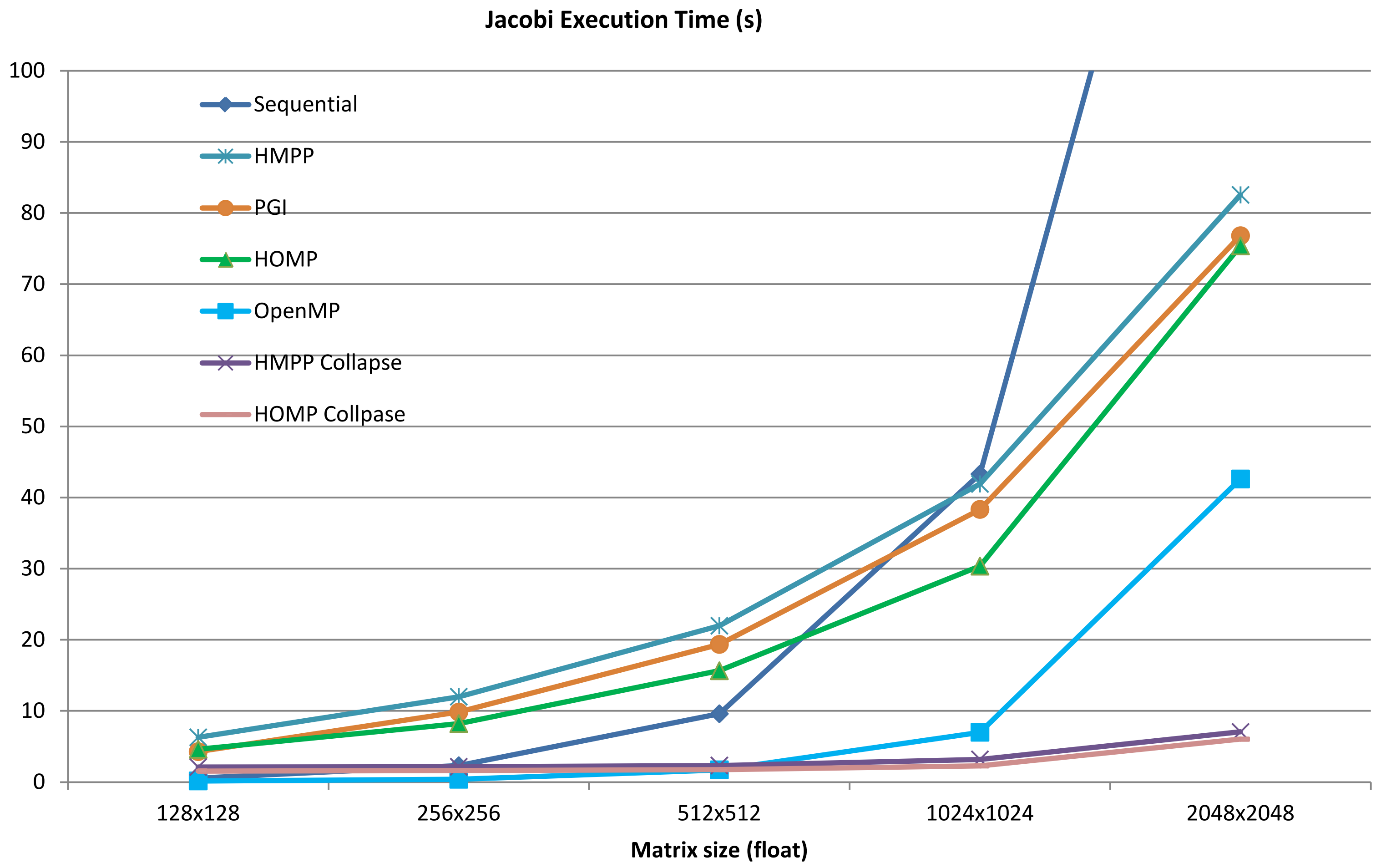
Preliminary results: AXPY (Y=a*X)



Hardware configuration:
- 4 quad-core Intel Xeon processors (16 cores) 2.27GHz with 32GB DRAM.
- NVIDIA Tesla K20c GPU (Kepler architecture)

Software configuration:
- PGI OpenACC compiler version 13.4
- HMPP OpenACC compiler version 3.3.3
- GCC 4.4.7 and the CUDA 5.0 compiler

Jacobi



Jacobi Execution Time (s)
100
90
80
70
60
50
40
30
20
10
0
Sequential
HMPP
PGI
HOMP
OpenMP
HMPP Collapse
HOMP Collpase
128x128
256x256
512x512
1024x1024
2048x2048
Matrix size (float)

Agenda

- What Now?
- OpenMP ARB Corporation
- A Quick Tutorial
- A few key features in 4.0
- Accelerators and GPU Programming
- **Implementation status and Design in clang/llvm**
- The future of OpenMP
- IWOMP 2014 and OpenMPCon 2015

Compilers are here!

- Oracle/Sun Studio 12.4 Beta just announced full OpenMP 4.0
- GCC 4.9 shipped April 9, 2014 supports OpenMP 4.0
- **Clang support for OpenMP injecting into trunk, first appears in 3.5**
- Intel 13.1 compiler supports Accelerators/SIMD
- Cray, TI, IBM coming

OpenMP in Clang update

- I Chaired Weekly OpenMP Clang review WG (Intel, IBM, AMD, TI, Micron) to help speedup OpenMP upstream into clang: April-on going
  - Joint code reviews, code refactoring
  - Delivered Most of OpenMP 3.1 constructs (except atomic and ordered) into Clang 3.5 stream for AST/Semantic Analysis support.
  - have OpenMP –fsyntax-only, Runtime, and basic parallel for loop region to demonstrate code capability
  - Added U of Houston OpenMP tests into clang
  - IBM team Delivered changes for OpenMP RT for PPC, other teams added their platform/architecture
  - Released Joint design on Multi-device target interface for LLVM to llvm-dev for comment
- Future:
  - Clang 3.5 (Sept 2, 2014): Initial support for AST/SEMA for OpenMP 3.1 (except atomic and ordered) + OpenMP library for AMD, ARM, TI, IBM, Intel
  - Clang 3.6 (~Feb 2015): aim for functional codegen of all OpenMP 3.1 + accelerator support(from 4.0)
  - Clang 3.7 (~Sept 2015): aim for full OpenMP 4.0 functional support

Release note commited by me to clang/llvm 3.5

- Clang 3.5 now has parsing and semantic-analysis support for all OpenMP 3.1 pragmas (except atomics and ordered). LLVM's OpenMP runtime library, originally developed by Intel, has been modified to work on ARM, PowerPC, as well as X86. Code generation support is minimal at this point and will continue to be developed for 3.6, along with the rest of OpenMP 3.1. Support for OpenMP 4.0 features, such as SIMD and target accelerator directives, is also in progress. Contributors to this work include AMD, Argonne National Lab., IBM, Intel, Texas Instruments, University of Houston and many others.

Many Participants/companies

- Ajay Jayaraj, TI
- Alexander Musman, Intel
- Alex Eichenberger, IBM
- Alexey Bataev, Intel
- Andrey Bokhanko, Intel
- Carlo Bertolli, IBM
- Eric Stotzer, TI
- Guansong Zhang, AMD
- Hal Finkel, ANL
- Ilia Verbyn, Intel
- James Cownie, Intel
- Kelvin Li, IBM
- Kevin O'Brien, IBM
- Samuel Antao, IBM
- Sergey Ostanevich, Intel
- Sunita Chandrasekaran, UH
- Michael Wong, IBM
- Priya Unikhrishnan, IBM
- Robert Ho, IBM
- Wael Yehia, IBM
- Yan Liu, IBM

Summary of upstream progress of OpenMP clan

- Upstream progress to clang 3.5
  - https://github.com/clang-omp/clang/wiki/Status-of-supported-OpenMP-constructs
- Benchmark OpenMP clang vs OpenMP GCC
  - http://www.phoronix.com/scan.php?page=article&item=llvm_clang_openmp&num=1
    - Unfairly Used –O3 for GCC and noopt for clang
- Link to OpenMP offload infrastructure in LLVM
  - http://lists.cs.uiuc.edu/pipermail/llvmdev/attachments/20140809/cd6c7f7a/attachment-0001.pdf

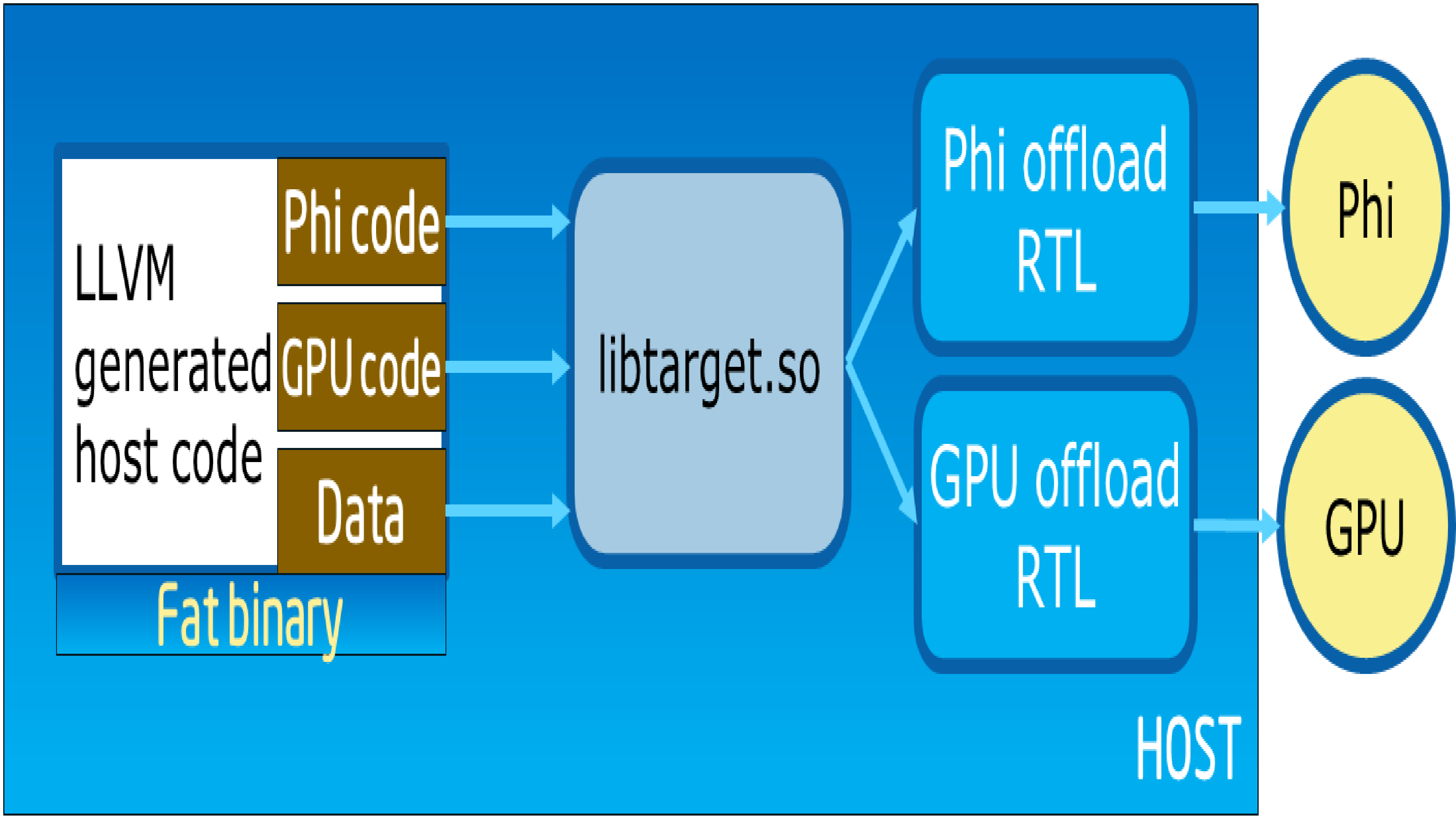
OpenMP offload/target in LLVM

- Samuel Antao (IBM)
- Carlo Bertolli (IBM)
- Andrey Bokhanko (Intel)
- Alexandre Eichenberger (IBM)
- Hal Finkel (Argonne National Laboratory)
- Sergey Ostanevich (Intel)
- Eric Stotzer (Texas Instruments)
- Guansong Zhang (AMD)

Goal of Design

1. support multiple target platforms at runtime and be extensible in the future with minimal or no changes
2. determine the availability of the target platform at runtime and able to make a decision to offload depending on the availability and load of the target platform

Clang/llvm offload design



Example code

```
#pragma omp declare target
int foo(int[1000]);
#pragma omp end declare target
...
int device_count = omp_get_num_devices();
int device_no;
int *red = malloc(device_count * sizeof(int));
#pragma omp parallel
for (i = 0; i < 1000; i++) {
  device_no = i % device_count;
  #pragma omp target device(device_no) map(to:c) map(red[i])
  {
    red[i] += foo(c);
  }
}

for (I = 0; i< device_count; i++)
  total_red = red[i];
```

Generation of fat binary

1. The driver called on a source code should spawn a number of front-end executions for each available target. This should generate a set of object files for each target
2. Target linkers combine dedicated target objects into target shared libraries – one for each target
3. The host linker combines host object files into an executable/shared library and incorporates shared libraries for each target into a separate section within host binary. This process and format is target-dependent and will be thereafter handled by the target RTL at runtime

Agenda

- What Now?
- OpenMP ARB Corporation
- A Quick Tutorial
- A few key features in 4.0
- Accelerators and GPU programming
- Implementation status and Design in clang/llvm
- **The future of OpenMP**
- IWOMP 2014 and OpenMPCon 2015

What did we accomplish in OpenMP 4.0?

- Much broader form of accelerator support
- SIMD
- Cancellation (start of a full error model)
- Task dependencies and task groups
- Thread Affinity
- User-defined reductions
- Initial Fortran 2003
- C/C++ array sections
- Sequentially Consistent Atomics
- Display initial OpenMP internal control variable state

OpenMP future features

- OpenMP Tools: Profilers and Debuggers
  - Just released as TR2
- Consumer style parallelism: event/async/futures
- Enhance Accelerator support/FPGA
  - Multiple device type, linkable to match OpenACC2
- Additional Looping constructs
- Transactional Memory, Speculative Execution
- Task Model refinements
- CPU Affinity
- Common Array Shaping
- Full Error Model
- Interoperability
- Rebase to new C/C++/Fortran Standards, C/C++11 memory model

Agenda

- What Now?
- OpenMP ARB Corporation
- A Quick Tutorial
- A few key features in 4.0
- Accelerators
  - OpenMP and OpenACC
- Affinity
- VectorSIMD
- The future of OpenMP
- **IWOMP 2014 and OpenMPCon 2015**

IWOMP, SC14 and OpenMPCon

- International Workshop on OpenMP
  - 2014 to be held in Brazil
  - A strongly academic conference, with refereed papers, and a Springer-Verlag published proceeding
- SC14
  - Chairing OpenMP Bof, Steering commitete for LLVM in HPC, giving keynote at OpenMP Exhibitor's Forum
- What is missing is a user conference similar to ACCU, pyCON, CPPCON (next week presenting 2 talks), C++Now
- OpenMPCON
  - A user conference paired with IWOMP
  - Non-refereed, user abstracts
  - 1st one will be held in Europe in 2015 to pair with the 2015 IWOMP

OpenMP

OpenMPCon USERS CONVENTION

Hotel Novo... | Sheraton Ri... | Sheraton Ri... | aircanada.c... | Air Canada: ... | AC2U.com -... | OpenMP...

aircaanada

mail - [Omp-error-m... Custom Query – Open... FreeStockCharts.com -... Learning Center Send Free SMS World... (99+) ISO C++ Standa...

HOME | OPENMPCON 2015 | CALL FOR PAPERS | CFP SUBMISSIONS | OPENMP | IWOMP

OpenMPCon

The Event for and by the OpenMP User Community

What is OpenMPCon?

OpenMPCon is the annual, face-to-face gathering, organized by the OpenMP community, for the community. Enjoy keynotes, inspirational talks, and a friendly atmosphere that helps attendees meet interesting people, learn from each other, and have a stimulating experience. Multiple diverse technical tracks are being formulated that will appeal to anyone, from the OpenMP novice to the seasoned expert.

OpenMPCon 2015 | About OpenMP | Be an OpenMPCon Speaker

openmpcon.org/call-for-papers/



IWOMP 2014

September 28-30, 2014

SENAI CIMATEC – Salvador, Brazil





- **Salvador** is the largest city on the northeast coast of Brazil
  - The capital of the Northeastern Brazilian state of Bahia
  - It is also known as Brazil's capital of happiness
- **Salvador** was the first colonial capital of Brazil
  - The city is one of the oldest in the Americas

- **Getting There (SSA):**
  - Direct flights from US (Miami) and Europe (Lisbon, Madrid, & Frankfurt)
  - Alternatively, fly to Rio (GIG) or Sao Paulo (GRU) and connect to Salvador (SSA)

- **Average Temperatures in September:**
  - Average high: 27 °C / 81 °F
  - Daily mean: 25 °C / 77 °F
  - Average low 22 °C / 72 °F

Common-vendor Specification Parallel Programming model on Multiple compilers

AMD, Convey, Cray, Fujitsu, HP, IBM, Intel, NEC, NVIDIA, Oracle, RedHat (GNU), ST Mircoelectronics, TI, clang/llvm

A de-facto Standard: Across 3 Major General Purpose Languages

C++, C, Fortran

A de-facto Standard: One High-Level Accelerator Language

One High-Level Vector SIMD language too!

Support Multiple Devices and let the local compiler generate the best code

Xeon Phi, NVIDIA, GPU, GPGPU, DSP, MIC, ARM and FPGA

My blogs and email address

- **ISOCPP.org Director, VP http://isocpp.org/wiki/faq/wg21#michael-wong OpenMP CEO: http://openmp.org/wp/about-openmp/ My Blogs: http://ibm.co/pCvPHR C++11 status: http://tinyurl.com/43y8xgf Boost test results http://www.ibm.com/support/docview.wss?rs=2239&context=SSJT9L&uid=swg27006911 C/C++ Compilers Feature Request Page http://www.ibm.com/developerworks/rfe/?PROD_ID=700 Chair of WG21 SG5 Transactional MemoryM: https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!forum/tm**

FRAGEN?

Ich freue mich auf Ihr Feedback!

Vielen Dank!

Michael Wong