The Future of GPU/Accelerator Programming Models

LLVM HPC 2015

Michael Wong (IBM)
michaelw@ca.ibm.com; http://wongmichael.com
IBM and Canadian C++ Standard Committee HoD
OpenMP CEO
Chair of WG21 SG5 Transactional Memory, SG14 Games/Low Latency
Director, Vice President of ISOCPP.org
Vice Chair Standards Council of Canada Programming Languages
Acknowledgement and Disclaimer

Numerous people internal and external to the original OpenMP group, 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!
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.

Other company, product, and service names may be trademarks or service marks of others.
Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model
OpenMP Mission Statement changed in 2013

• 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"
OpenMP in Clang update

• I Chair Weekly OpenMP Clang review WG (Intel, IBM, AMD, TI, Micron) to help speedup OpenMP upstream into clang: April 2015-on going
  – Joint code reviews, code refactoring
  – Delivered full OpenMP 3.1 into Clang 3.7 (default lib is still GCC OpenMP)
  – 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

– LLVM developer Conf Oct 2015 talk:
  • http://llvm.org/devmtg/2015-10/slides/WongBataev-OpenMPGPUAcceleratorsComingOfAgeInClang.pdf
  • https://www.youtube.com/watch?v=dCdOaL3asx8&list=PL_R5A0lGi1AA4Lv2bBFSwhgDaHvypVU21&index=18
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
• Yaoqing Gao, IBM
• Kelvin Li, IBM
• Kevin O’Brien, IBM
• Samuel Antao, IBM
• Sergey Ostanevich, Intel
• Sunita Chandrasekaran, UH
• Michael Wong, IBM
• Wang Chan, IBM
• Robert Ho, IBM
• Wael Yehia, IBM
• Ettore Tiotto, IBM
• Melanie Ullmer, IBM
• Kevin Smith, Intel
The codebase

LLVM main repository
http://llvm.org

- Version 3.5
- Version 3.7
- Version 3.8 Trunk

Clang-OMP repository
http://clang-omp.github.io

- Initial version
- Current version

Clang/LLVM snapshot

- All OpenMP 3.1 merged
- Now merging OpenMP 4.0
- Added OpenMP features to Clang

How to use it:

- Grab the latest source files and **install LLVM as usual**
- Use the right options to **specify host and target** machines, e.g.:

  $ clang -fopenmp -target powerpc64le-ibm-linux-gnu -mcpu pwr8
  -omptargets=nvptx64sm_35-nvidia-cuda <source files>
Offloading in OpenMP – Impl. components

Input Program
C/C++

OpenMP enabled compiler

Fat binary
Host code
Device code

Device runtime library

Host runtime library
Host component
Target agnostic component
Target API

Operating System
Device Driver

Host machine

Device Driver

Device
Clang with OpenMP

- **Compiler actions:**
  - **Driver** preprocesses input source files using **host/target preprocessor**
    - Header files may be in different places
    - We may revisit this in the future
  - For each source file, the driver spawns a **job using the host toolchain and an additional job for each target** specified by the user
  - Flags informing the frontend that we are compiling code **for a target so only the relevant target regions are considered**
  - **Target linker creates a self-contained** (no undefined symbols) image file
  - **Target image file is embedded “as is”** by the host linker into the host fat binary
  - The **host linker** is provided with information to generate the symbols required by the RTL
Offloading in Clang: Current Status

• Initial implementation available at https://github.com/clang-omp/clang_trunk

• First patches are committed to trunk
  – Support for target constructs parsing/sema/codegen for host

• Several patches are under review
  – Support for new driver option
  – Offloading descriptor registration and device codegen
heterogeneous device model

- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  - one host
  - multiple accelerators / coprocessors of the same kind
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.
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 | to/from | 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
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;
}
```
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;
}
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;
}
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++) {
            workshare (w/o barrier)
            y[j] = a*x[j] + y[j];
        }
    }
    workshare (w/ barrier)
}
free(x); free(y); return 0; }
Building Fat Binary

• Clang generates objects for each target
• Target toolchains combine objects into target-dependent binaries
• Host linker combines host + target-dependent binaries into an executable (Fat Binary)
• New driver command-line option -omptargets=T1,...,Tn

```
clang -fopenmp -omptargets=nvptx64-nvidia-cuda,x86-pc-linux-gnu foo.c bar.c -o foobar.bin
```
Heterogeneous Execution of Fat Binary

LLVM Generated host code
  - Data
  - Xeon Phi Code
  - GPU Code
  - DSP Code

Fat Binary

libomptarget library

- Xeon Phi offload RTL
- GPU offload RTL
- DSP offload RTL

Xeon Phi

GPU

DSP
Libomptarget and offload RTL

• Source code available at https://github.com/clang-omp/libomptarget

• Planned to be upstreamed

• Supported platforms
  – libomptarget
    • Platform neutral implementation (tested on Linux for x86-64, PowerPC*)
    • NVIDIA* (Tested with CUDA* compilation tools V7.0.27)
  – Offload target RTL
    • x86-64, PowerPC, NVIDIA

*Other names and brands may be claimed as the property of others.
What did we learn?

• Multi-Vendor/University collaboration works even outside of ISO
• Support separate vendor-dependent target RTL to enable other programming models
• Production compilers need support for L10N and I18N for multiple countries and languages
Future plans

• Clang 3.8 (~Feb, 2016): trunk switches to clang OpenMP lib, upstream OpenMP 4.0 with focus on Accelerator delivery; start code dropping for OpenMP 4.5
• Clang 3.9 (~Aug 2016): Complete OpenMP 4.0 and continue to Add OpenMP 4.5 functionality
• Clang 4.0 (~Feb 2017): clang/llvm becomes reference compiler; follow OpenMP ratification with collaborated contribution?
Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model
What now?

• 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
  – 160 pages and growing
• OpenMP 4.0 is
  – 320 pages
• OpenMP 4.5 is
  – 359 pages
A tale of two cities
Will the two galaxies ever join?
OH, East is East, and West is West,
and never the twain shall meet...
-Rudyard Kipling
What did we learn from the OpenMP Accelerator model?

• Consumer threads needed
• More concurrency controls needed
• Excellent HPC domain usage
• Some use in financials
• but almost none in consumers and commercial applications
• C++ support? Can it get better?
Its like the difference between:

An Aircraft Carrier Battle Group (ISO)
And a Cruiser (Consortium: OpenMP)
And a Destroyer (Company Specific language)
C++ support for Accelerators

- Memory allocation
- Templates
- Exceptions
- Polymorphism
- Current Technical Specifications
  - Concepts, Parallelism, Concurrency, TM
Programming GPU/Accelerators

- OpenGL
- DirectX
- CUDA
- OpenCL
- OpenMP
- OpenACC
- C++ AMP
- HPX
- HSA
- SYCL
- Vulkan
- A preview of C++
  - WG21 Accelerator model SG1/SG14 TS2
  - (SC15 LLVM HPC talk)
texture<float, 2, cudaReadModeElementType> tex;
void foo() {
  cudaArray* cu_array;
  // Allocate array
  cudaChannelFormatDesc description = cudaCreateChannelDesc<float>();
  cudaMemcpyArray(&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
}
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];
    });
}
C++1Y (1Y=17 or 22) Concurrency Plan

**Parallelism**
- Parallel STL Algorithms:
  - Data-Based Parallelism. (Vector, SIMD, ...)
- Task-based parallelism (cilk, OpenMP, fork-join)
- MapReduce
- Pipelines

**Concurrency**
- Future Extensions (then, wait_any, wait_all):
- Executors:
- Resumable Functions, await (with futures)
- Counters
- Queues
- Concurrent Vector
- Unordered Associative Containers
- Latches and Barriers
- upgrade_lock
- Atomic smart pointers
<table>
<thead>
<tr>
<th>Project</th>
<th>What’s in it?</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filesystem TS</td>
<td>Standard filesystem interface</td>
<td>Published!</td>
</tr>
<tr>
<td>Library Fundamentals TS I</td>
<td>optional, any, string_view and more</td>
<td>Published!</td>
</tr>
<tr>
<td>Library Fundamentals TS II</td>
<td>source code information capture and various utilities</td>
<td>Voted out for balloting by national standards bodies</td>
</tr>
<tr>
<td>Concepts (“Lite”) TS</td>
<td>Constrained templates</td>
<td>Publication imminent</td>
</tr>
<tr>
<td>Parallelism TS I</td>
<td>Parallel versions of STL algorithms</td>
<td>Published!</td>
</tr>
<tr>
<td>Parallelism TS II</td>
<td>TBD. Exploring task blocks, progress guarantees, SIMD</td>
<td>Under active development</td>
</tr>
<tr>
<td>Transactional Memory TS</td>
<td>Transactional Memory TS</td>
<td>Published!</td>
</tr>
<tr>
<td>Project</td>
<td>What’s in it?</td>
<td>Status</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Concurrency TS I</td>
<td>improvements to future, latches and barriers, atomic smart pointers</td>
<td>Voted out for publication!</td>
</tr>
<tr>
<td>Concurrency TS II</td>
<td>TBD. Exploring executors, synchronic types, atomic views, concurrent data structures</td>
<td>Under active development</td>
</tr>
<tr>
<td>Networking TS</td>
<td>Sockets library based on Boost.ASIO</td>
<td>Design review completed; wording review of the spec in progress</td>
</tr>
<tr>
<td>Ranges TS</td>
<td>Range-based algorithms and views</td>
<td>Design review completed; wording review of the spec in progress</td>
</tr>
<tr>
<td>Numerics TS</td>
<td>Various numerical facilities</td>
<td>Beginning to take shape</td>
</tr>
<tr>
<td>Array Extensions TS</td>
<td>Stack arrays whose size is not known at compile time</td>
<td>Direction given at last meeting; waiting for proposals</td>
</tr>
<tr>
<td>Reflection</td>
<td>Code introspection and (later) reification mechanisms</td>
<td>Still in the design stage, no ETA</td>
</tr>
<tr>
<td>Project</td>
<td>What’s in it?</td>
<td>Status</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Graphics</td>
<td>2D drawing API</td>
<td>Waiting on proposal author to produce updated standard wording</td>
</tr>
<tr>
<td>Modules</td>
<td>A component system to supersede the textual header file inclusion model</td>
<td>Microsoft and Clang continuing to iterate on their implementations and converge on a design. The feature will target a TS, not C++17.</td>
</tr>
<tr>
<td>Coroutines</td>
<td>Resumable functions</td>
<td>At least two competing designs. One of them may make C++17.</td>
</tr>
<tr>
<td>Contracts</td>
<td>Preconditions, postconditions, etc.</td>
<td>In early design stage</td>
</tr>
</tbody>
</table>
Agenda

- Clang/OpenMP Multi-company collaboration
- What Now?
- SG14
- C++ Std GPU Accelerator Model
The Birth of Study Group 14

Towards Improving C++ for Games & Low Latency
Among the top users of C++!

About SG14

1. About SG14
2. Control & Reliability
3. Metrics & Performance
4. Fun & Productivity
5. Current Efforts
6. The Future
The Breaking Wave: N4456

CppCon 2014

C++ committee panel leads to impromptu game developer meeting.

Google Group created.

Discussions have outstanding industry participation.

N4456 authored and published!

| N4456 | Towards improved support for games, graphics, real-time, low latency, embedded systems |
Formation of SG14

N4456 presented at Spring 2015 Standards Committee Meeting in Lenexa.

Very well received!

Formation of Study Group 14: Game Dev & LowLatency

Chair: Michael Wong (IBM)

Two SG14 meetings planned:
- CppCon 2015 (this Wednesday)
- GDC 2016, hosted by SONY
Improving Communication/Feedback/review cycle

Industry members come to CppCon/GDC

Standard C++ Committee members come to CPPCon/GDC (hosted by SONY)

SG14 approved papers are presented by C++ Committee members at Standard meetings for voting

Feedback goes back through SG14 to industry for revision

Rinse and repeat
The Industry name linkage brings in lots of people

- The First Industry-named SG that gains connection with
  - Games
  - Financial/Trading
  - Banking
  - Simulation
  - +HPC/Big Data Analysis?
Audience of SG14 Goals and Scopes: Not just games!

- Video Games

**Shared Common Interest**

Better support in C++ for:

- Interactive Simulation
- Real-time Graphics
- Low Latency Computation
- Constrained Resources
- HPC/BigData Analytic workload

- Simulation and Training Software
- Finance/Trading
- Embedded Systems
Where We Are

Google Groups

https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/s g14

GitHub

https://github.com/WG21-SG14/SG14

Created by Guy Davidson
SG14 are interested in following these proposals

- **GPU/Acccelerator support**
- ** Executors**
  - 3 ways: low-latency, parallel loops, server task dispatch
- ** Atomic views**
- ** Coroutines**
- ** noexcept library additions**
  - Use `std::error_code` for signaling errors
- **Early SIMD in C++ investigation**
  - There are existing SIMD papers suggesting eg. “Vec<T,N>” and “for simd (;;)”

- **Array View**
- ** Node-based Allocators**
- **String conversions**
- **hot set**
- **vector and matrix**
- **Exception and RTTI costs**
- **Ring or circular buffers**
- **Flat_map**
- **Intrusive containers**
- **Allocator interface**
- **Radix sort**
  - Spatial and geometric algorithms
  - Imprecise but faster alternatives for math algorithms
  - Cache-friendly hash table
  - Contiguous containers
  - Stack containers
  - Fixed-point numbers
  - `plf::colony` and `plf::stack`
Agenda

• Clang/OpenMP Multi-company collaboration
• What Now?
• SG14
• C++ Std GPU Accelerator Model
C++ Standard GPU/Accelerators

• Attended by both National Labs and commercial/consumers
• Glimpse into the future
• No design as yet, but several competing design candidates
• Offers the best chance of a model that works across both domains for C++ (only)
Grand Unification?
```cpp
#include <thread>
#include <iostream>

void func()
{
    std::cout << "**Inside thread "
    << std::this_thread::get_id() << "!" << std::endl;
}

int main()
{
    std::thread t;
    t = std::thread(func);
    t.join();
    return 0;
}
```

**“Hello World” with std::thread**

A simple function for thread to do...

Create and schedule thread...

Wait for thread to finish...
#include <thread>
#include <iostream>

void func()
{
    std::cout << "**Hello world...\n";
}

int main()
{
    std::thread t;
    t = std::thread( func );
    t.join();
    return 0;
}

(1) Thread function must do exception handling; unhandled exceptions ==> termination...

(2) Must join, otherwise termination...

NOTE: avoid use of detach() in C++11, difficult to use safely.
Example: saxpy

- Saxpy == *Scalar Alpha X Plus Y*

- Scalar multiplication and vector addition

```c
for (int i=0; i<n; i++)
    z[i] = a * x[i] + y[i];
```

```c
int start = ...;
int end   = ...;
for (int t=0; t<NumThreads; t++)
{
    thread(
        [&z,x,y,a,start,end]()->void
    {
        for (int i = start; i < end; i++)
            z[i] = a * x[i] + y[i];
    });
    start += ...;
    end   += ...;
}
```
Sequential Matrix Multiplication

// Naïve, triply-nested sequential solution:
//
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        C[i][j] = 0.0;

        for (int k = 0; k < N; k++)
            C[i][j] += (A[i][k] * B[k][j]);
    }
}
Structured ("fork-join") parallelism
• A common pattern when creating multiple threads

```cpp
#include <vector>
std::vector<std::thread> threads;
int cores = std::thread::hardware_concurrency();
for (int i=0; i<cores; ++i) // 1 per core:
{
    auto code = [](()){ DoSomeWork(); };
    threads.push_back(thread(code));
}
for (std::thread& t : threads) // new range-based for:
    t.join();
```

Going Parallel with C++11 by Joe Hummel
Parallel solution

```c++
int rows = N / numthreads;
int extra = N % numthreads;
int start = 0; // each thread does [start..end)
int end = rows;
vector<thread> workers;
for (int t = 1; t <= numthreads; t++)
{
    if (t == numthreads) // last thread does extra rows:
        end += extra;
    workers.push_back(thread([start, end, N, &C, &A, &B]() {
        for (int i = start; i < end; i++)
            for (int j = 0; j < N; j++)
            {
                C[i][j] = 0.0;
                for (int k = 0; k < N; k++)
                    C[i][j] += (A[i][k] * B[k][j]);
            }
    }));
    start = end;
    end = start + rows;
}

for (thread& t : workers)
    t.join();
```

// 1 thread per core:
numthreads = thread::hardware_concurrency();

Going Parallel with C++11 by Joe Hummel
What does C++ Standard parallelism still need?

- Parallelism alone is not enough for HPC...

\[ HPC \equiv \text{Parallelism} + \text{Memory Hierarchy} - \text{Contention} \]

Exposé parallelism

Maximize data locality:
- network
- disk
- RAM
- cache
- core

Minimize interaction:
- false sharing
- locking
- synchronization

Going Parallel with C++11 by Joe Hummel
Asynchronous Calls

• Building blocks:
  – `std::async`: Request asynchronous execution of a function.
  – `Future`: token representing function’s result.

• Unlike raw use of `std::thread` objects:
  – Allows values or exceptions to be returned.
    • Just like “normal” function calls.
WHAT IS A (THE) FUTURE

- A future is an object representing a result which has not been calculated yet
  - Enables transparent synchronization with producer
  - Hides notion of dealing with threads
  - Makes asynchrony manageable
  - Allows for composition of several asynchronous operations
  - (Turns concurrency into parallelism)
WHAT IS A (THE) FUTURE?

- Many ways to get hold of a future, simplest way is to use (std) async:

```cpp
int universal_answer() { return 42; }

void deep_thought()
{
    future<int> promised_answer = async(&universal_answer);

    // do other things for 7.5 million years

    cout << promised_answer.get() << endl;  // prints 42, eventually
}
```
WAYS TO CREATE A FUTURE

- Standard defines 3 possible ways to create a future,
  - 3 different *asynchronous providers*
    - std::async
      - See previous example, std::async has caveats
    - std::packaged_task
    - std::promise
Standard Concurrency Interfaces

- `std::async<>` and `std::future<>`: concurrency as with sequential processing
  - one location calls a concurrent task and dealing with the outcome is as simple as with local sub-functions
- `std::thread`: low-level approach
  - one location calls a concurrent task and has to provide low-level techniques to handle the outcome
- `std::promise<>` and `std::future<>`: Simplify processing the outcome
  - one location calls a concurrent task but dealing with the outcome is simplified
- `packaged_task<>`: helper to separate task definition from call
  - one location defines a task and provides a handle for the outcome
  - another location decides when to call the task and the arguments
  - the call must not necessarily happen in another thread
std::async + std::future

- Use `async` to start asynchronous operation
- Use returned `future` to wait upon result / exception

```cpp
#include <future>

std::future<int> f = std::async([]() -> int {
    int result = PerformLongRunningOperation();
    return result;
});

try {
    int x = f.get(); // wait if necessary, harvest result:
    cout << x << endl;
} catch(exception &e) {
    cout << "**Exception: " << e.what() << endl;
}
```

Going Parallel with C++11 by Joe Hummel
Async operations

• Run on current thread *or* a new thread
• By default, system decides...

```cpp
// runs on current thread when you “get” value (i.e. lazy execution):
future<T> f1 = std::async( std::launch::deferred, []() -> T { ... } );

// runs now on a new, dedicated thread:
future<T> f2 = std::async( std::launch::async, []() -> T { ... } );

// let system decide (e.g. maybe you created enough work to keep system busy?):
future<T> f3 = std::async( ^[]( ) -> T { ... } );

 optional argument missing
```
Commercial application

- Netflix data-mining...

Netflix Movie Reviews (.txt) → Netflix Data Mining App → Average rating...
Sequential solution

```
cin >> movieID;

vector<string> ratings = readFile("ratings.txt");

tuple<int, int> results = dataMine(ratings, movieID);

int numRatings = std::get<0>(results);
int sumRatings = std::get<1>(results);
double avgRating = double(numRatings) / double(sumRatings);

cout << numRatings << endl;
cout << avgRating << endl;
```

```
dataMine(vector<string> &ratings, int id) {
    foreach rating
        if ids match num++, sum += rating;

    return tuple<int, int>(num, sum);
}
```
Parallel solution

```cpp
int chunksize = ratings.size() / numthreads;
int leftover = ratings.size() % numthreads;
int begin = 0; // each thread does [start..end)
int end = chunksize;

vector<future<tuple<int,int>>> futures;
for (int t = 1; t <= numthreads; t++)
{
    if (t == numthreads) // last thread does extra rows:
        end += leftover;

    futures.push_back(
        async([&ratings, movieID, begin, end]() -> tuple<int,int>
        {
            return dataMine(ratings, movieID, begin, end);
        }));

    begin = end;
    end = begin + chunksize;
}

for (future<tuple<int,int>> &f: futures)
{
    tuple<int, int> t = f.get();
    numRatings += std::get<0>(t);
    sumRatings += std::get<1>(t);
}
```

Going Parallel with C++11 by Joe Hummel
Other things C++ need: Types of parallelism

• Most common types:
  – Data: coming in SIMD proposal
  – Task: coming in executors and task blocks
  – Embarrassingly parallel: async and threads
  – Dataflow: Concurrency TS (.then)
EXTENDING STD::FUTURE

• Several proposals (draft technical specifications) for next C++ Standard
  • Extension for future<>
    • Compositional facilities
      • Parallel composition
      • Sequential composition
  • Parallel Algorithms
  • Parallel Task Regions
• Extended async semantics: dataflow
MAKE A READY FUTURE

• Create a future which is ready at construction (N3857)

```cpp
future<int> compute(int x)
{
    if (x < 0) return make_ready_future<int>(-1);
    if (x == 0) return make_ready_future<int>(0);

    return async([](int par) { return do_work(par); }, x);
}
```
COMPOSITIONAL FACILITIES

- Sequential composition of futures (see N3857)

```cpp
string make_string()
{
    future<int> f1 = async([]() -> int { return 123; });
    future<string> f2 = f1.then([](future<int> f) -> string {
        return to_string(f.get()); // here .get() won’t block
    });
}
```
COMPOSITIONAL FACILITIES

- Parallel composition of futures (see N3857)

```cpp
void test_when_all() {
    shared_future<int> shared_future1 = async([]() -> int { return 125; });
    future<string> future2 = async([]() -> string { return string("hi"); });

    future<tuple<shared_future<int>, future<string>>> all_f =
        when_all(shared_future1, future2); // also: when_any, when_some, etc.

    future<int> result = all_f.then(
        [](future<tuple<shared_future<int>, future<string>>> f) -> int {
            return do_work(f.get());
        });
}
```
PARALLEL ALGORITHMS

- Parallel algorithms (N4071)
  - Mostly, same semantics as sequential algorithms
  - Additional, first argument: execution_policy (seq, par, etc.)
- Extension
  - task_execution_policy
  - Algorithm returns future

Asynchronous Computing in C++ by Hartmut Kaiser
EXTENDING ASYNC: DATAFLOW

- What if one or more arguments to ‘async’ are futures themselves?
- Normal behavior: pass futures through to function
- Extended behavior: wait for futures to become ready before invoking the function:

```cpp
template <typename F, typename... Arg>
future<typename result_of<F(Arg...>::type> dataflow(F&& f, Arg&&... arg);
```

- If ArgN is a future, then the invocation of F will be delayed
- Non-future arguments are passed through
C++ Std+ proposals already have many features for accelerators

• Asynchronous tasks (C++11 futures plus C++17 `then, when`, `is_ready`,...)
• Parallel Algorithms
• Executors
• Multi-dim arrays, Layouts
Candidates to C++ Std Accelerator Model

• C++AMP
  – Restrict keyword is a mistake
  – GPU Hardware removing traditional hurdles
  – Modern GPU instruction sets can handle nearly full C++
  – Memory systems evolving towards single heap
Better candidates

• Goal: Use standard C++ to express all intra-node parallelism
  – Agency extends Parallelism TS
  – HCC
  – SYCL extends Parallelism TS
Food for thought and Q/A

• C11/C++14 Standards
  – C++: http://www.open-std.org/jtc1/sc22/wg21/prot/14882fdis/n3937.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:
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 Memory:
  https://groups.google.com/a/isocpp.org/forum/?hl=en&fromgroups#!forum/tm
• Chair of WG21 SG14 Games Dev/Low Latency:
  https://groups.google.com/a/isocpp.org/forum/?fromgroups#!forum/s g14