OpenMP GPU/Accelerators Coming of Age in Clang

LLVM Developer Conference Oct 2015

Michael Wong (IBM), Alexey Bataev (Intel)

Andrey Bokhanko (Intel), Alexandre Eichenberger(IBM), Carlo Bertolli (IBM), Eric Stotzer (TI), Guansong Zhang (AMD), Hal Finkel (ANL), Kelvin Li (IBM), Kevin O'Brien (IBM), Samuel Antao (IBM), Sergey Ostanevich (Intel), Ravi Narayanaswamy (Intel), Sunita Chandrasekaran (University of Delaware)

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

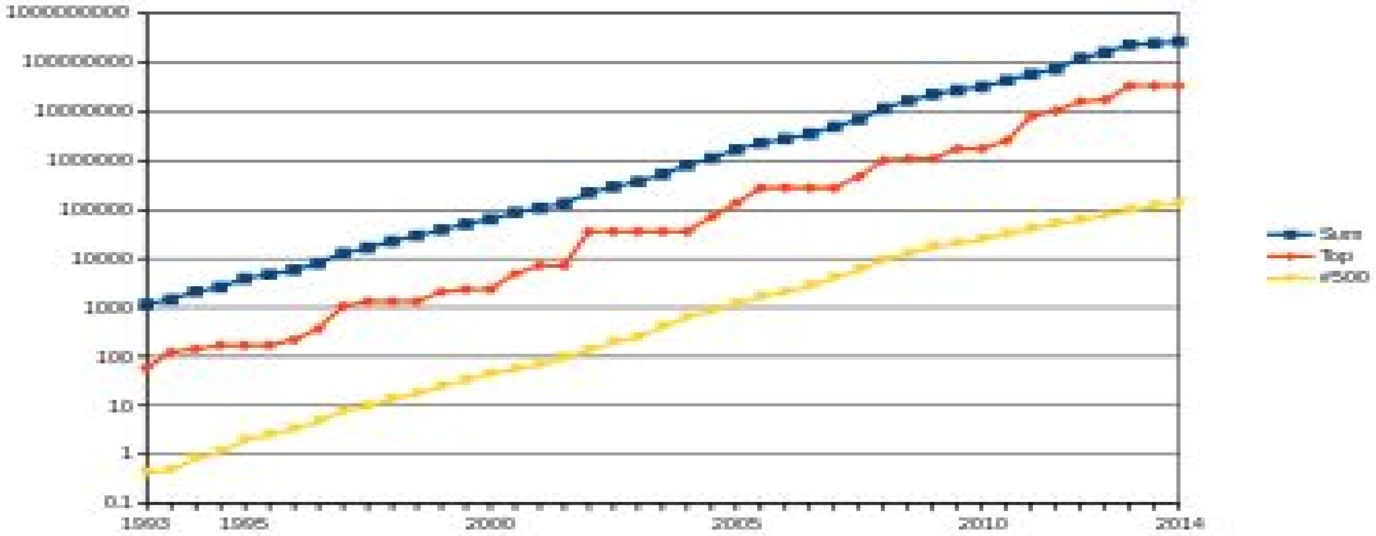
Agenda

- Accelerator Programming
- OpenMP 4.0 Accelerator Programming Model
- Clang/OpenMP Target-independent Offload Design
- Clang/OpenMP Offloading in Action
- Users of OpenMP-enabled clang

WORLD OF WARSHIPS
WARGAMING.NET

So, how do you program GPU?

Why is GPU important now?

- Or is it a flash in the pan?
- The race to exascale computing .. 10 ^{18} flops
- 

Top500 contenders

Programming GPU/Accelerators

- OpenGL
- DirectX
- CUDA
- OpenCL
- OpenMP
- OpenACC
- C++ AMP
- HSA
- SYCL
- Vulcan
- Soon a preview of C++ WG21 Parallelism SG1/SG14 TS2 (SC15 LLVM HPC talk)

WG21 SG1 Parallelism TS1

```
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(par_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());
```

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)

What is OpenMP Model’s aim?

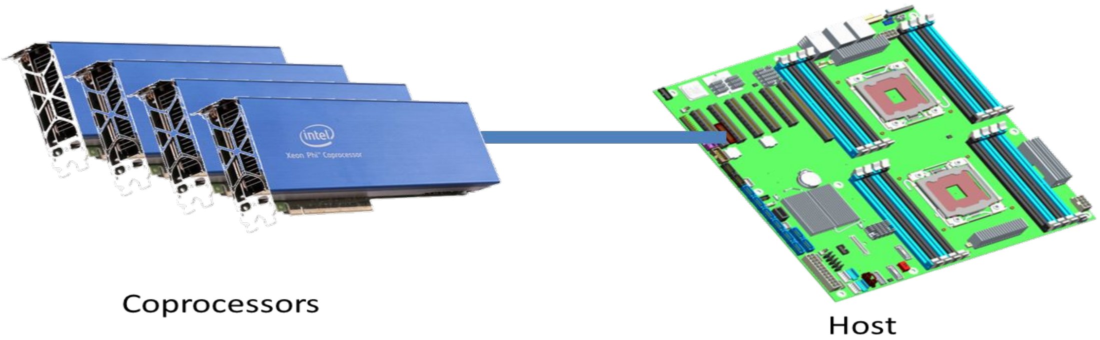
- All forms of accelerators, DSP, GPU, APU, GPGPU
- Network heterogenous consumer devices
  - Kitchen appliances, drones, signal processors, medical imaging, auto, telecom, automation, not just graphics engines

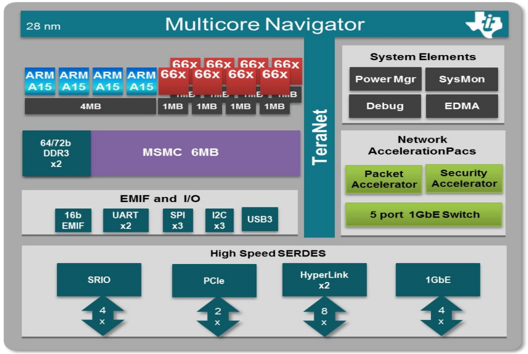
Agenda

- Accelerator Programming
- OpenMP 4.0 Accelerator Programming Model
- Clang/OpenMP Target-independent Offload Design
- Clang/OpenMP Offloading in Action
- Users of OpenMP-enabled clang

heterogeneous device model

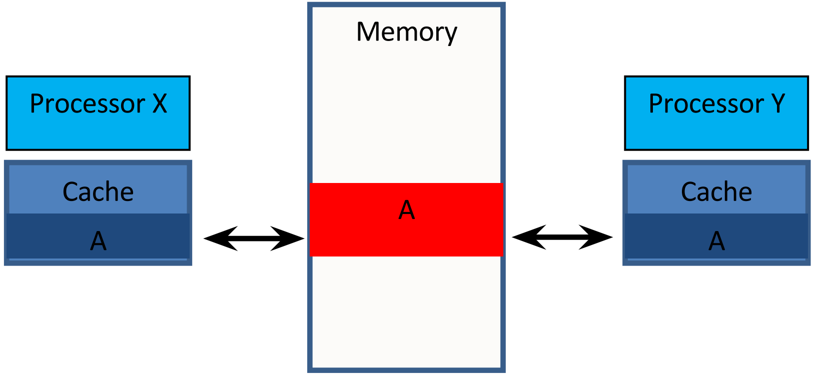
- OpenMP 4.0 supports accelerators/coprocessors
- Device model:
  – one host
  – multiple acclerators / coprocessors of the same kind





Data mapping: shared or distributed memory

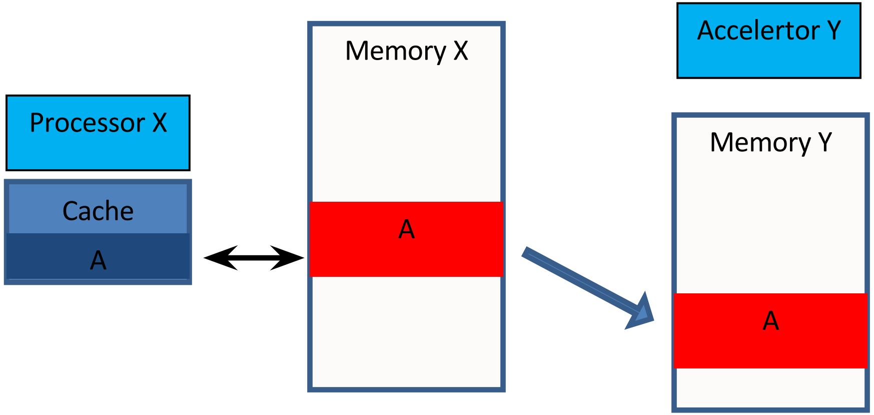
Shared memory



- 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.

Distributed memory



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

device constructs

terminology

- **device**: An implementation defined logical execution engine. (A device could have one or more processors.)
- **host device**: The device on which the OpenMP program begins execution.
- **target device**: A device onto which code and data may be offloaded from the host device. It is implementation defined (i.e. optional).
- **league**: The set of threads teams created by a teams construct.

device constructs – device data model

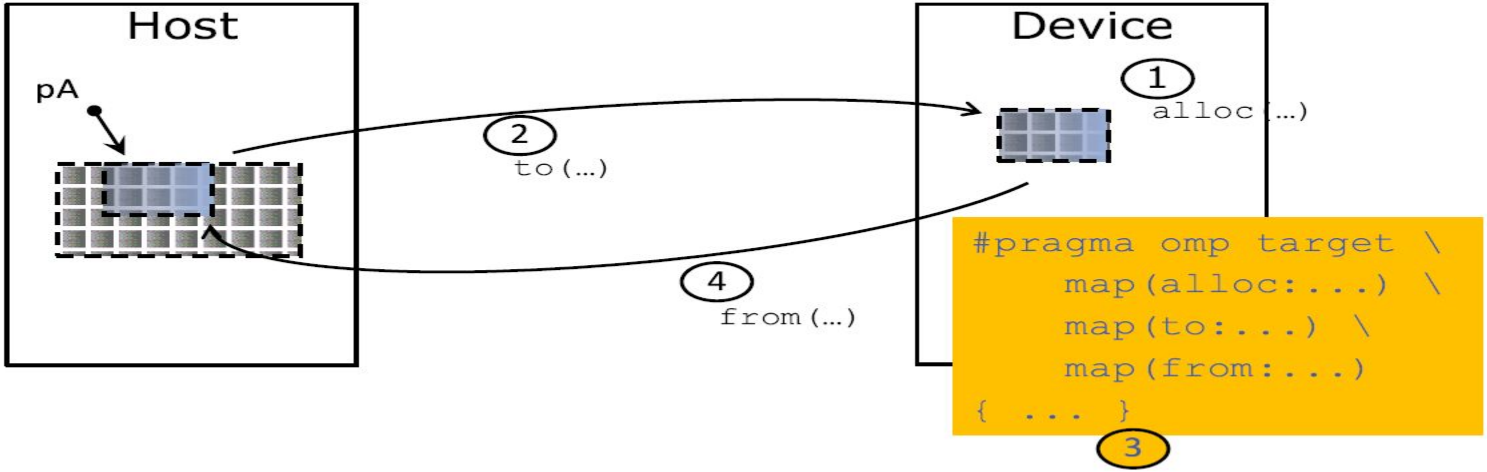
- very important to get the data right on host and target devices
- possible to be out of sync or unavailable
- data can implicitly be mapped to the target region
- data can explicitly (via data-mapping attribute clause) mapped to the target region
  - provide more precise information to the compiler
  - reduce unnecessary data transfer – that is expensive!

device constructs - execution model and data environment

- the **`target`** construct transfers the control flow to the target device
  - the map 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 map 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

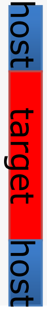


```
#pragma omp target \
    map(alloc:...) \
    map(to:...) \
    map(from:...)
{ ... }
```

`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;
   }
 }
```



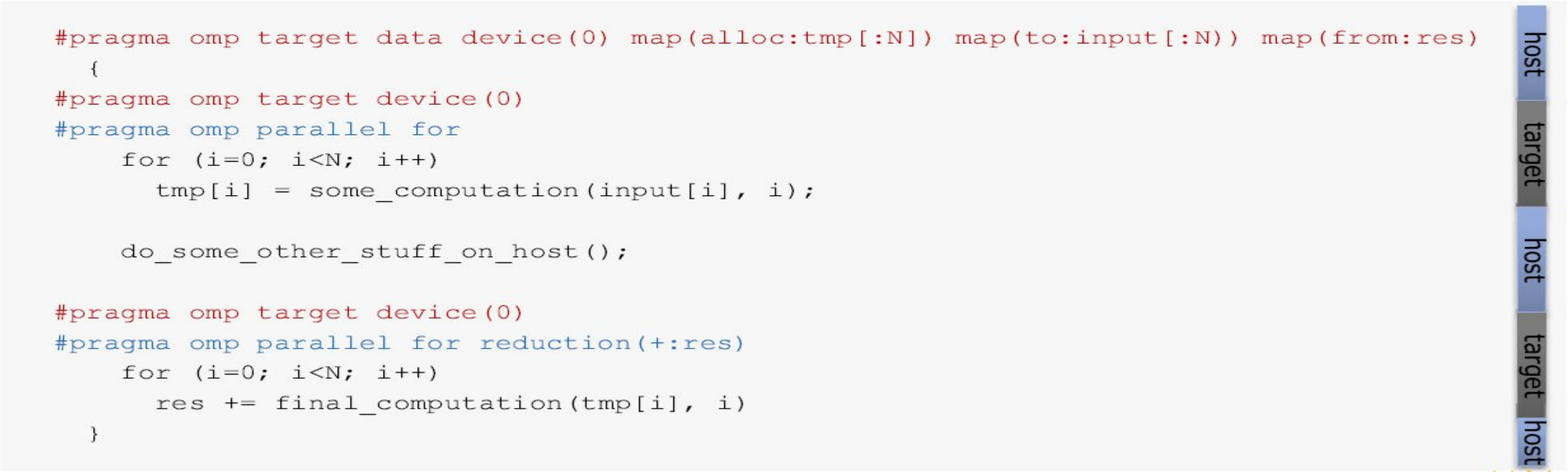
host

target

host

data environments – example

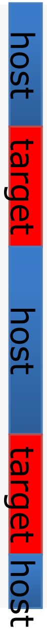
- Create a data environment to keep data on devices
  - → Avoid frequent transfers or overlap computation/comm.
  - → Pre-allocate temporary fields



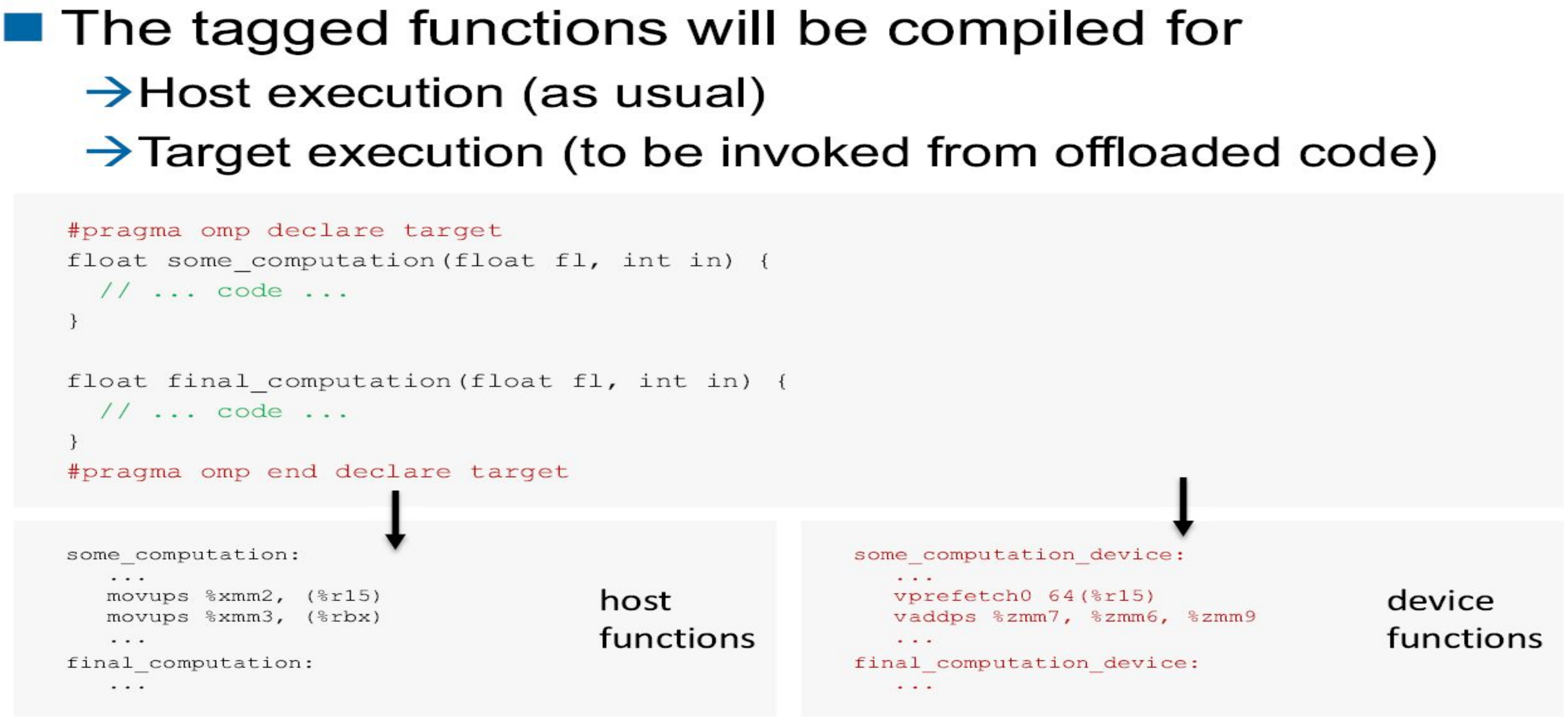
```
#pragma omp target data device(0) map(alloc:tmp[:N]) map(to:input[:N)) map(from:res)
  {
#pragma omp target device(0)
#pragma omp parallel for
    for (i=0; i<N; i++)
      tmp[i] = some_computation(input[i], i);

    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)
  }
```

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 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:
   ...
```

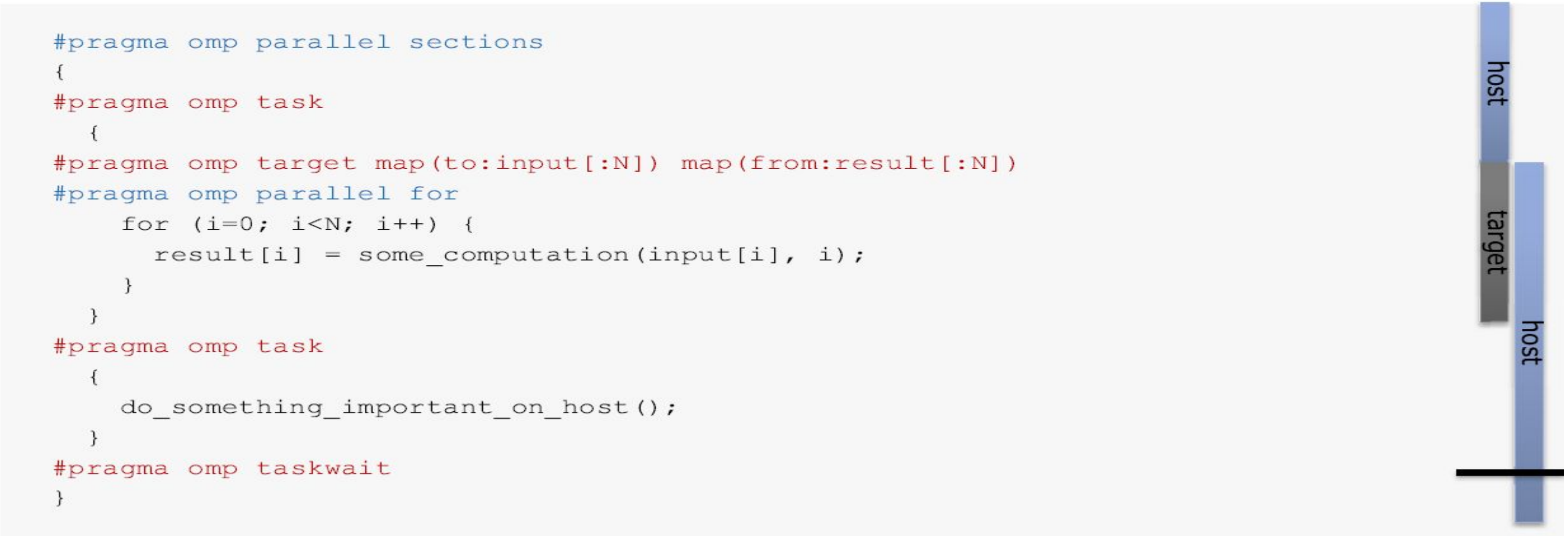
host functions

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

device functions

asynchronous offloading example

- 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
}
```

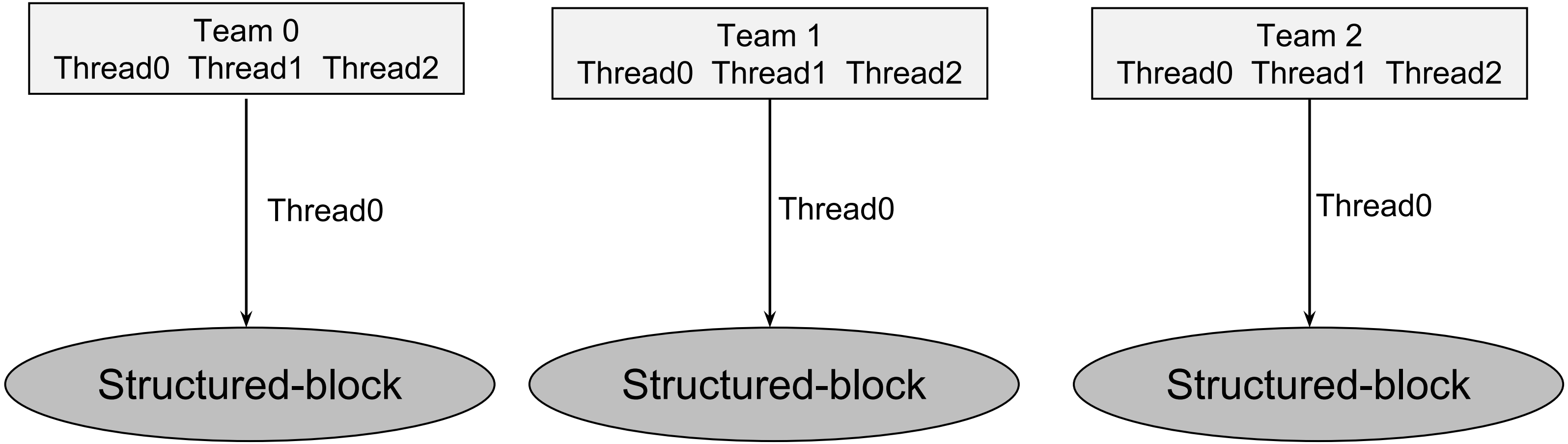
`teams` construct – restrictions

- creates a league to thread teams
  - the master thread of each team executes the teams region
  - number of teams is specified with num_teams clause
  - each team executes num_threads threads
- a teams constructs must be “perfectly” nested in a target construct
  - no statement or directives outside the teams construct
- only special OpenMP constructs can be nested inside a teams construct
  - distribute
  - parallel
  - parallel for / do
  - parallel sections

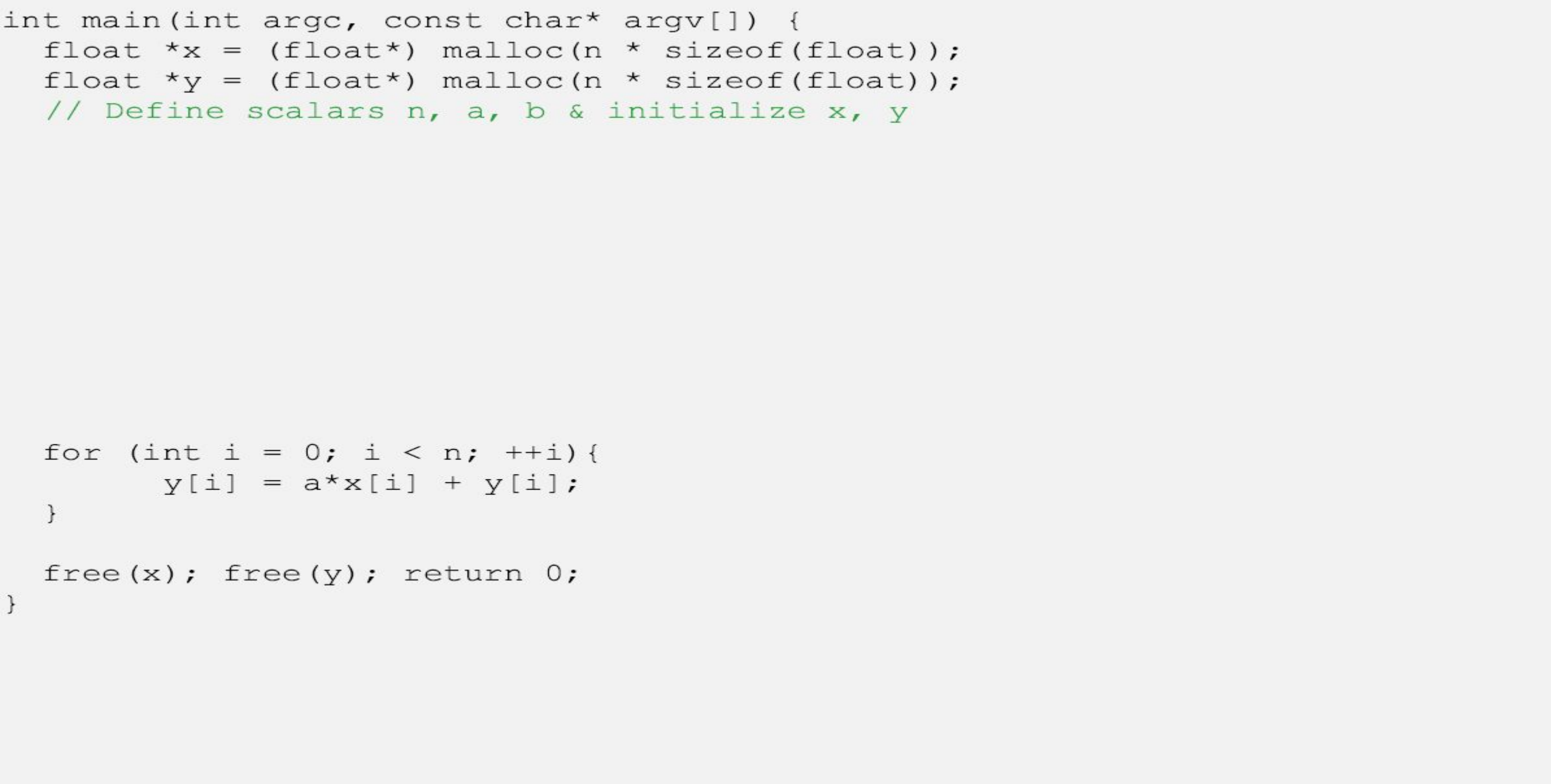
Teams Execution Model – **`teams`** constructs

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

structured-block

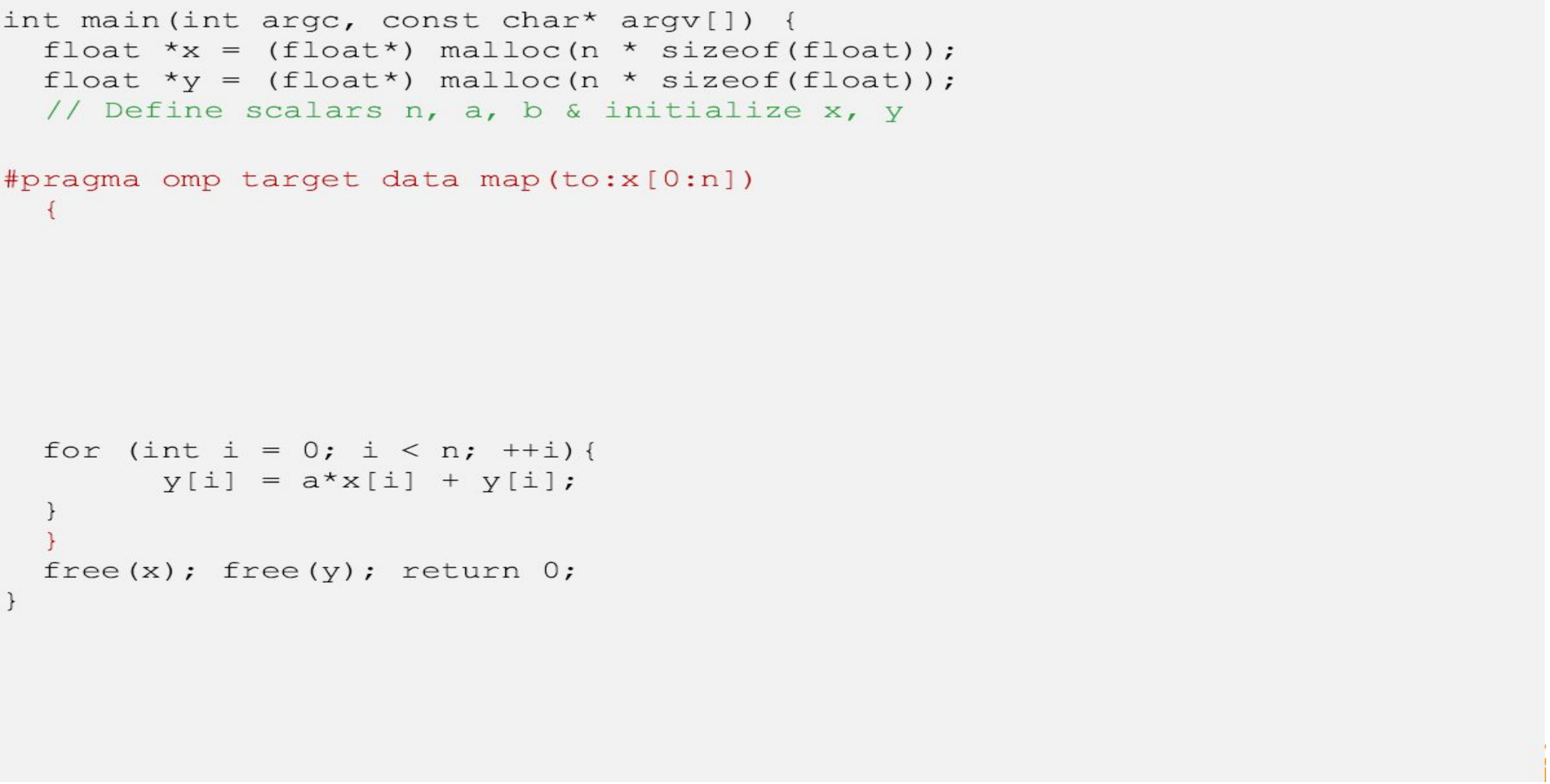


SAXPY: Serial (host)



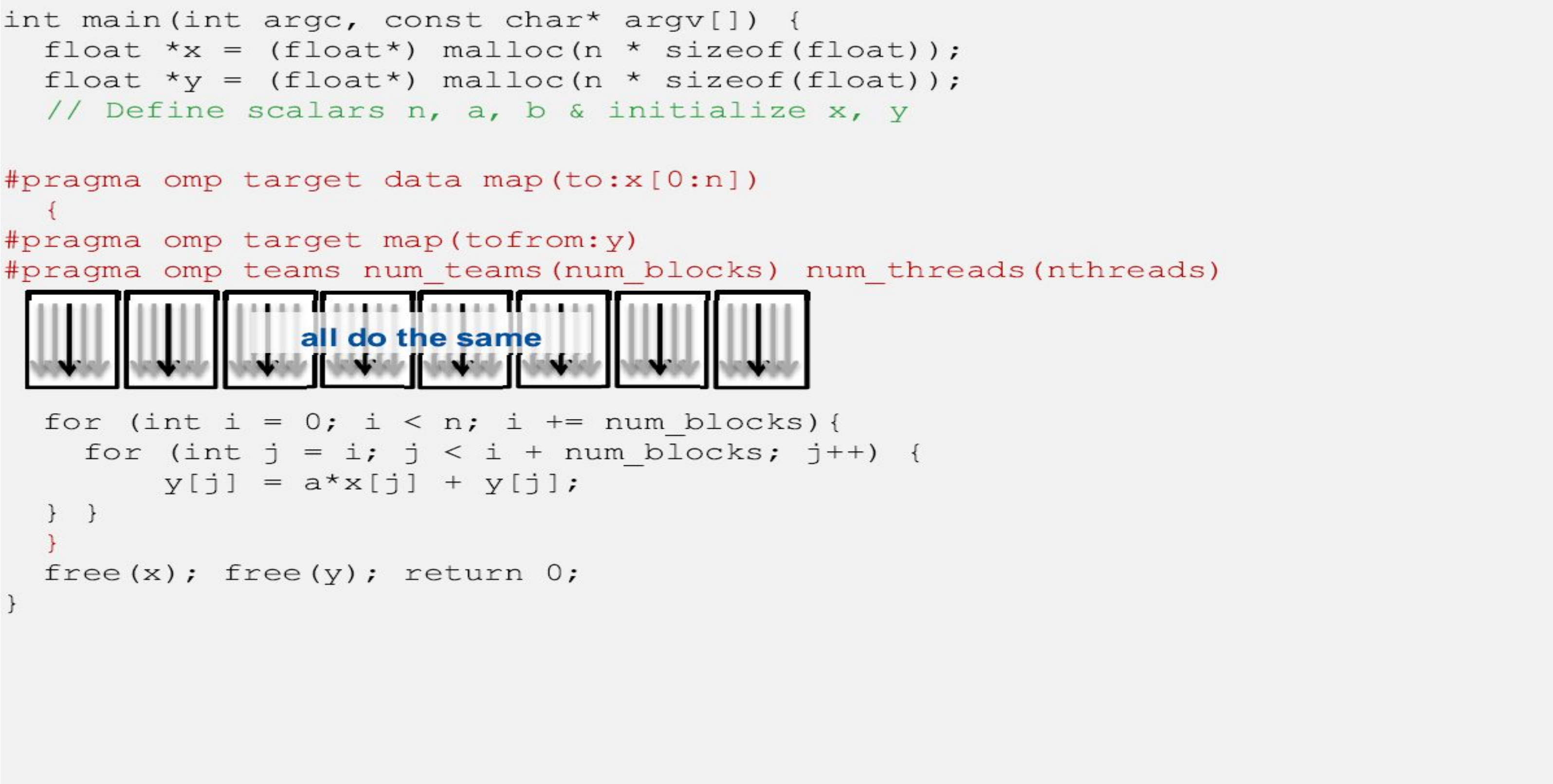
```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y










  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: 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(nthreads)
```

all do the same

```
  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` construct

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

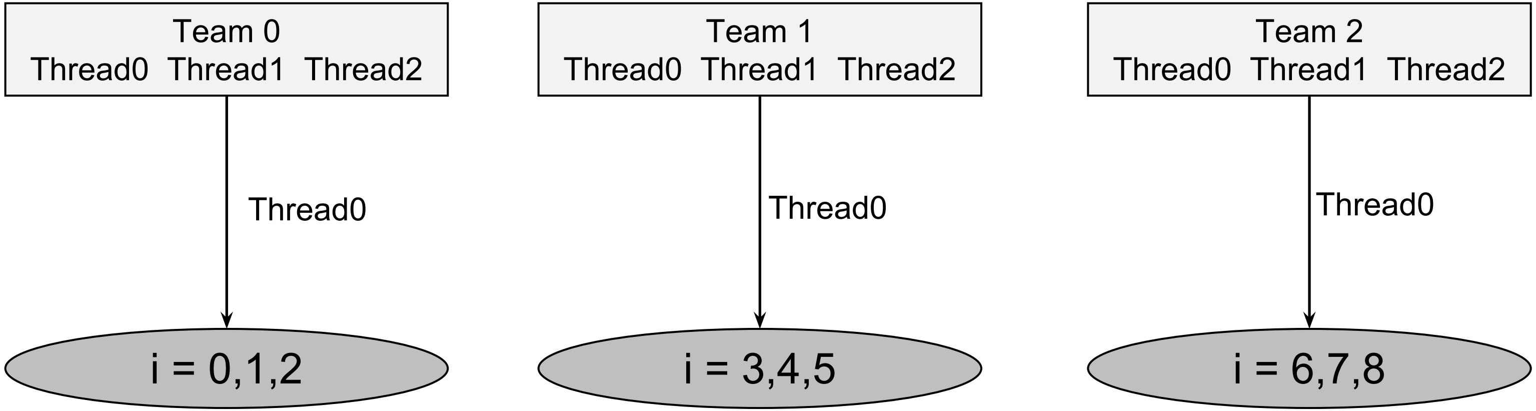
- iterations distributed among master threads of all teams
- specify to the loops only
- must be closely nested to the **`teams`** construct
- no implicit barrier at the end of the construct
- workshare among teams to exploit the parallelism on the target device

clause:

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

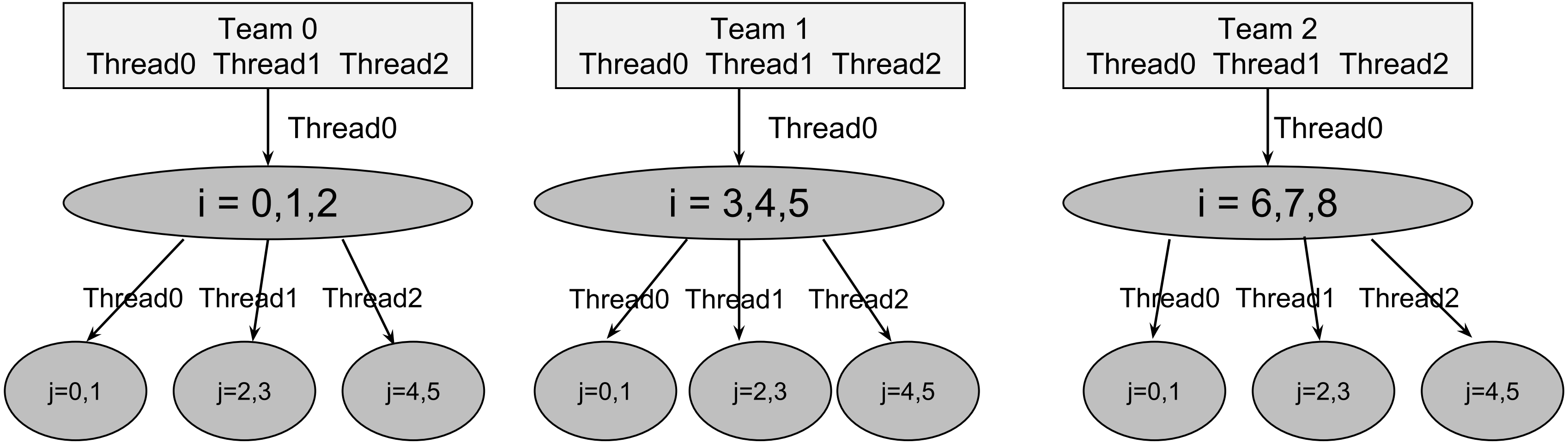
Teams + Distribute Execution Model

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

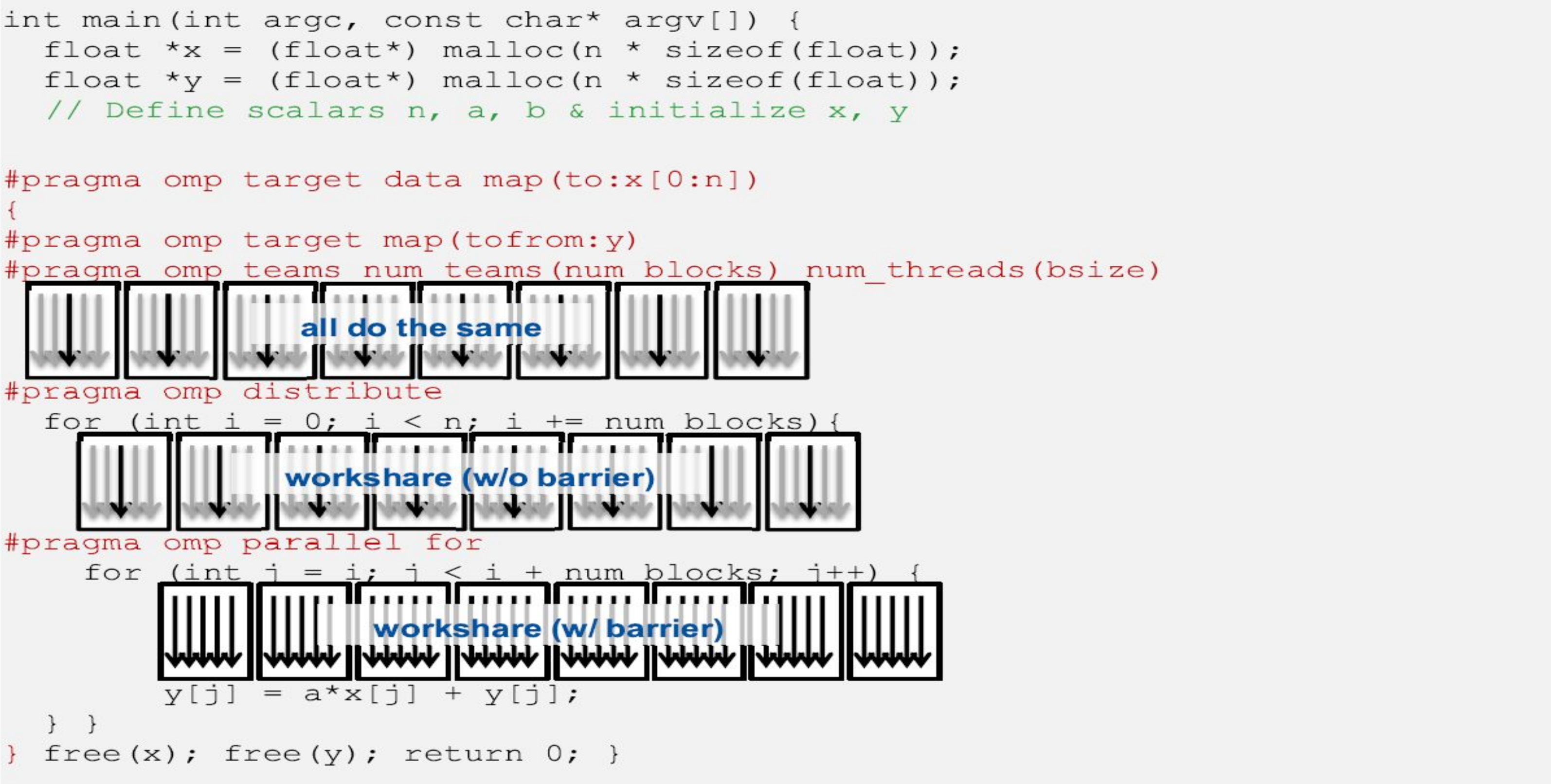


Teams + Distribute Constructs

```
#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++)  {
```

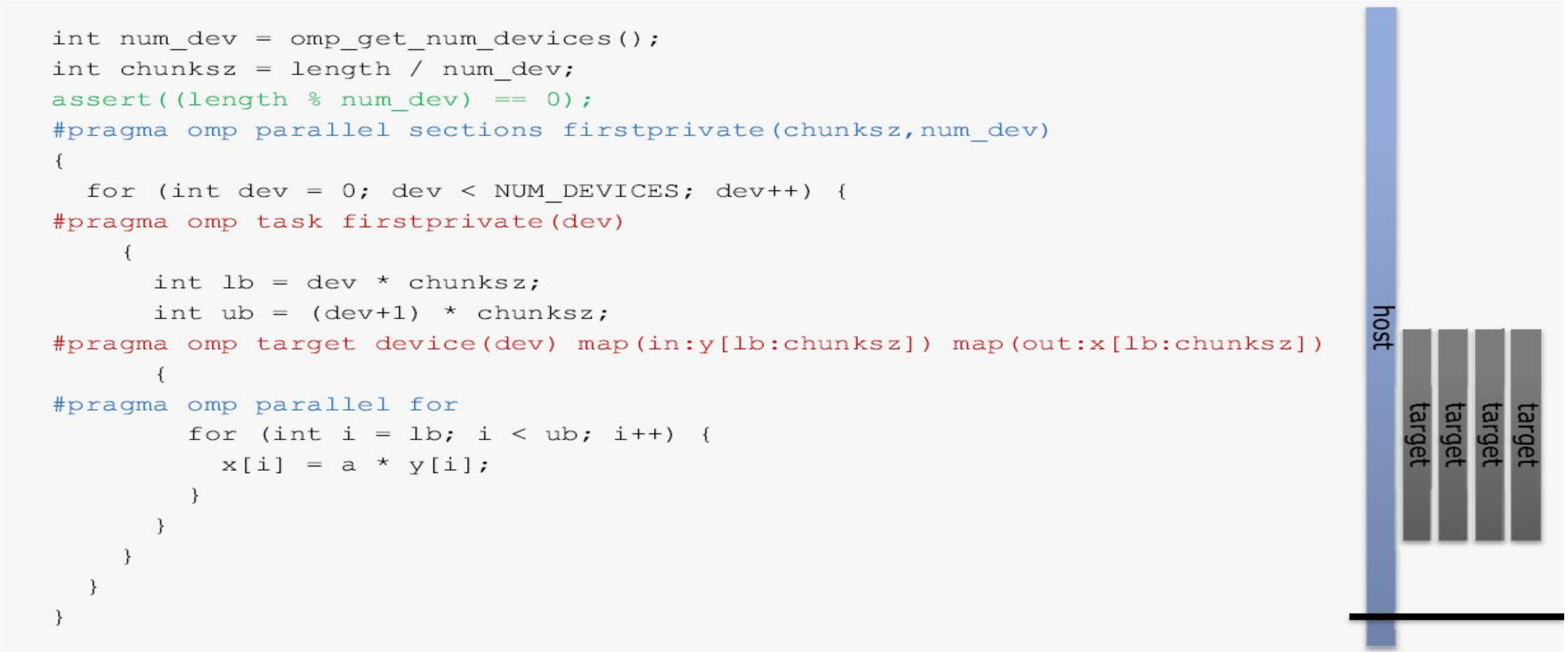


SAXPY: Coprocessor/Accelerator



```
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; }
```

using multi-device – example

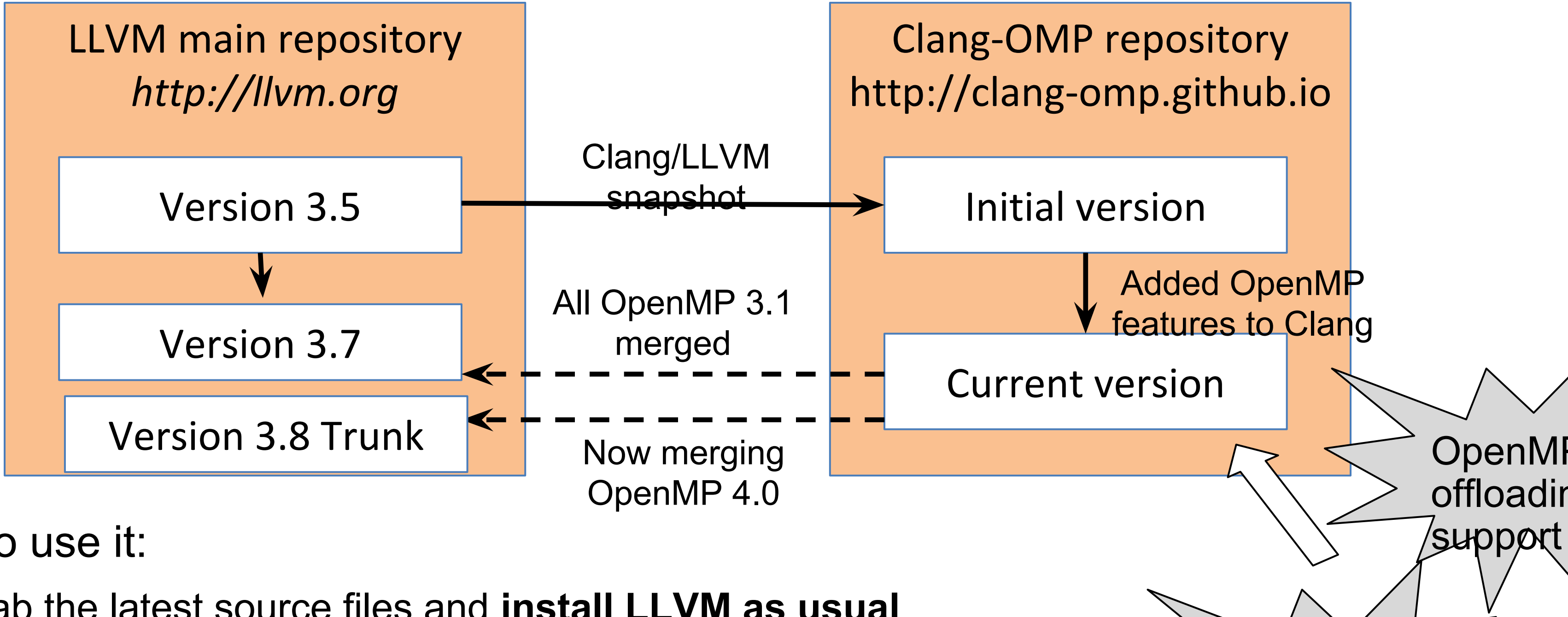


```
int num_dev = omp_get_num_devices();
int chunksz = length / num_dev;
assert((length % num_dev) == 0);
#pragma omp parallel sections firstprivate(chunksz,num_dev)
{
  for (int dev = 0; dev < NUM_DEVICES; dev++) {
#pragma omp task firstprivate(dev)
    {
      int lb = dev * chunksz;
      int ub = (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];
        }
      }
    }
  }
}
```

Agenda

- Accelerator Programming
- OpenMP 4.0 Accelerator Programming Model
- Clang/OpenMP Target-independent Offload Design
- Clang/OpenMP Offloading in Action
- Users of OpenMP-enabled clang

The codebase



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

```
$ clang -fopenmp -target powerpc64le-ibm-linux-gnu -mcpu pwr8
        -omptargets=nvptx64sm_35-nvidia-cuda <source files>
```

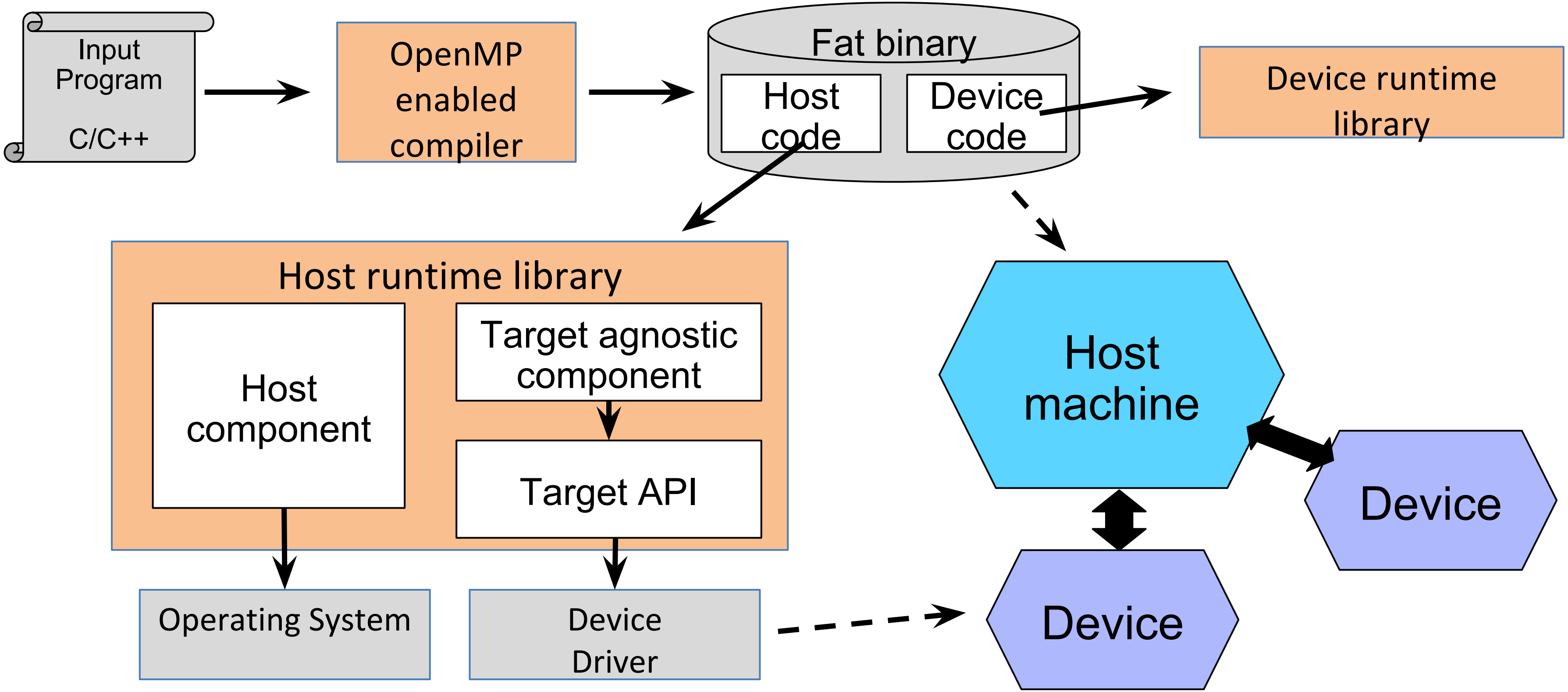
OpenMP 4.5 support

Other players

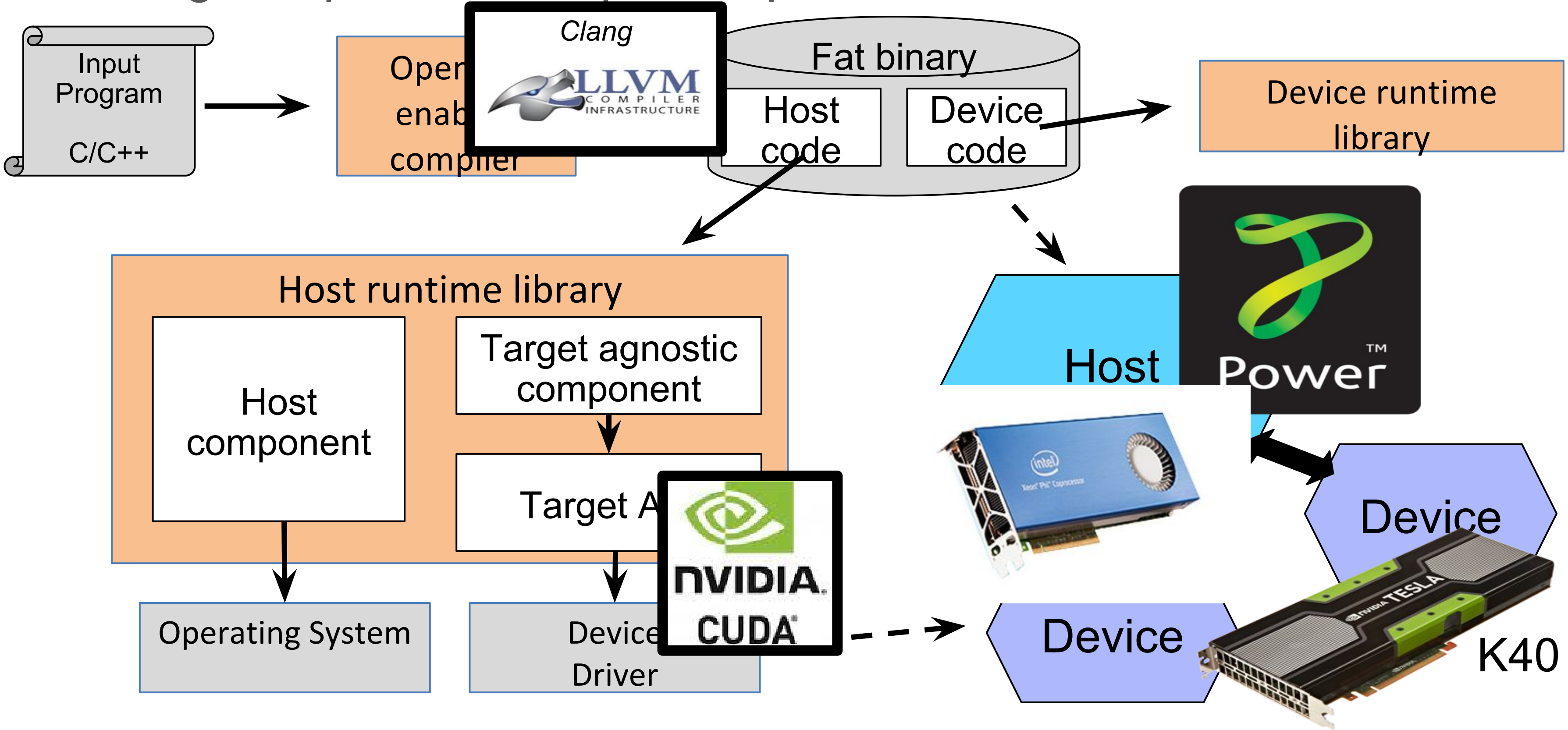
- OpenMP 4.0 support in Clang has been a joint effort
  - Offloading model specification
  - Code drops
  - Code reviews

- Project contributors include
  - IBM
  - Intel
  - Texas Instruments
  - AMD
  - DoE Laboratories
  - Other distinguished members of the Clang/LLVM community

Offloading in OpenMP – Impl. components

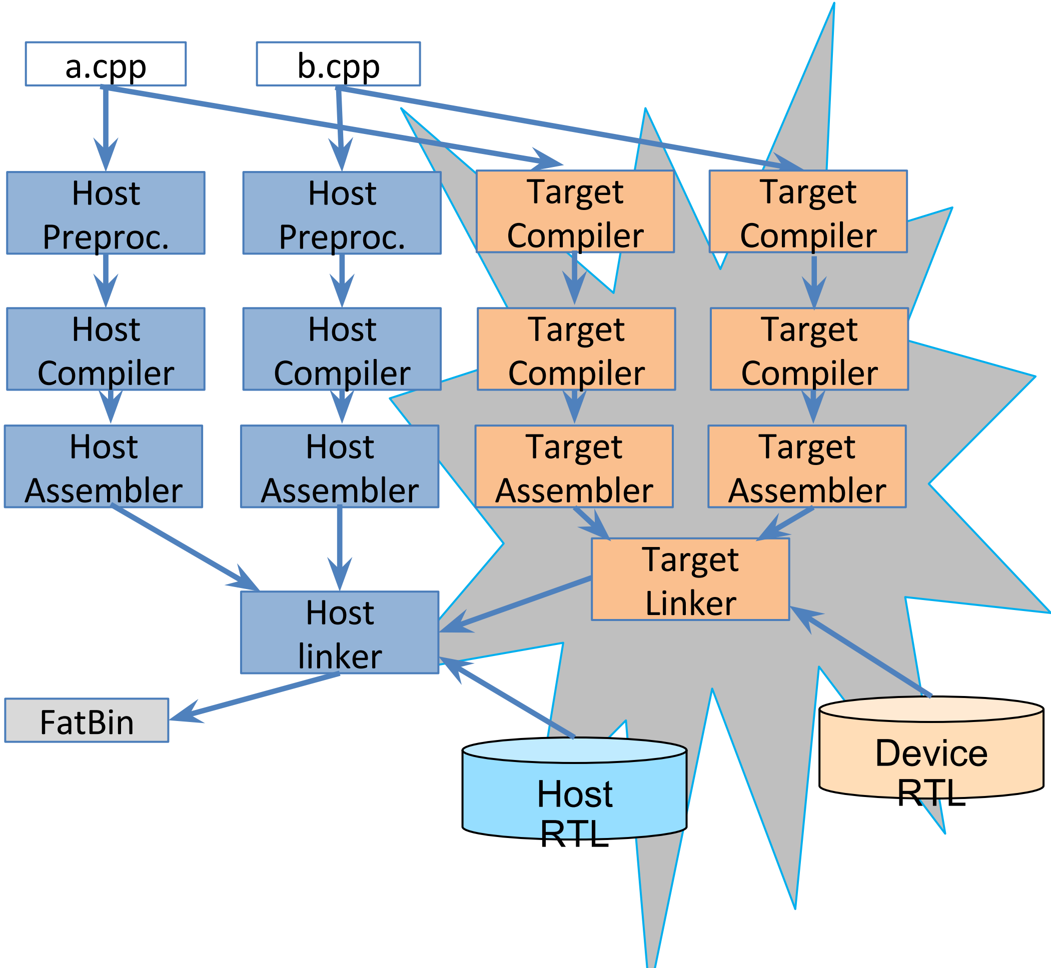


Offloading in OpenMP – Impl. components



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

Agenda

- Accelerator Programming
- OpenMP 4.0 Accelerator Programming Model
- Clang/OpenMP Target-independent Offload Design
- Clang/OpenMP Offloading in Action
- Users of OpenMP-enabled clang

OpenMP* 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

*Other names and brands may be claimed as the property of others.

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

- Initialize target device

◼ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]

⬜ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

⬜ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

⬜ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

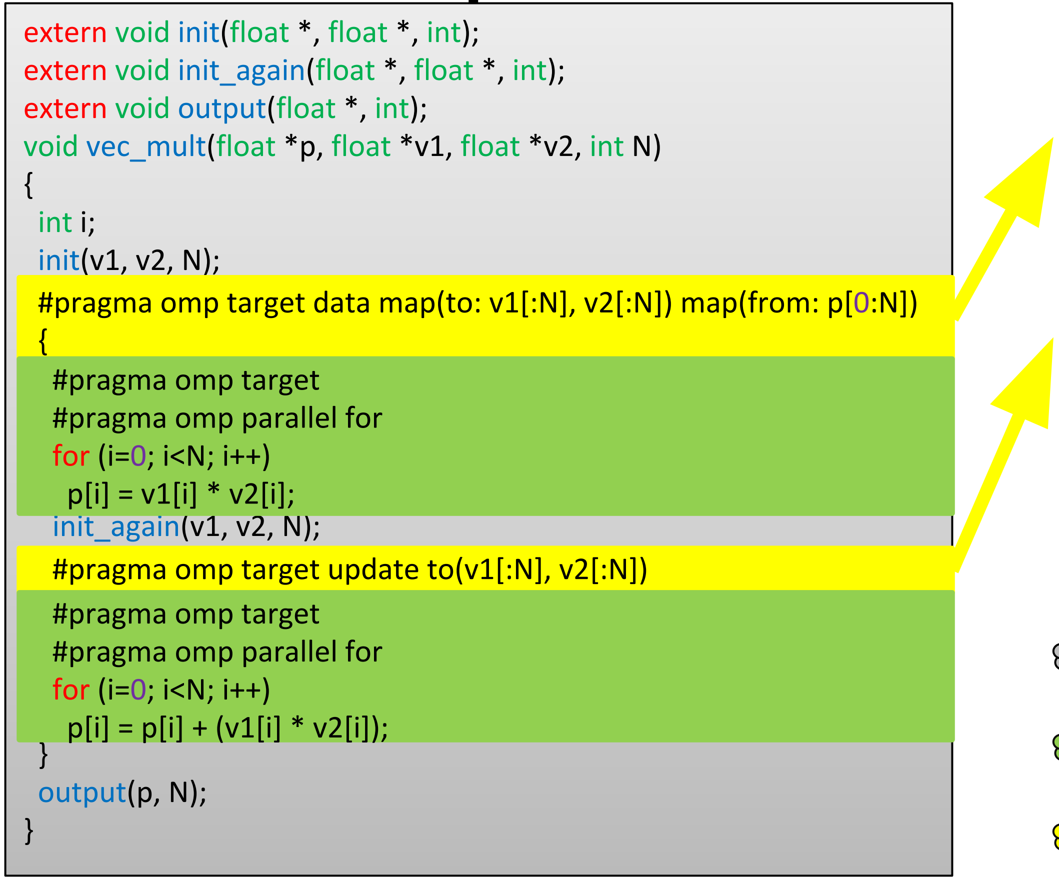
- Sync v1[N] and v2[N] between host and target

⬜ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

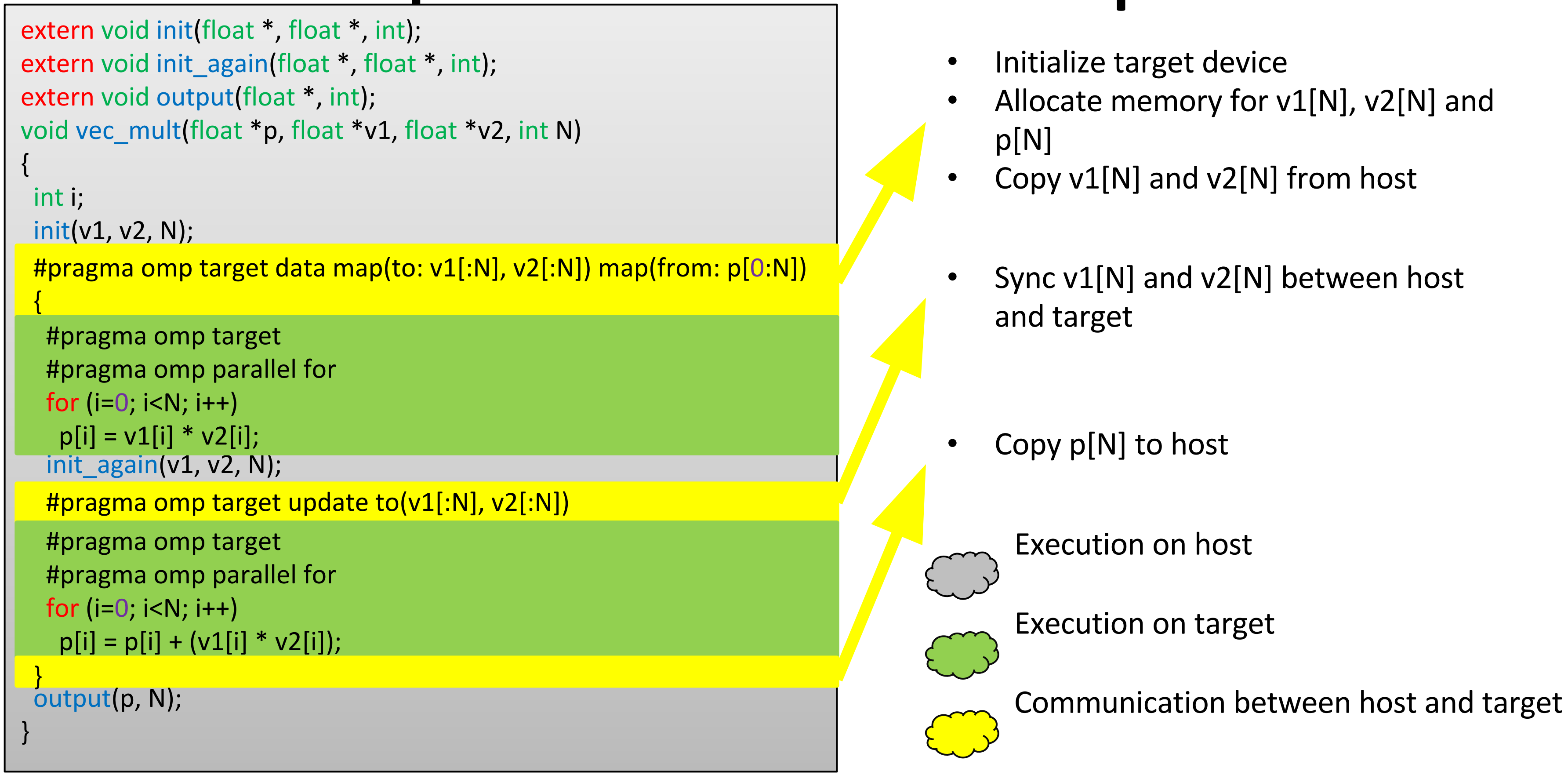
- Sync v1[N] and v2[N] between host and target

▨ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

- Sync v1[N] and v2[N] between host and target

- Copy p[N] to host

⬜ Execution on host

🟩 Execution on target

🟨 Communication between host and target

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

- Initialize target device
- Allocate memory for v1[N], v2[N] and p[N]
- Copy v1[N] and v2[N] from host

- Sync v1[N] and v2[N] between host and target

- Copy p[N] to host
- Free memory on target

⬜ Execution on host

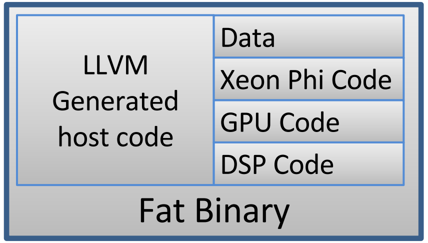
🟩 Execution on target

🟨 Communication between host and target

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
```



LLVM Generated host code

Data

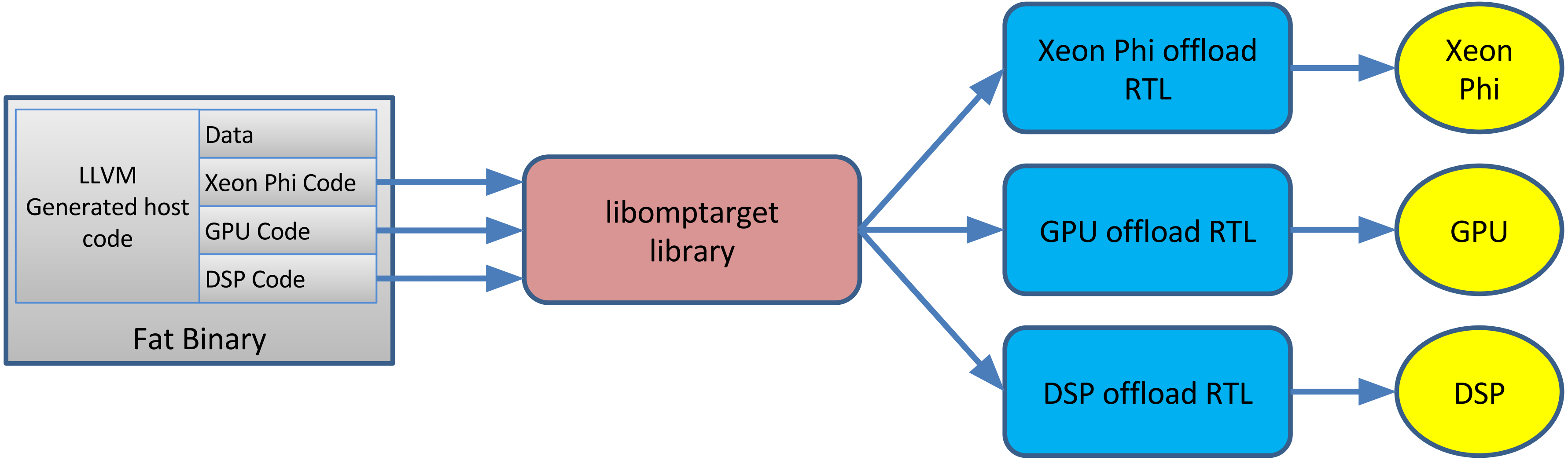
Xeon Phi Code

GPU Code

DSP Code

Fat Binary

Heterogeneous Execution of Fat Binary



libomptarget library

Used for communication between host and target offload RTL

- void __tgt_register_lib() – register library and initialize target
- void __tgt_target_data_begin() – initiate a device data environment + data upload
- void __tgt_target_data_end() – close a device data environment + data download
- void __tgt_target_data_update() – sync data between host and target
- int32_t __tgt_target() – data upload-run code on target-data download
- int32_t __tgt_target_teams() – data upload-run code on target-data download for target teams.

libomptarget library

Target offload RTL

Used for communication between host (libomptarget) and target devices

- int32_t __tgt_rtl_device_type() – device type
- int32_t __tgt_rtl_number_of_devices() – number of devices
- int32_t __tgt_init_device() – initialize device
- tgt_target_table* __tgt_rtl_load_binary() – send executable section to device
- void* __tgt_rtl_data_alloc() – allocate memory on device
- int32_t __tgt_rtl_data_delete() – delete memory on device
- int32_t __tgt_rtl_data_submit() – send data to device
- int32_t __tgt_rtl_data_retrieve() – get data from device
- int32_t __tgt_rtl_run_target_region() – run code on device
- int32_t __tgt_rtl_run_target_team_region() – run code on device

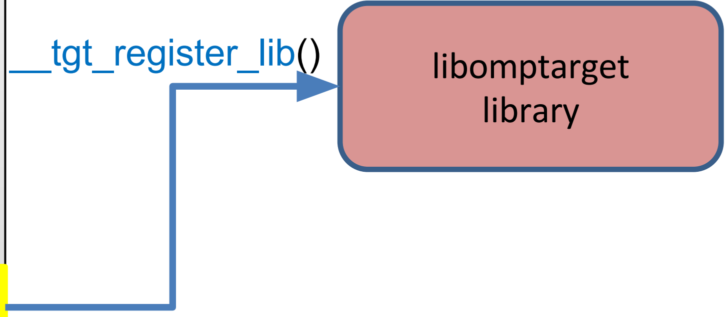
Target offload RTL

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```

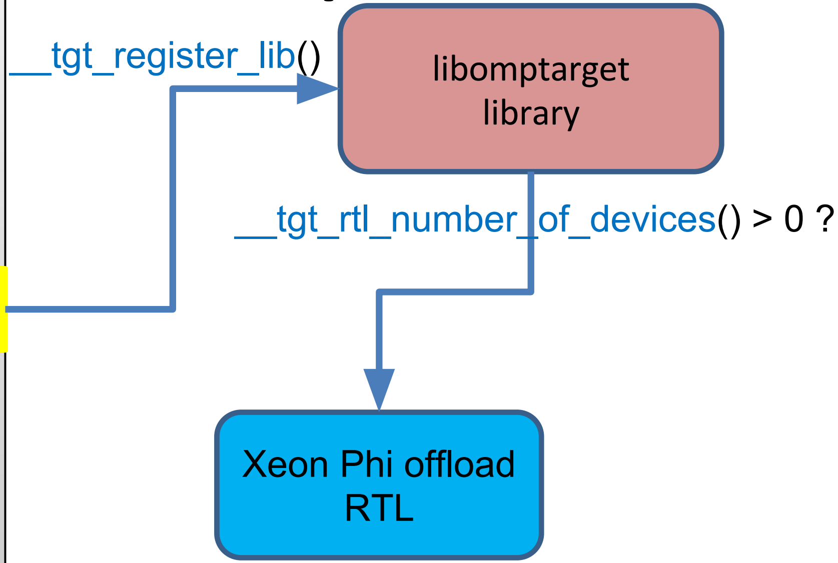
OpenMP 4.0 Example

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



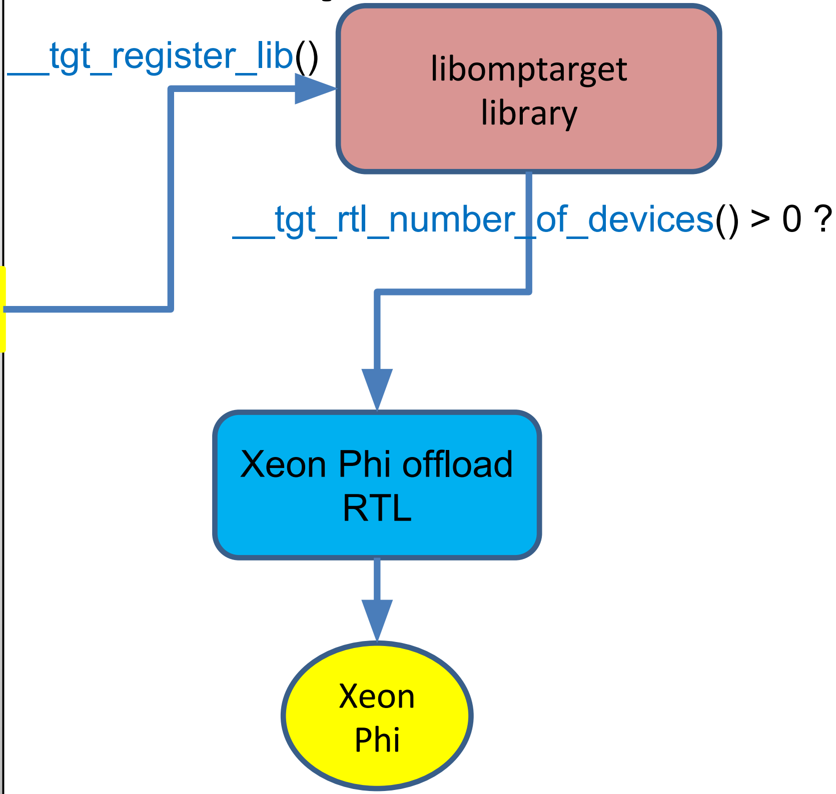
OpenMP 4.0 Example

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



OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```



__tgt_register_lib()

libomptarget library

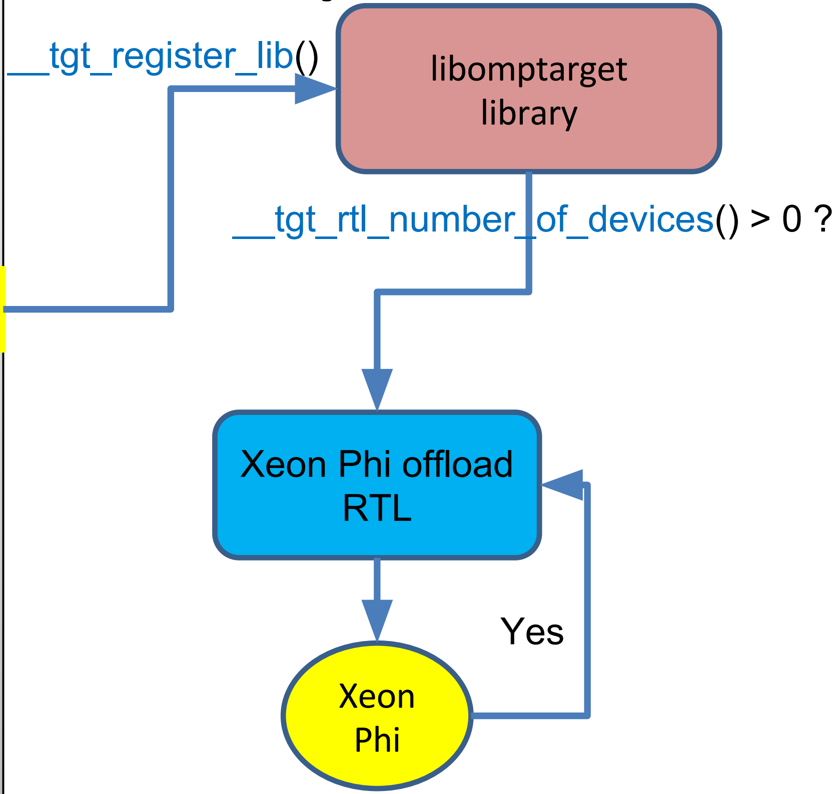
__tgt_rtl_number_of_devices() > 0 ?

Xeon Phi offload RTL

Xeon Phi

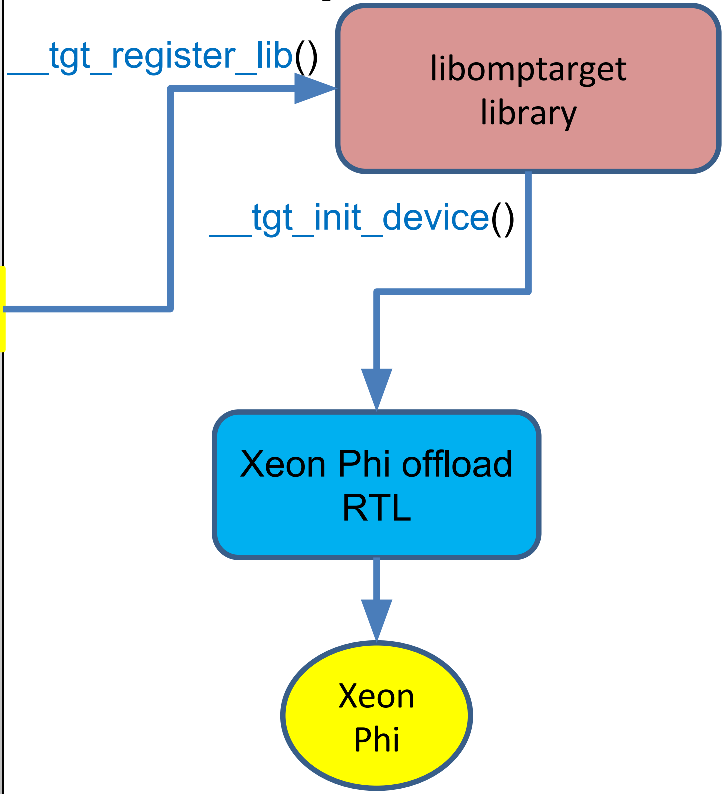
OpenMP 4.0 Example

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



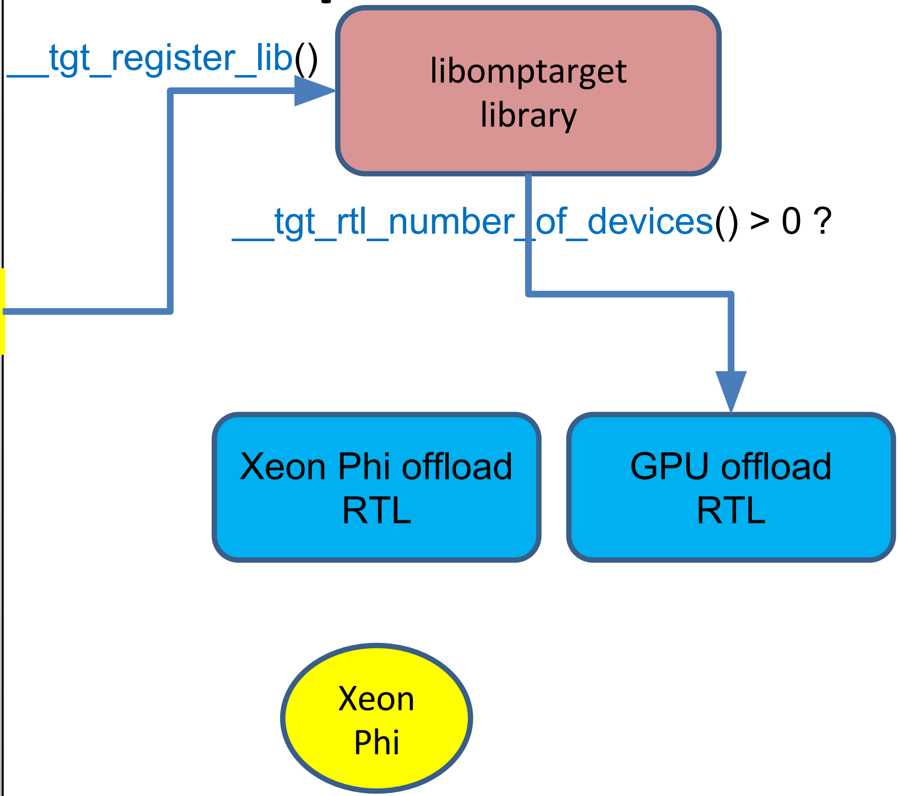
OpenMP 4.0 Example

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



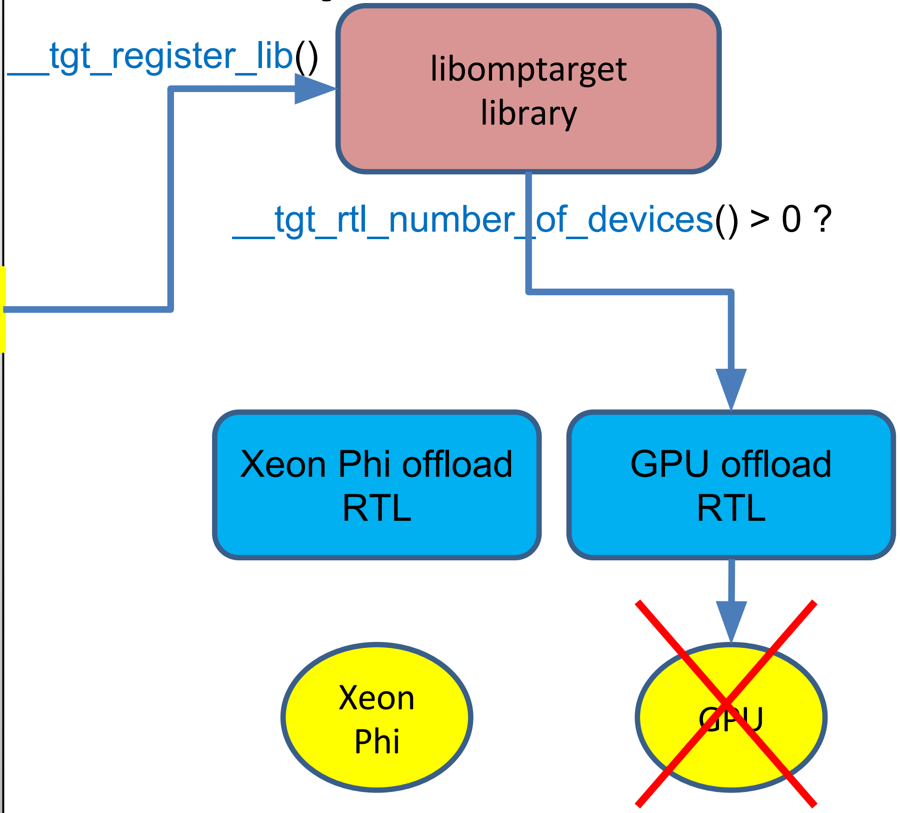
OpenMP 4.0 Example

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



OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```



__tgt_register_lib()

libomptarget library

__tgt_rtl_number_of_devices() > 0 ?

Xeon Phi offload RTL

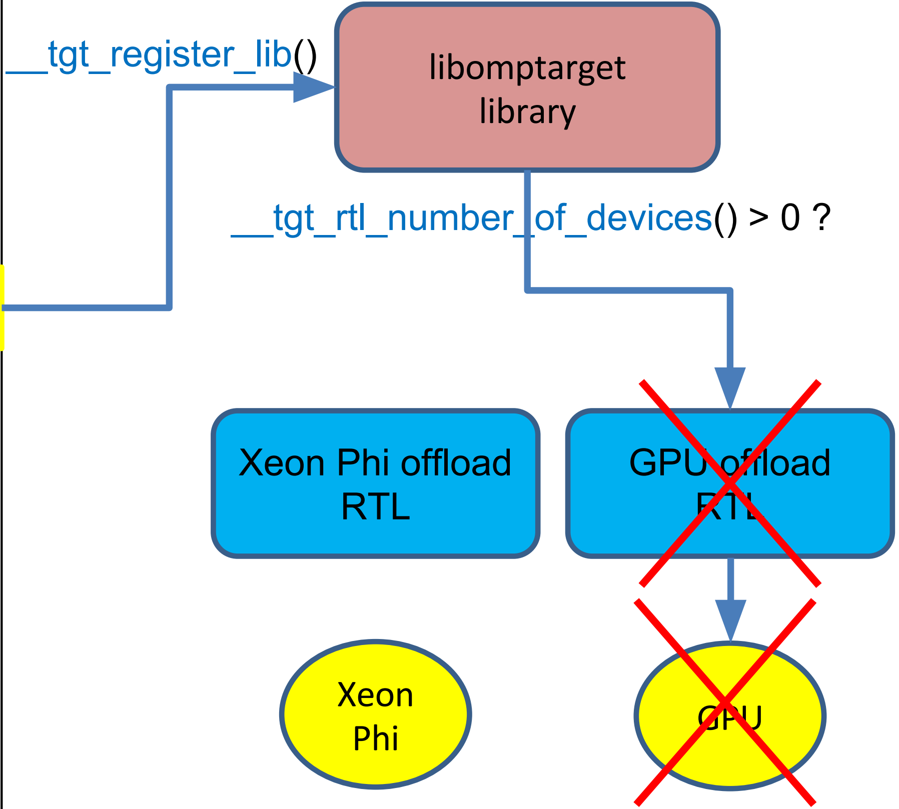
GPU offload RTL

Xeon Phi

GPU

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```

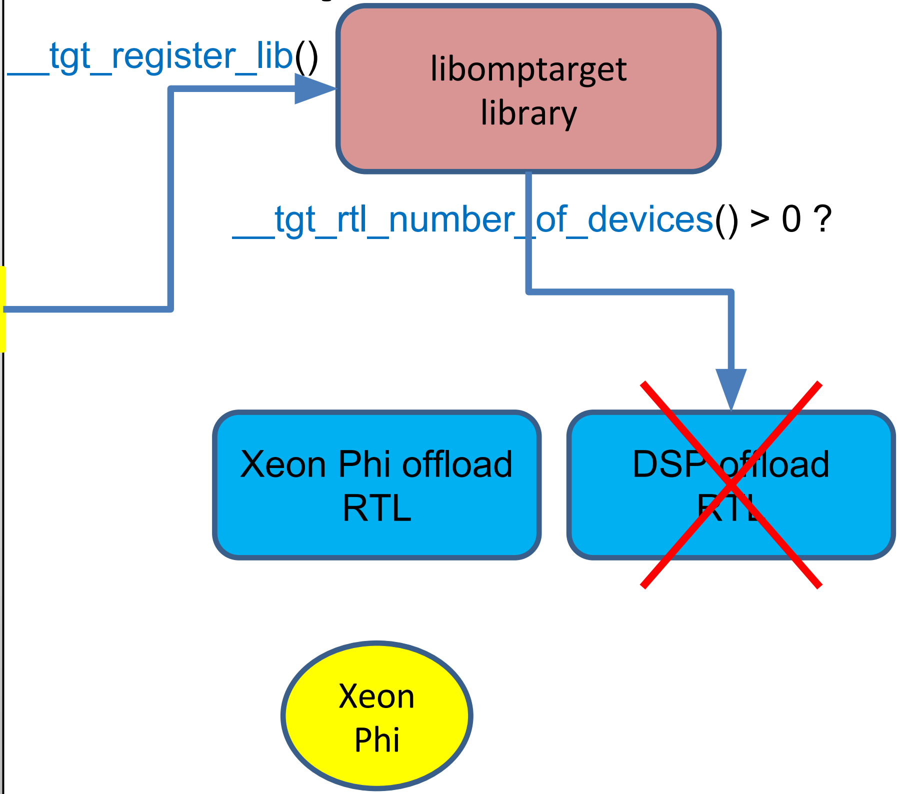


__tgt_register_lib()

__tgt_rtl_number_of_devices() > 0 ?

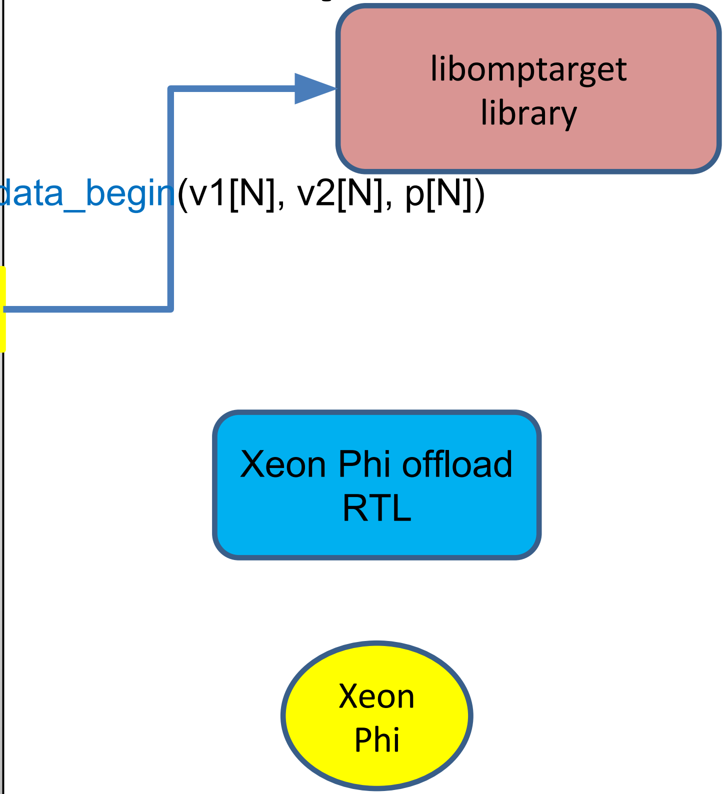
OpenMP 4.0 Example

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



OpenMP 4.0 Example

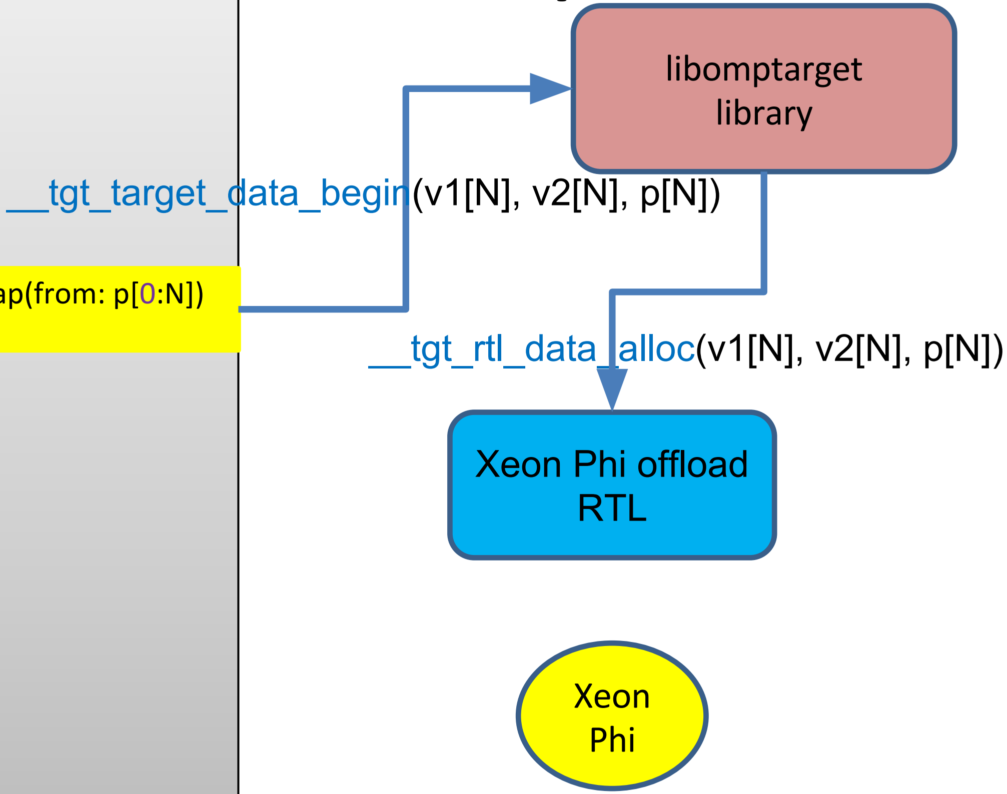
```
extern void init(float *, float *, int);
extern void init_again(float *, float *, int);
extern void output(float *, int);
void vec_mult(float *p, float *v1, float *v2, int N)
{
 int i;
 init(v1, v2, N);
 #pragma omp target data map(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



__tgt_target_data_begin(v1[N], v2[N], p[N])

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



libomptarget library

__tgt_target_data_begin(v1[N], v2[N], p[N])

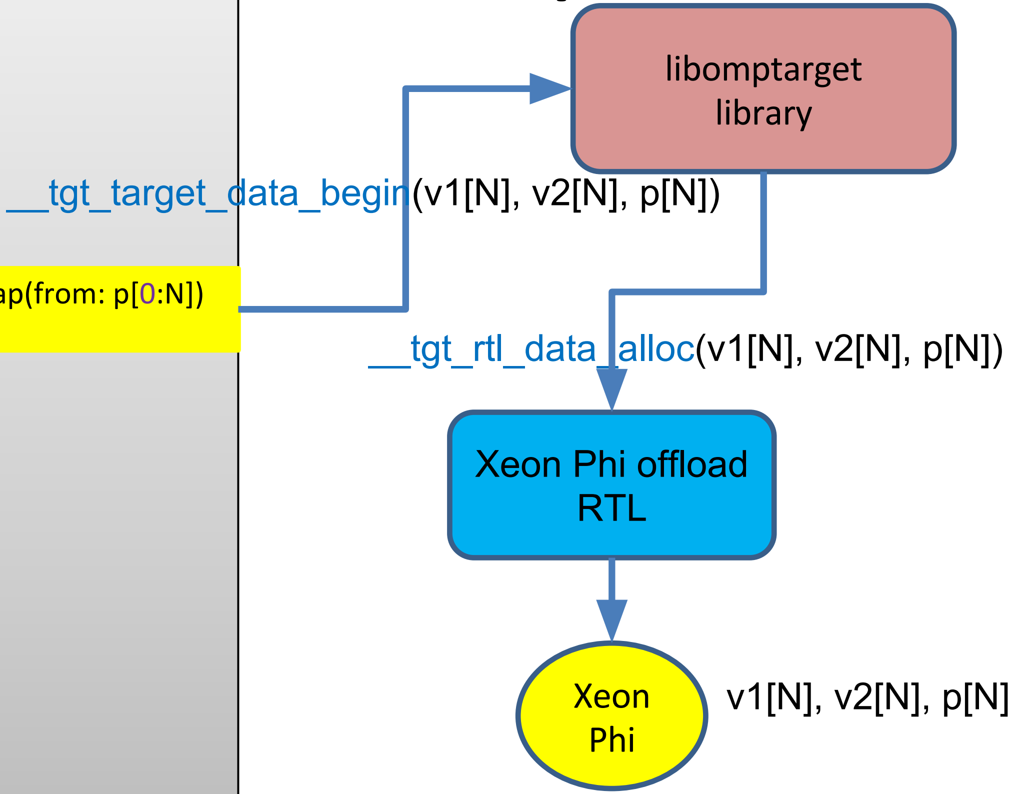
__tgt_rtl_data_alloc(v1[N], v2[N], p[N])

Xeon Phi offload RTL

Xeon Phi

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



libomptarget library

__tgt_target_data_begin(v1[N], v2[N], p[N])

__tgt_rtl_data_alloc(v1[N], v2[N], p[N])

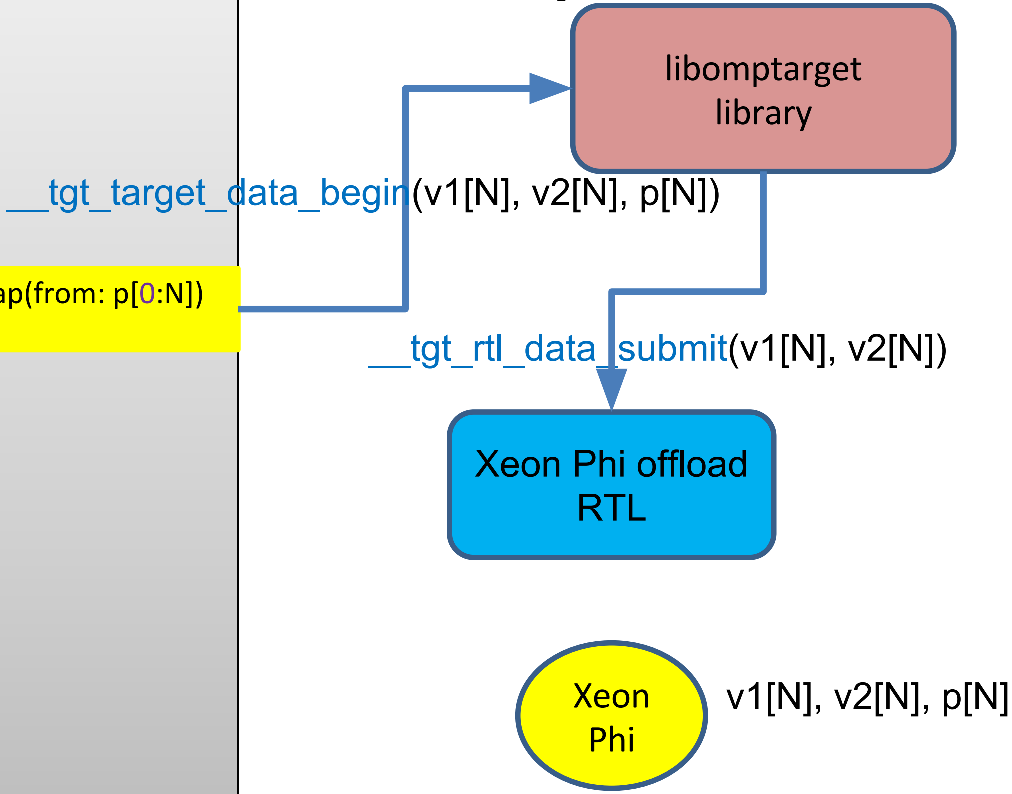
Xeon Phi offload RTL

Xeon Phi

v1[N], v2[N], p[N]

OpenMP 4.0 Example

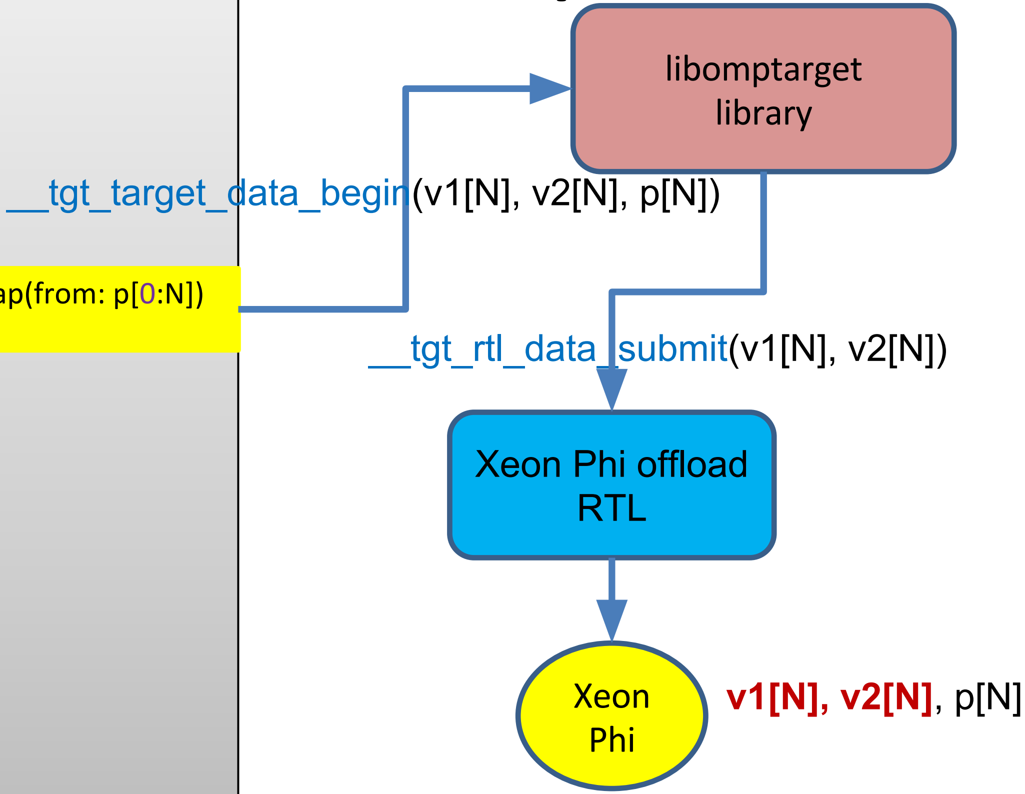
```
extern void init(float *, float *, int);
extern void init_again(float *, float *, int);
extern void output(float *, int);
void vec_mult(float *p, float *v1, float *v2, int N)
{
 int i;
 init(v1, v2, N);
 #pragma omp target data map(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



libomptarget library
__tgt_target_data_begin(v1[N], v2[N], p[N])
__tgt_rtl_data_submit(v1[N], v2[N])
Xeon Phi offload RTL
Xeon Phi
v1[N], v2[N], p[N]

OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



libomptarget library

__tgt_target_data_begin(v1[N], v2[N], p[N])

__tgt_rtl_data_submit(v1[N], v2[N])

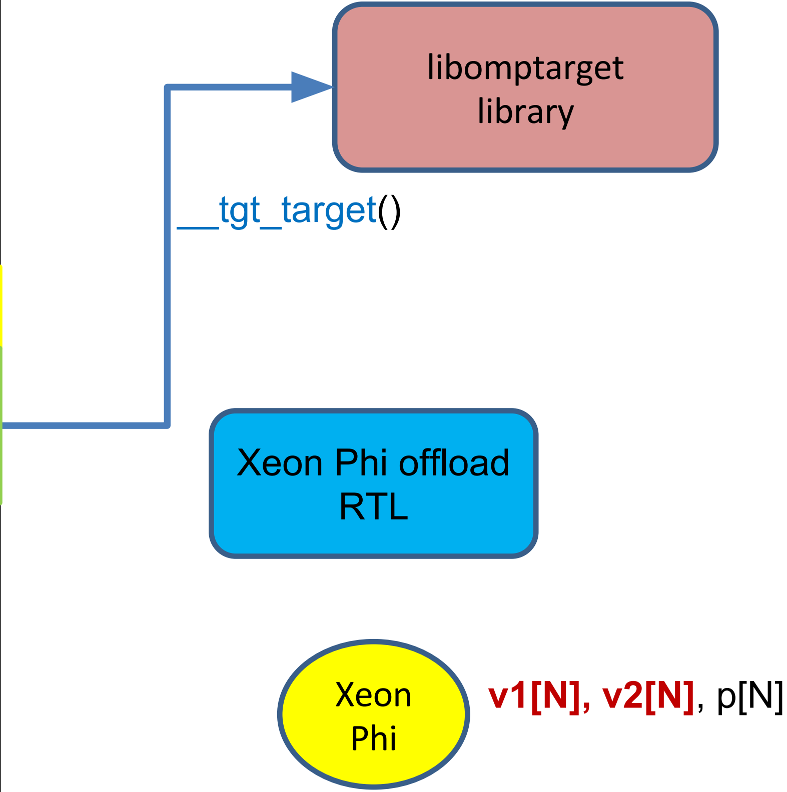
Xeon Phi offload RTL

Xeon Phi

v1[N], v2[N], p[N]

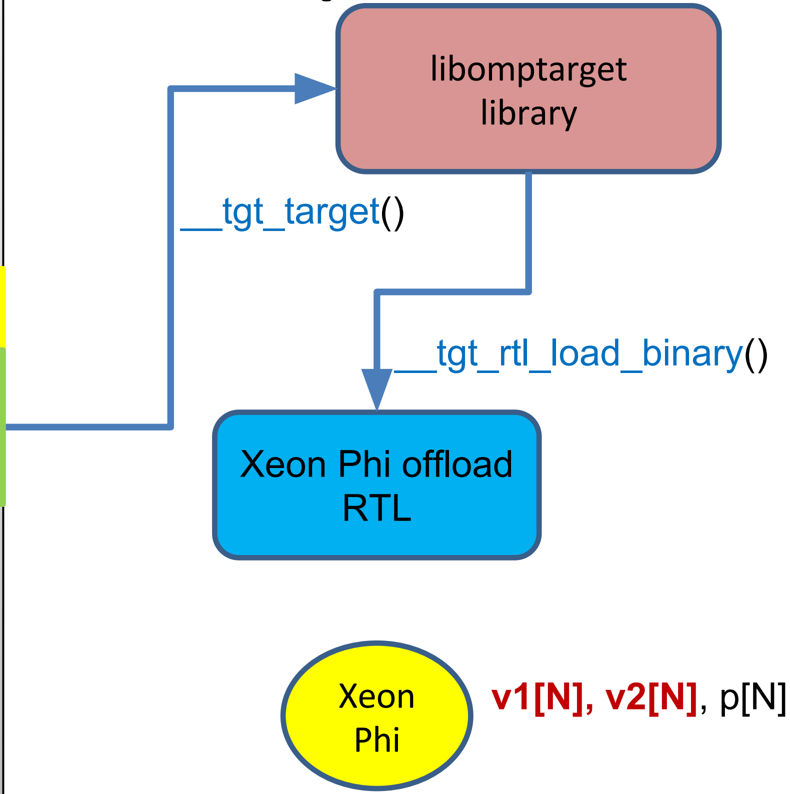
OpenMP 4.0 Example

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



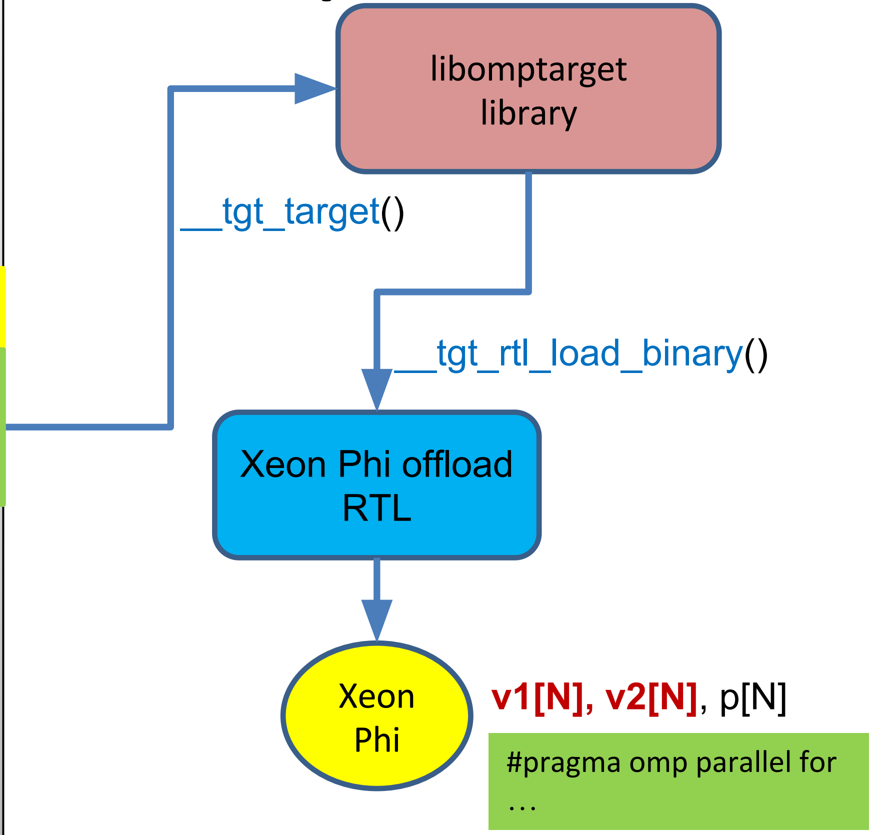
OpenMP 4.0 Example

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



OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



libomptarget library

__tgt_target()

__tgt_rtl_load_binary()

Xeon Phi offload RTL

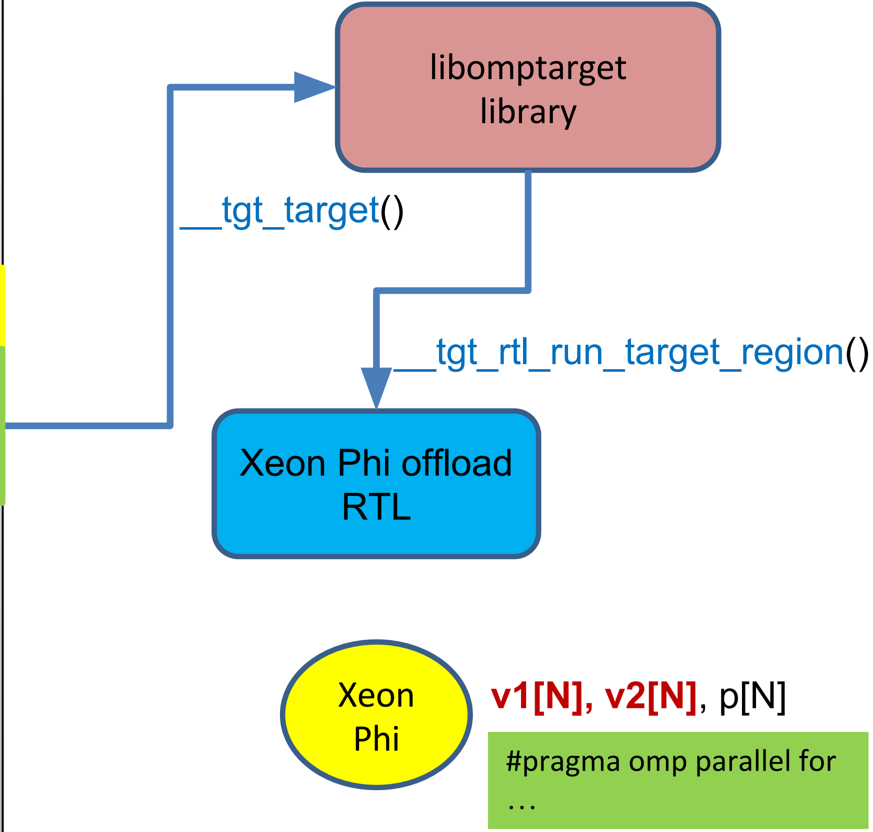
Xeon Phi

v1[N], v2[N], p[N]

```
#pragma omp parallel for
…
```

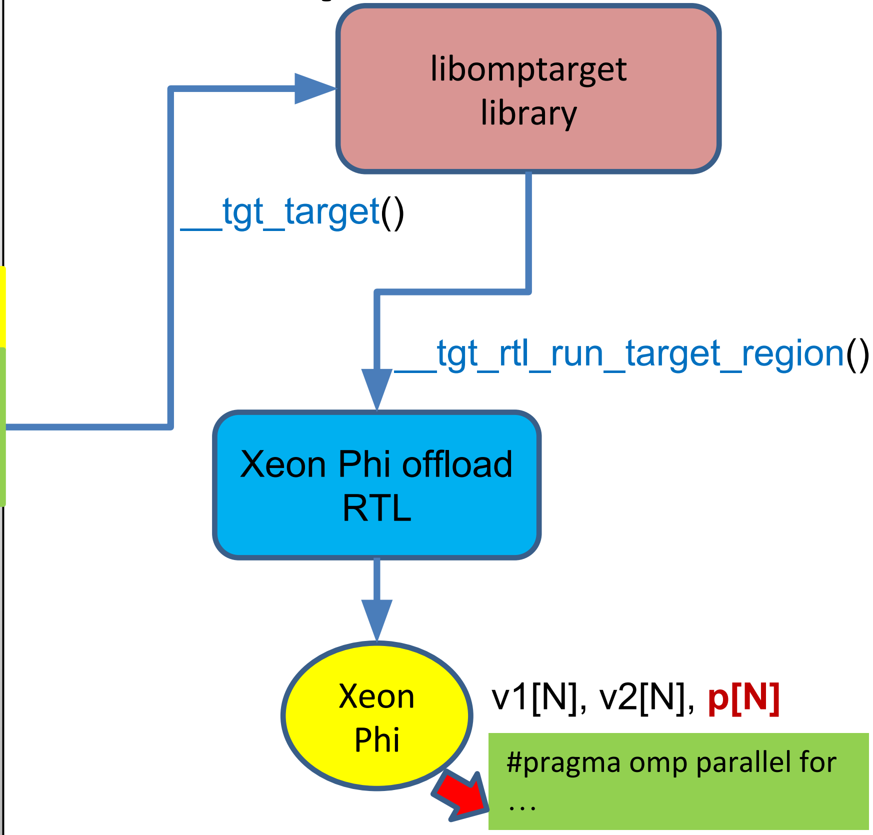
OpenMP 4.0 Example

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



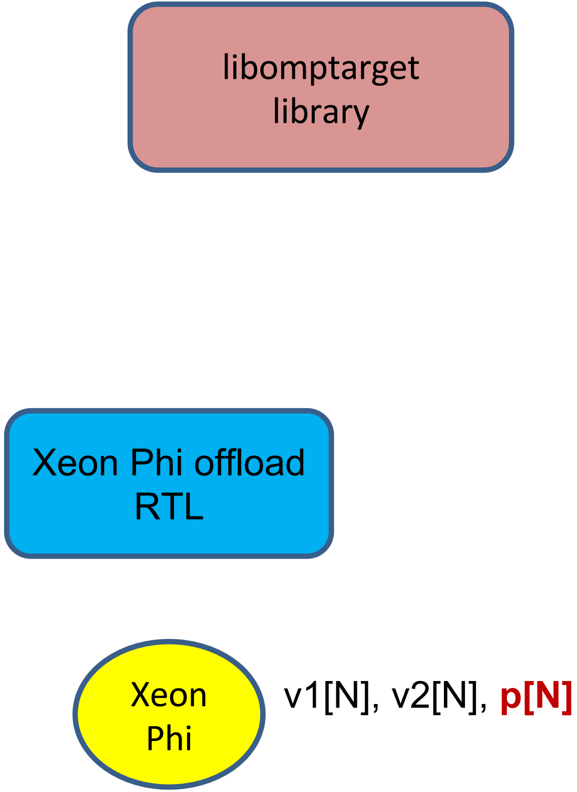
OpenMP 4.0 Example

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



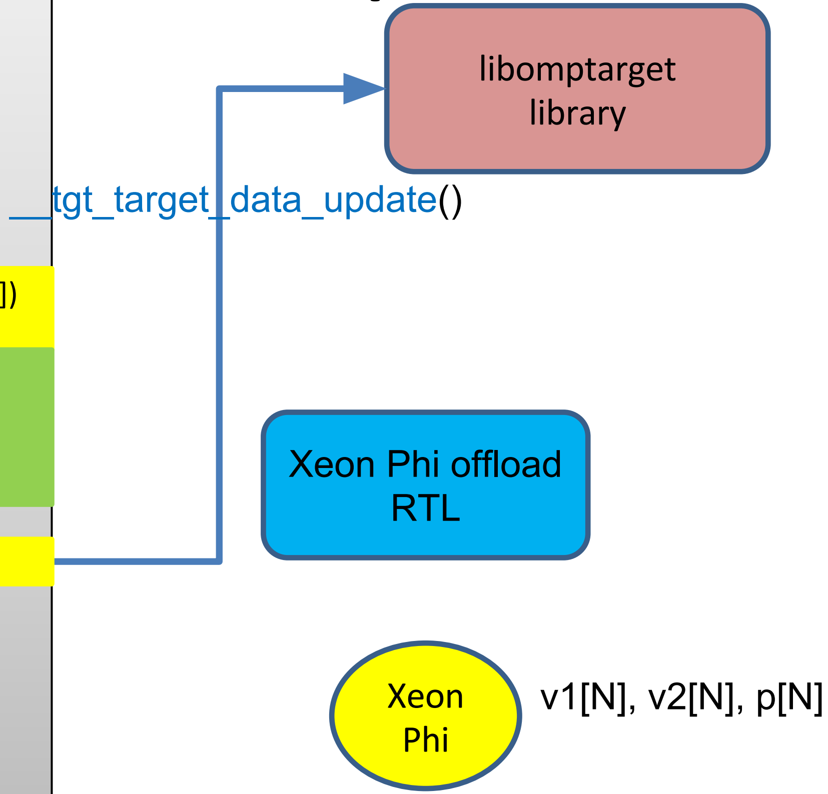
OpenMP 4.0 Example

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



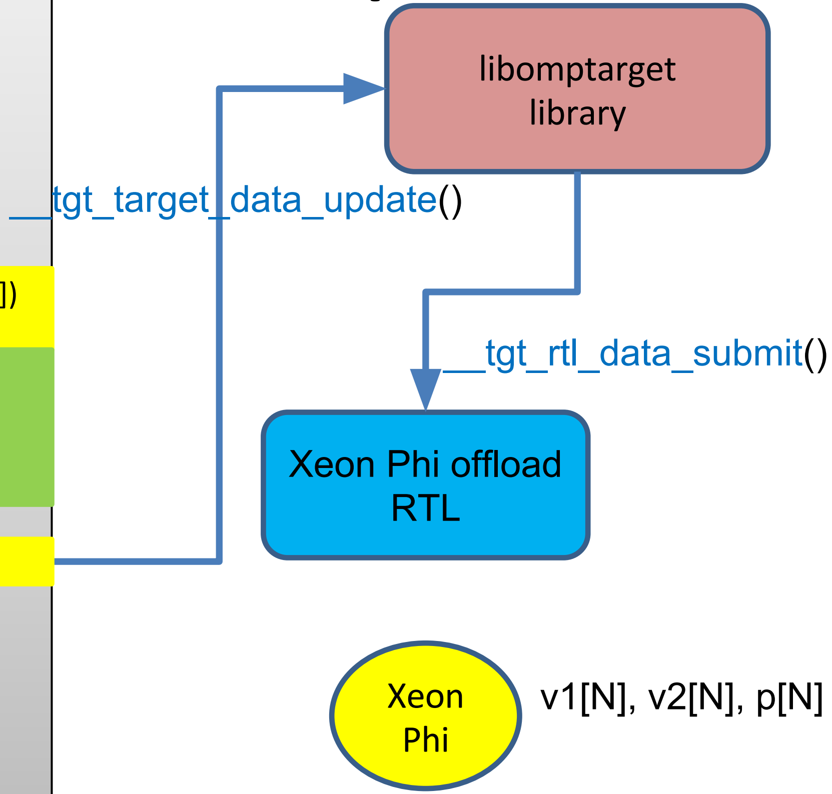
OpenMP 4.0 Example

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



OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
  {
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
    init_again(v1, v2, N);
    #pragma omp target update to(v1[:N], v2[:N])
    #pragma omp target
    #pragma omp parallel for
    for (i=0; i<N; i++)
      p[i] = p[i] + (v1[i] * v2[i]);
  }
  output(p, N);
}
```



libomptarget library

__tgt_target_data_update()

__tgt_rtl_data_submit()

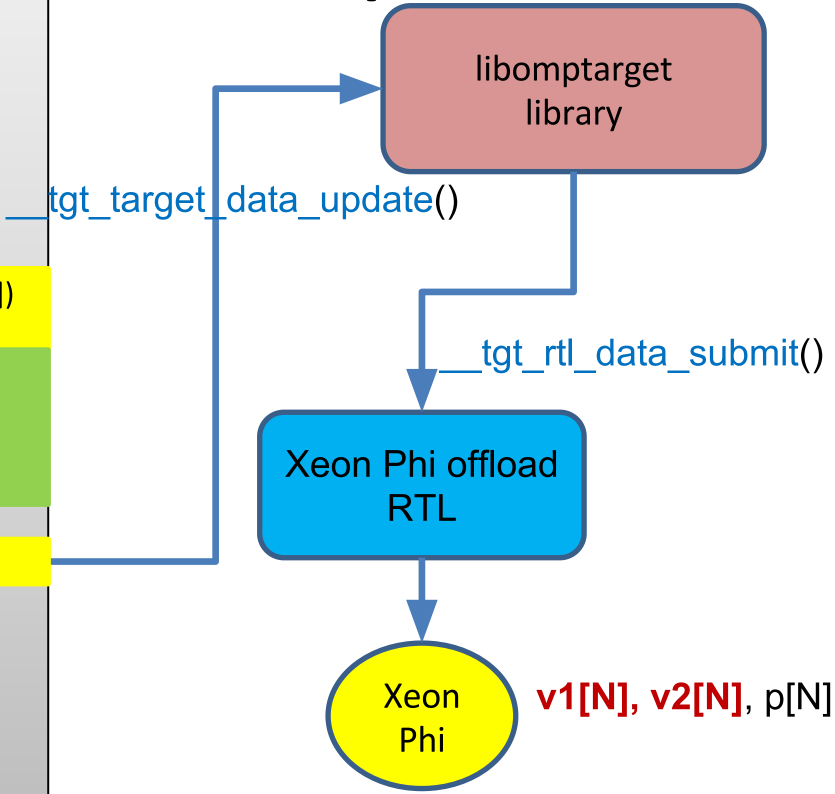
Xeon Phi offload RTL

Xeon Phi

v1[N], v2[N], p[N]

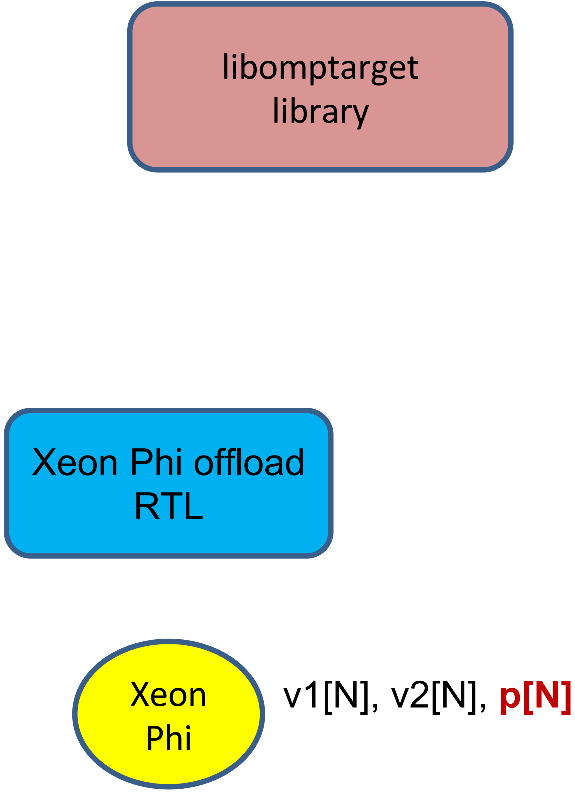
OpenMP 4.0 Example

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



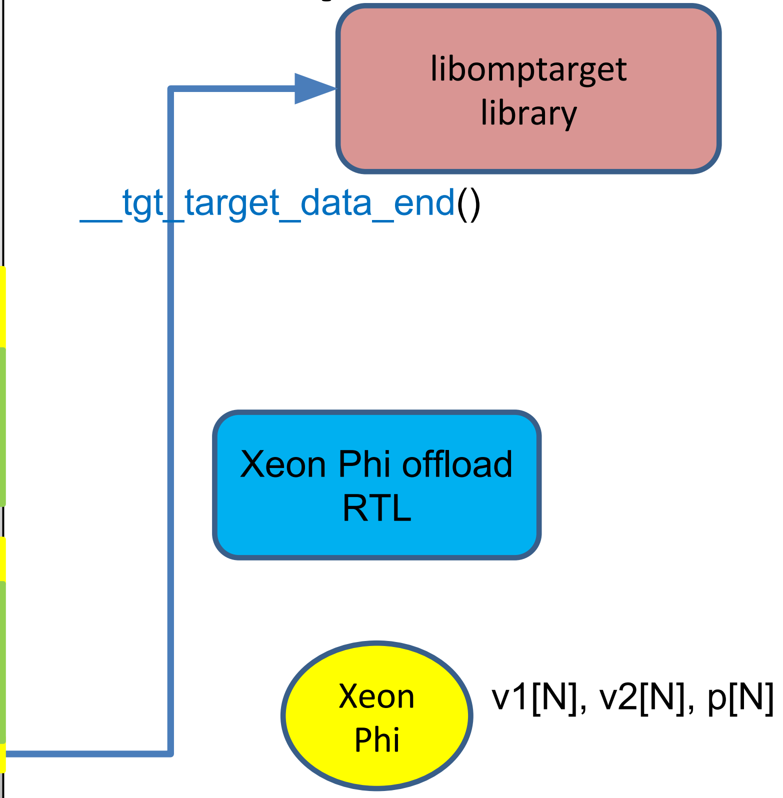
OpenMP 4.0 Example

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



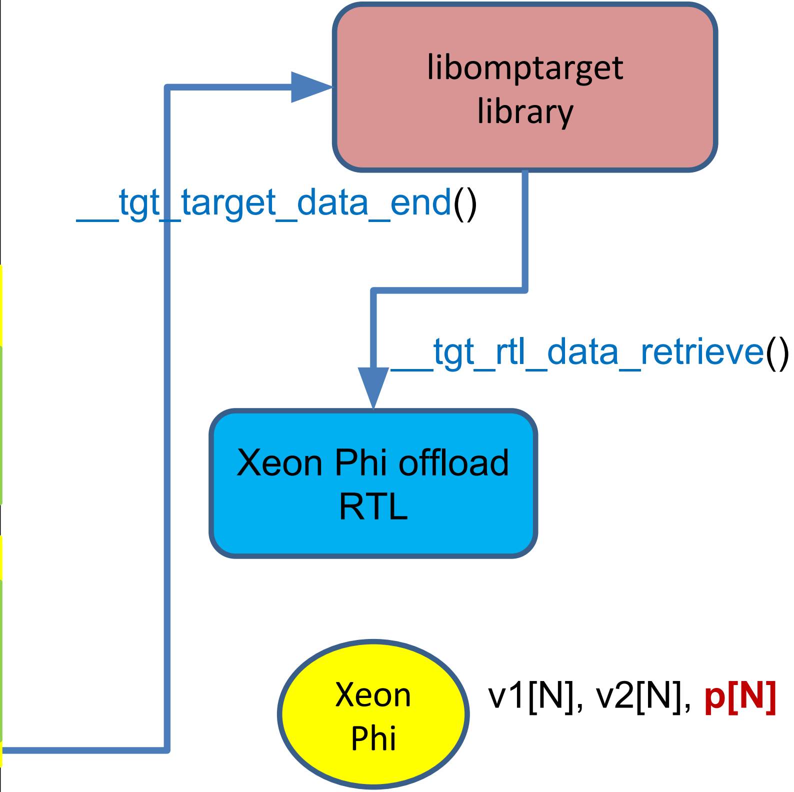
OpenMP 4.0 Example

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



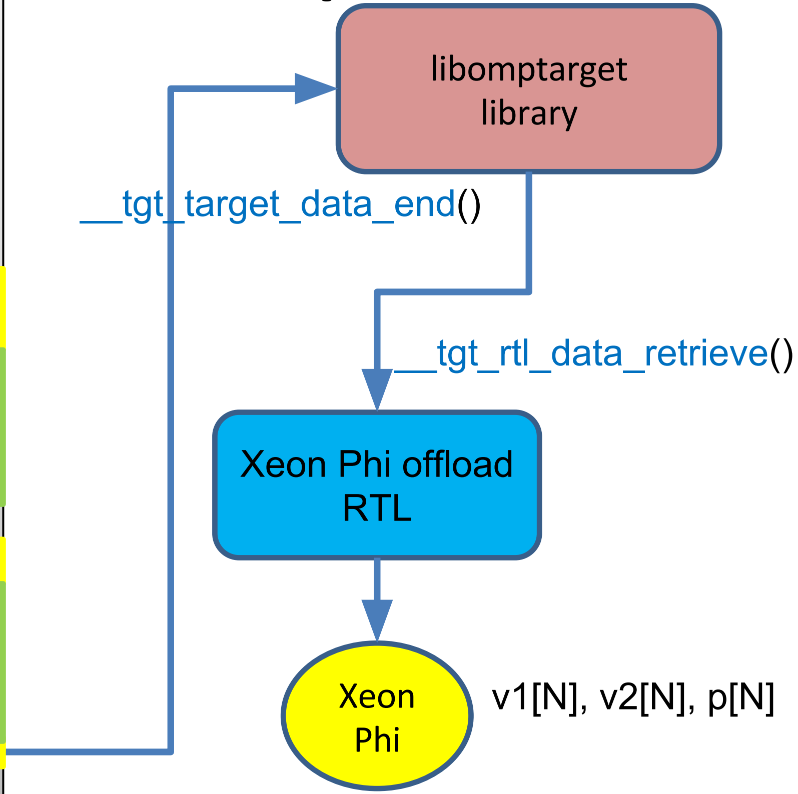
OpenMP 4.0 Example

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



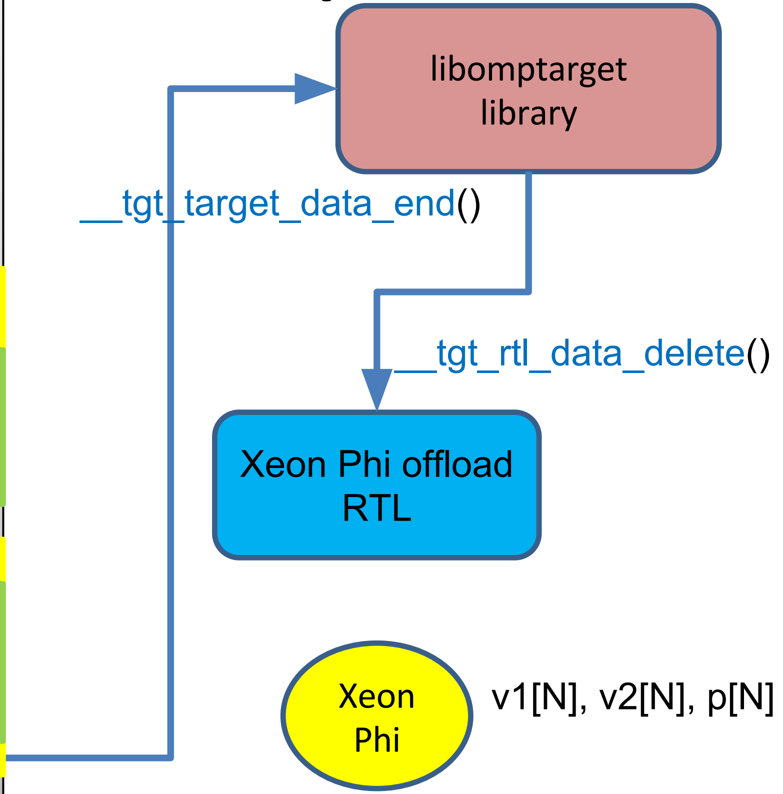
OpenMP 4.0 Example

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



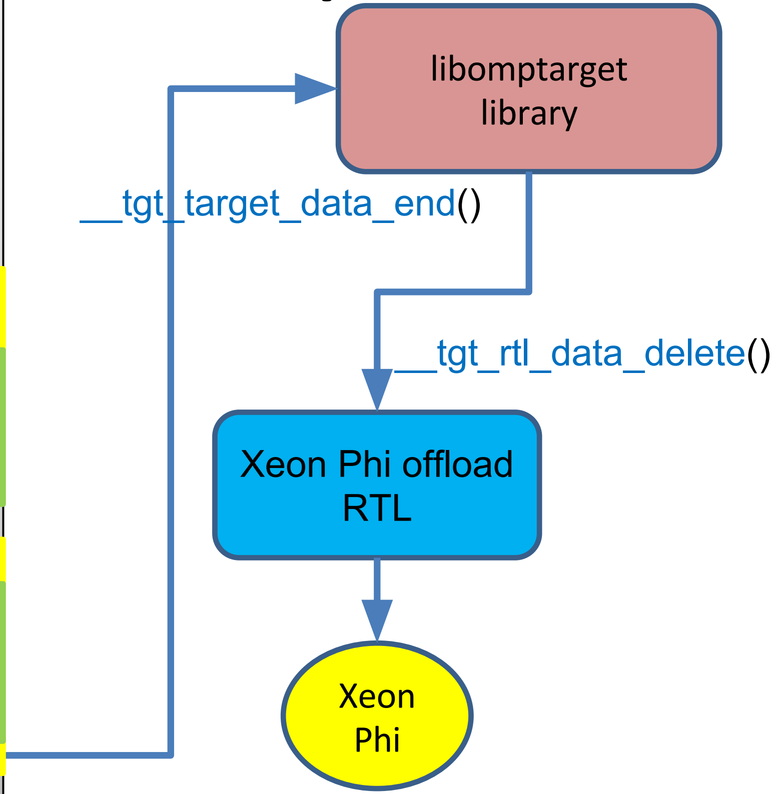
OpenMP 4.0 Example

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



OpenMP 4.0 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(to: v1[:N], v2[:N]) map(from: p[0:N])
 {
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = v1[i] * v2[i];
  init_again(v1, v2, N);
  #pragma omp target update to(v1[:N], v2[:N])
  #pragma omp target
  #pragma omp parallel for
  for (i=0; i<N; i++)
   p[i] = p[i] + (v1[i] * v2[i]);
 }
 output(p, N);
}
```



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.

Agenda

- Accelerator Programming
- OpenMP 4.0 Accelerator Programming Model
- Clang/OpenMP Target-independent Offload Design
- Clang/OpenMP Offloading in Action
- Users of OpenMP-enabled clang

Users of OpenMP-enabled Clang

AMD

– Embraces and enhances OpenMP API for high-performance computing
– Clang-based compiler toolchain for offloading to Kaveri* and Carrizo* hardware
– Plan to contribute Heterogeneous Compute Compiler (HCC) to LLVM community

IBM

– CORAL is the new Dawn of Exascale Computing using Accelerators
– Long time committed supporter and original Founder of OpenMP
– Currently leads as CEO of OpenMP

intel

– Proponent of OpenMP as a universal parallel programming model, both for homogeneous and heterogeneous computing
– Long time committed supporter and original Founder of OpenMP
– Full support of offloading in Clang / LLVM toolchain to Xeon Phi range of processors

*Other names and brands may be claimed as the property of others.

Users of OpenMP-enabled Clang

TEXAS INSTRUMENTS

- Exploring using OpenMP as a unified Multi-Processor SoC (MPSoC) programming model
- Targeting Keystone* and Sitara AM572x* family MPSoCs which combine ARM* Cortex-A15*, ARM Cortex-M4*, C66x* DSP and GPU cores
- Clang with OpenMP 4 offloading provides a single compiler solution for MPSoCs

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

- Depends on OpenMP offloading as a unifying programming model for heterogeneous parallelism
- Clang compiler toolchain for Summit* and Sierra* supercomputers, combining POWER9* CPUs with NVIDIA* Volta* GPUs
- Also, offloading based on Clang compiler toolchain for upcoming Theta* and Aurora* supercomputers with thousands of “Knights Hill” Xeon Phi processors

*Other names and brands may be claimed as the property of others.

Summary

- Offloading is vital to harness power of heterogeneous computing systems
- OpenMP 4 is a uniform multi-platform standard for offloading
- OpenMP 4 and offloading support are under active development in Clang / LLVM
  - Design finished, implementation under way
  - Supported by developers from AMD*, ANL*, IBM*, Intel and Texas Instruments*
- Join us!

*Other names and brands may be claimed as the property of others.