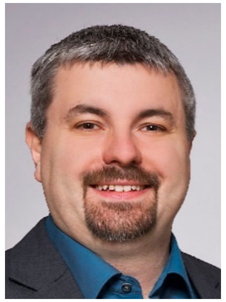
Programming OpenMP

Christian Terboven

Michael Klemm



Agenda (in total 7 Sessions)

- Session 1: OpenMP Introduction
- Session 2: Tasking
- Session 3: Optimization for NUMA and SIMD
- Session 4: What Could Possibly Go Wrong Using OpenMP
- Session 5: Introduction to Offloading with OpenMP
- **Session 6: Advanced Offloading Topics**
  - → Unstructured Data Movement
  - → Asynchronous Offloading
  - → Integration of GPU-Kernels (i.e., HIP)
  - → Detachable (GPU) tasks
  - → Real-World Application Case Study: NWChem
  - → Homework assignments ☺

Programming OpenMP

Review

Christian Terboven
Michael Klemm

Questions?

Jacobi

Example solution: Jacobi basic

```
  while ( err > tol && iter < iter_max ) {
        err = 0.0;

#pragma omp target teams distribute parallel for reduction(max:err)\
            schedule(nonmonotonic:static,1) map(to:A[0:n*m]) map(from:Anew[0:n*m], err)
        for( j = 1; j < n-1; j++) {
            for( i = 1; i < m-1; i++ ) {
                Anew[j *m+ i] = 0.25 * ( A[j     *m+ (i+1)] + A[j     *m+ (i-1)]
                                     +   A[(j-1) *m+ i]     + A[(j+1) *m+ i]);
                err = fmax(err,fabs(Anew[j*m+i]-A[j*m+i]));
            }
        }
         for( j = 1; j < n-1; j++) {
            for( i = 1; i < m-1; i++ ) {
                A[j *m+ i] = Anew[j *m+ i];
            }
        }
        …
        iter++;
    } // end while
```

Example solution: Jacobi data

```
#pragma omp target data map(to:A[0:n*m]) map(alloc:Anew[0:n*m])
    while ( err > tol && iter < iter_max ) {
        err = 0.0;

#pragma omp target teams distribute parallel for reduction(max:err)  \
            schedule(nonmonotonic:static,1)
        for( j = 1; j < n-1; j++) {
            for( i = 1; i < m-1; i++ ) {
                Anew[j *m+ i] = 0.25 * ( A[j     *m+ (i+1)] + A[j     *m+ (i-1)]
                                     +   A[(j-1) *m+ i]     + A[(j+1) *m+ i]);
                err = fmax(err,fabs(Anew[j*m+i]-A[j*m+i]));
            }
        }

#pragma omp target teams distribute parallel for schedule(nonmonotonic:static,1)
        for( j = 1; j < n-1; j++) {
            for( i = 1; i < m-1; i++ ) {
                A[j *m+ i] = Anew[j *m+ i];
            }
        }
        …
        iter++;
    } // end while
```

Old setup (reference numbers from 2021)

```
=> nvidia-smi
Thu May 20 17:21:08 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:61:00.0 Off |                    0 |
| N/A   39C    P0    53W / 300W |      0MiB / 16160MiB |      0%   E. Process |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  Off  | 00000000:62:00.0 Off |                    0 |
| N/A   37C    P0    53W / 300W |      0MiB / 16160MiB |      0%   E. Process |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- clang 12.0.0 with gcc 4.8.5 from CentOS 7.9.2009

Jacobi on GPU / 1

- Task 0: You might want to acquire reference measurements on the host (wo/ GPU)...
  - Skipped…
- Task 1: Get it to the GPU: Parallelize only the one most compute-intensive loop

```
Jacobi relaxation Calculation: 16384 x 16384 mesh with 1 threads and at most 100 iterations. 0 rows out of 16384 on CPU.
    0, 0.250000
   10, 0.250000
   20, 0.250000
   30, 0.250000
   40, 0.250000
   50, 0.250000
   60, 0.250000
   70, 0.250000
   80, 0.250000
   90, 0.250000
 total: 144.992748 s
```

Jacobi on GPU / 2

- Task 2: Improve the data management and the amount of parallelism on the GPU

```
=> ./jacobi.sol.gpu-v100
Jacobi relaxation Calculation: 16384 x 16384 mesh with 1 threads. 0 rows out of 16384 on CPU.
    0, 0.250000
   10, 0.021563
   20, 0.011489
   30, 0.007826
   40, 0.005857
   50, 0.004751
   60, 0.003945
   70, 0.003412
   80, 0.002980
   90, 0.002658
 total: 7.872561 s
```

- Task 3: Optimize that scheduling of iterations for the GPU

```
=> ./jacobi.sol.gpu-v100
Jacobi relaxation Calculation: 16384 x 16384 mesh with 1 threads. 0 rows out of 16384 on CPU.
    0, 0.250000
   10, 0.021563
   20, 0.011489
   30, 0.007826
   40, 0.005857
   50, 0.004751
   60, 0.003945
   70, 0.003412
   80, 0.002980
   90, 0.002658
 total: 5.519289 s
```

Programming OpenMP

GPU: unstructured data movement

Christian Terboven
Michael Klemm

Map variables across multiple target regions

- Optimize sharing data between host and device.
- The **`target data`**, **`target enter data`**, and **`target exit data`** constructs map variables but do not offload code.
- Corresponding variables remain in the device data environment for the extent of the target data region.
- Useful to map variables across multiple target regions.
- The **`target update`** synchronizes an original variable with its corresponding variable.

`target update` Construct Syntax

- Issue data transfers to or from existing data device environment
- Syntax (C/C++)

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

- Syntax (Fortran)

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

- Clauses

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

- Map variables to a device data environment.
- Syntax (C/C++)

```
#pragma omp target enter data clause[[[,] clause]…]
#pragma omp target exit data clause[[[,] clause]…]
```

- Syntax (Fortran)

```
!$omp target enter data clause[[[,] clause]…]
!$omp target exit data clause[[[,] clause]…]
```

- Clauses

```
if([ target enter data :] scalar-expression) OR
   if([ target exit data :] scalar-expression)
device(integer-expression)
map([[map-type-modifier[ ,][map-type-modifier[,]...] map-type:]
locator-list)
depend([depend-modifier,] dependence-type: locator-list)
nowait
```

Code Examples

Map variables to a device data environment

- The host thread executes the data region
- Be careful when using the device clause

Map variables to a device data environment

- The host thread executes the data region
- Be careful when using the device clause

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

Map variables to a device data environment

- The host thread executes the data region
- Be careful when using the device clause

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

host
target
host
target
host

Synchronize mapped variables

- Synchronize the value of an original variable in a host data environment with a corresponding variable in a device data environment

Synchronize mapped variables

- Synchronize the value of an original variable in a host data environment with a corresponding variable in a device data environment

```
#pragma omp target data map(alloc:tmp[:N]) map(to:input[:N]) map(tofrom:res)
{
  #pragma omp target
  #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 to(input[:N])

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

Synchronize mapped variables

- Synchronize the value of an original variable in a host data environment with a corresponding variable in a device data environment

```
#pragma omp target data map(alloc:tmp[:N]) map(to:input[:N]) map(tofrom:res)
{
  #pragma omp target
  #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 to(input[:N])

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

`target data` Construct

```
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 maps variables to the *device data environment*.
  - structured mapping – the device data environment is created for the block of code enclosed by the construct
- v1 and v2 are mapped at each **target** construct.
- p is mapped once by the **target data** construct.

`target enter/exit data` Construct

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

```
void init(float *v1, float *v2, int N) {
  for (int i=0; i<N; i++)
    v1[i] = v2[i] = ...;
#pragma omp target enter data map(alloc:
p[:N])
}

void output(float *p, int N) {
  ...
#pragma omp target exit map(from: p[:N])
}
```

- The **target enter/exit data** construct maps variables to/from the *device data environment*.
  - unstructured mapping – the device data environment can span more than one function
- v1 and v2 are mapped at each **target** construct.
- p is allocated and remains undefined in the device data environment by the **target enter data map(alloc:...)** construct.
- The value of p in the *device data environment* is assigned to the original variable on the host by the **target exit data map(from:...)** construct.

Programming OpenMP

GPU: asynchronous offloading

Christian Terboven
Michael Klemm

Synchronization

- OpenMP target default: synchronous operations
  - CPU thread waits until OpenMP kernel/ movement is completed

- Remember:
  - Use `target` construct to
    - Transfer control from the host to the target device
  - Use `map` clause to
    - Map variables between the host and target device data environments

- Host thread waits until offloaded region completed
  - Use the `nowait` clause for asynchronous execution

- Remember: GPUs only allow for synchronization within a streaming multiprocessor
  - Synchronization or memory fences across SMs not supported due to limited control logic
  - Barriers, critical regions, locks, atomics only apply to the threads within a team
  - No cache coherence between L1 caches

Synchronization

- OpenMP target default: synchronous operations
  - CPU thread waits until OpenMP kernel/ movement is completed

- Remember:
  - Use `target` construct to
    - Transfer control from the host to the target device
  - Use `map` clause to
    - Map variables between the host and target device dat

- Host thread waits until offloaded region completed
  - Use the `nowait` clause for asynchronous execution

```
count = 500;
#pragma omp target map(to:b,c,d) map(from:a)
{
  #pragma omp parallel for
    for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
    }
}
a0 = a[0];
```

- Remember: GPUs only allow for synchronization within a streaming multiprocessor
  - Synchronization or memory fences across SMs not supported due to limited control logic
  - Barriers, critical regions, locks, atomics only apply to the threads within a team
  - No cache coherence between L1 caches

Synchronization

- OpenMP target default: synchronous operations
  - CPU thread waits until OpenMP kernel/ movement is completed

- Remember:
  - Use `target` construct to
    - Transfer control from the host to the target device
  - Use `map` clause to
    - Map variables between the host and target device dat

- Host thread waits until offloaded region completed
  - Use the `nowait` clause for asynchronous execution

```
count = 500;
#pragma omp target map(to:b,c,d) map(from:a)
{
  #pragma omp parallel for
    for (i=0; i<count; i++) {
      a[i] = b[i] * c + d;
    }
}
a0 = a[0];
```


- Remember: GPUs only allow for synchronization within a streaming multiprocessor
  - Synchronization or memory fences across SMs not supported due to limited control logic
  - Barriers, critical regions, locks, atomics only apply to the threads within a team
  - No cache coherence between L1 caches

Asynchronous Offloading

- A host task is generated that encloses the target region.
- The **nowait** clause specifies that the encountering thread does not wait for the target region to complete.
- The **depend** clause can be used for ensuring the order of execution with respect to other tasks.

```
subroutine vec_mult(p, v1, v2, N)
  real, dimension(*) :: p, v1, v2
  integer :: N, i
  call init(v1, v2, N)

!$omp target data map(tofrom:v1(1:N), v2(1:N), p(1:N))
!$omp target nowait
!$omp parallel do
  do i=1, N/2
    p(i) = v1(i) * v2(i)
  end do
!$omp end target

!$omp target nowait
!$omp parallel do
  do i=N/2+1, N
    p(i) = v1(i) * v2(i)
  end do
!$omp end target
!$omp end target data

  call output(p, N)
end subroutine
```

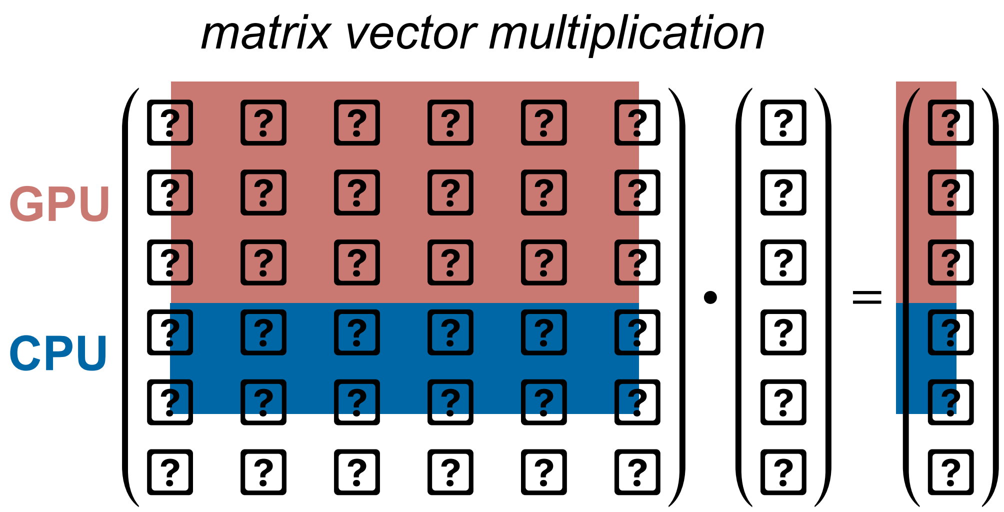
target task
A mergeable and untied task that is generated by a **target**, **target enter data**, **target exit data** or **target update** construct.

Remark on Heterogeneous Computing

Slides are taken from the lecture High-Performance Computing at RWTH Aachen University
Authors include: Sandra Wienke, Julian Miller

Heterogeneous Computing

- Heterogeneous Computing
  - CPU & GPU are (fully) utilized
- Challenge: load balancing
- Domain decomposition
  - If load is known beforehand, static decomposition
  - Exchange data if needed (e.g. halos)



Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately

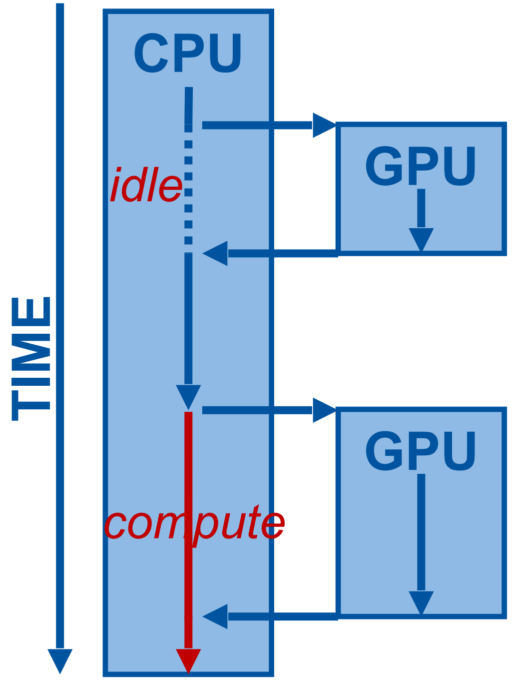
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)

Asynchronous Operations

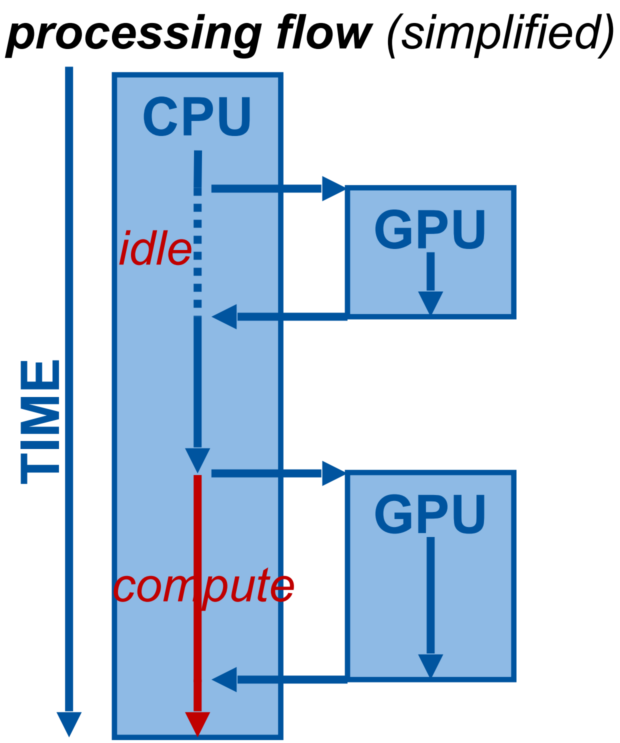
- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)

***processing flow** (simplified)*



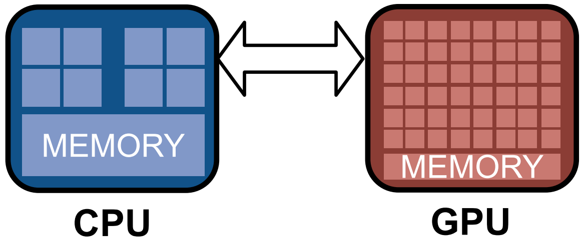
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions

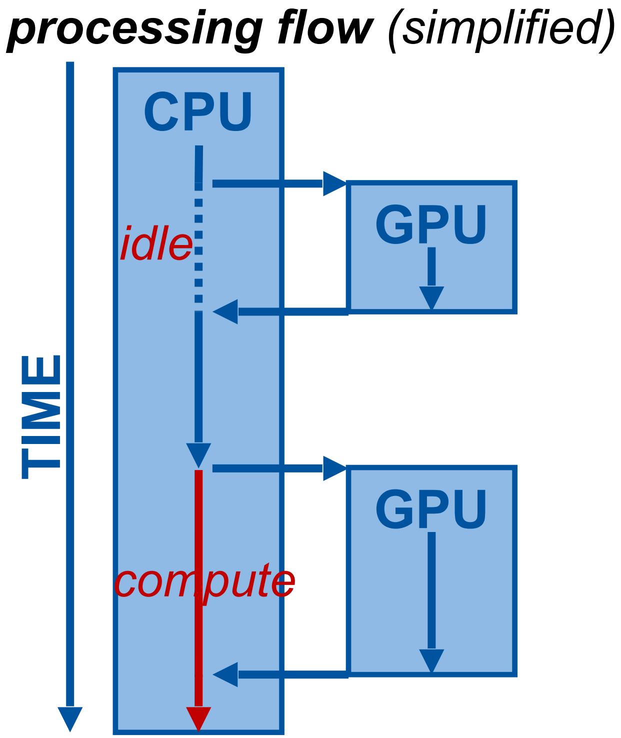


Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions

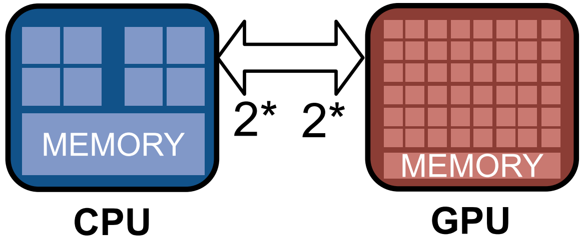


<num>* Can be executed simultaneously



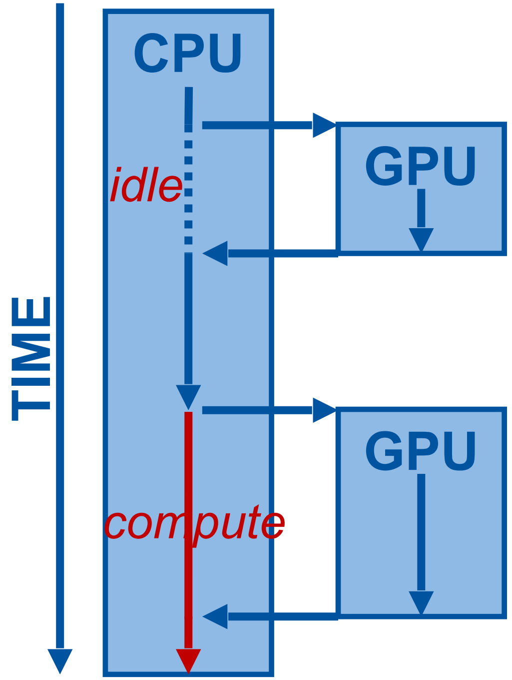
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions



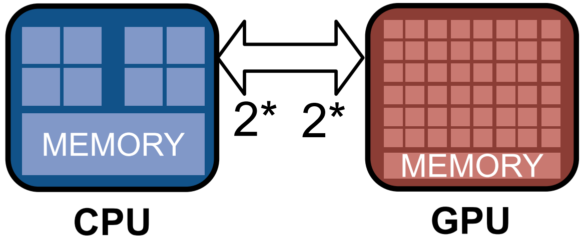
<num>* Can be executed simultaneously

processing flow *(simplified)*



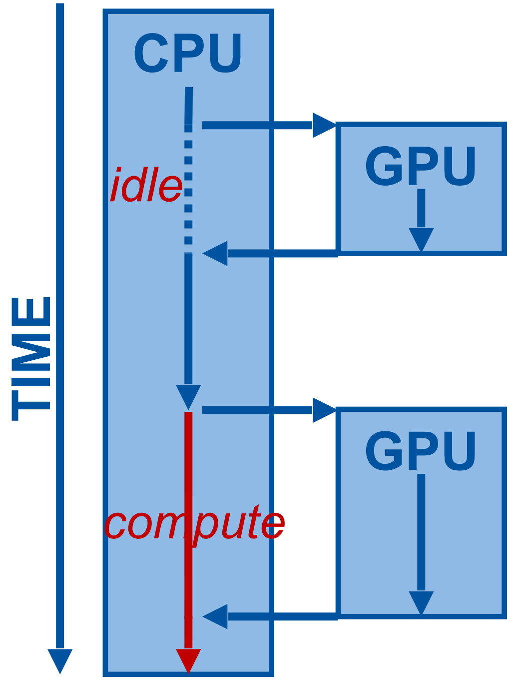
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions
  3. Overlap of data transfers and computation



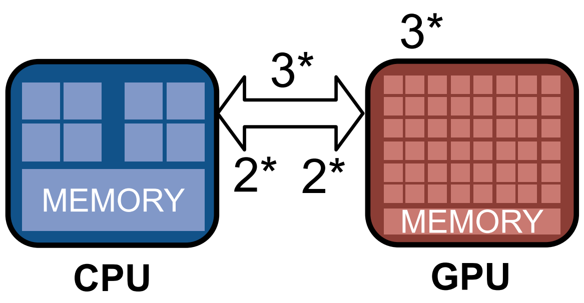
<num>* Can be executed simultaneously

processing flow *(simplified)*



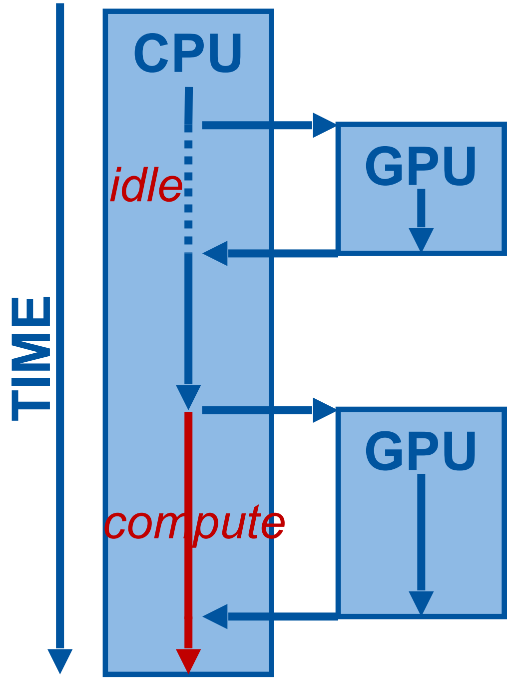
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions
  3. Overlap of data transfers and computation



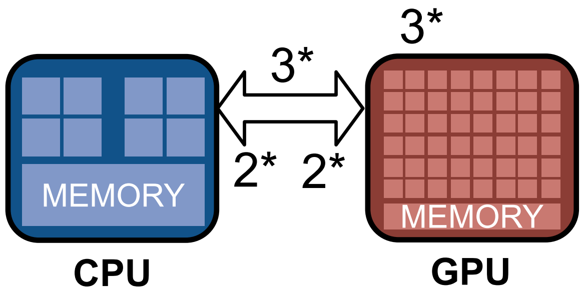
<num>* Can be executed simultaneously

processing flow *(simplified)*

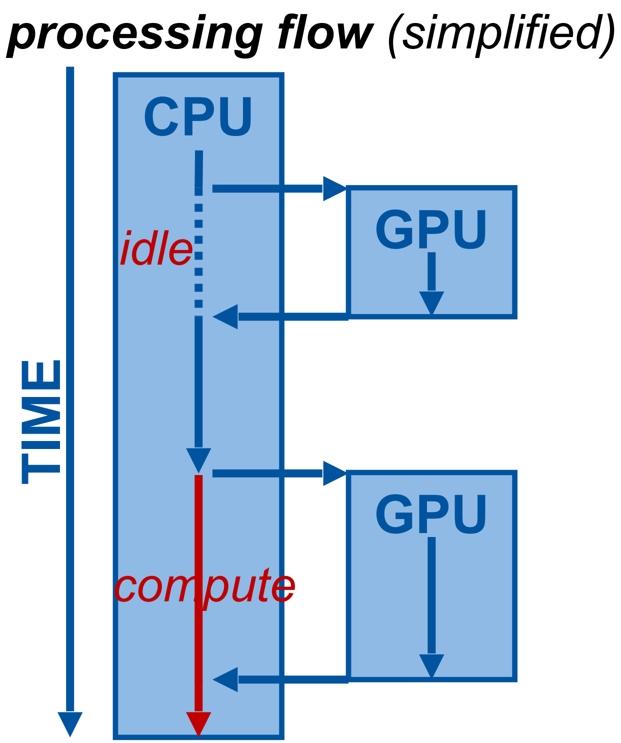


Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions
  3. Overlap of data transfers and computation
  4. Simultaneous execution of several kernels (if resources are available)

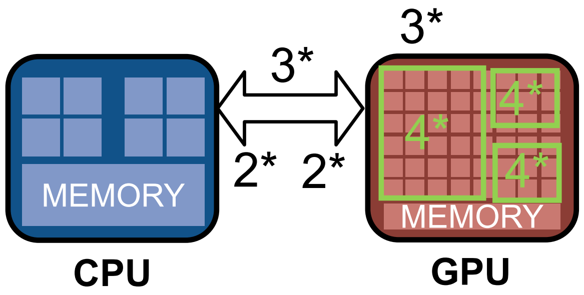


<num>* Can be executed simultaneously



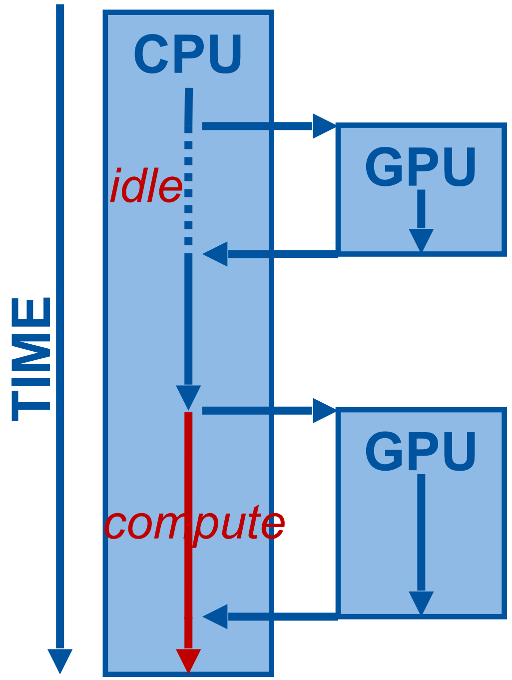
Asynchronous Operations

- Definition
  - Synchronous: Control does not return until accelerator action is complete
  - Asynchronous: Control returns immediately
- Asynchronicity allows, e.g.,
  1. Heterogeneous computing (CPU + GPU)
  2. Overlap of PCIe transfers in both directions
  3. Overlap of data transfers and computation
  4. Simultaneous execution of several kernels (if resources are available)



<num>* Can be executed simultaneously

processing flow *(simplified)*



Asynchronous Operations

- Default: synchronous operations
- Asynchronous operations with tasks
  - Execute asynchronously with dependency: `task depend`
  - Synchronize tasks: `taskwait`
- Synchronize async operations → `taskwait` directive
  - Wait for completion of an asynchronous activity

Asynchronous Operations

- Default: synchronous operations
- Asynchronous operations with tasks
  - Execute asynchronously with dependency: `task depend`
  - Synchronize tasks: `taskwait`
- Synchronize async operations → `taskwait` directive
  - Wait for completion of an asynchronous activity

```
#pragma omp target map(…) nowait depend(out:gpu_data)
// do work on device
#pragma omp task depend(out:cpu_data)
// do work on host
#pragma omp task depend(in:cpu_data) depend(in:gpu_data)
// combine work on host
#pragma omp taskwait
// wait for all tasks
```

Code Examples

Tasks and Target Example / 1

```
void vec_mult_async(float* p, float* v1, float* v2, int N)
{
#pragma omp target enter data map(alloc: v1[:N], v2[:N])

  #pragma omp target nowait depend(out: v1, v2)
    compute(v1, v2, N);

  #pragma omp task
    other_work(); // execute asynchronously on host device
                  // other_work does not involve v1 and v2

  #pragma omp target map(from:p[0:N]) nowait depend(in: v1,
v2)
  {
    #pragma omp parallel for
    for (int i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
  }

  #pragma omp taskwait

#pragma omp target exit data map(release: v1[:N], v2[:N])
}
```

- If other_work() does not involve v1 and v2, the encountering thread on the host will execute the task asynchronously.
- The dependency requirement between the two target tasks must be satisfied before the second target task starts execution.
- The **taskwait** directive ensures all sibling tasks complete before proceeding to the next statement.

Tasks and Target Example / 2

```
void vec_mult_async(float* p, float* v1, float* v2, int N)
{
#pragma omp target enter data map(alloc: v1[:N], v2[:N])

  #pragma omp target nowait depend(out: v1, v2)
    compute(v1, v2, N);

  #pragma omp target update from(v1[:N], v2[:N]) depend(inout: v1,
v2)

  #pragma omp task depend(inout: v1, v2)
    compute_on_host(v1, v2); // execute asynchronously on host
device
                             // other_work involves v1, v2

  #pragma omp target update to(v1[:N], v2[:N]) depend(inout: v1,
v2)

  #pragma omp target map(from:p[0:N]) nowait depend(in: v1, v2)
  {
    #pragma omp parallel for
    for (int i=0; i<N; i++)
      p[i] = v1[i] * v2[i];
  }

  #pragma omp taskwait

#pragma omp target exit data map(release: v1[:N], v2[:N])
}
```

- If compute_on_host() updates v1 and v2, the **depend** clause must be specified to ensure the execution of the target task and the explicit task respects the dependency.
- Since we update v1 and v2 on the host in compute_on_host(), we need to update the data results from compute() on the device to the host.
- After completion of compute_on_host(), the data in the target device is updated with the result.
- The **update** clause is required before and after the explicit task.

Programming OpenMP

Hands-on Exercises: Jacobi

Christian Terboven
Michael Klemm

Jacobi on GPU

- Task 0: You might want to acquire reference measurements on the host (wo/ GPU)...
- Task 1: Get it to the GPU: Parallelize only the one most compute-intensive loop
- Task 2: Improve the data management and the amount of parallelism on the GPU
- Task 3: Optimize that scheduling of iterations for the GPU
- Task 4: Make the code as fast as you can :-). Use sample codes in exercises/<C,Fortran>/Jacobi2 for hints