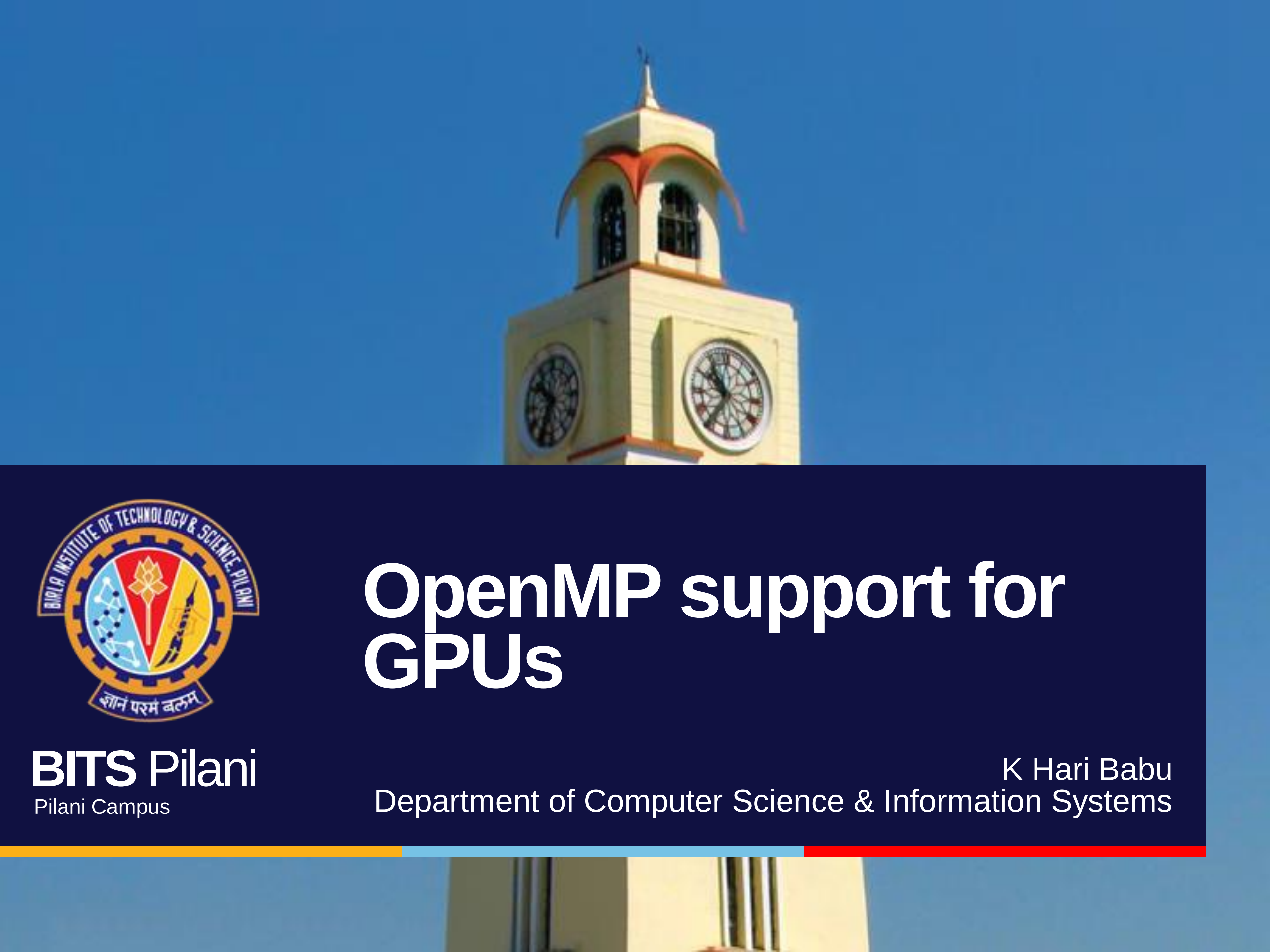


OpenMP support for GPUs

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

Host-device model

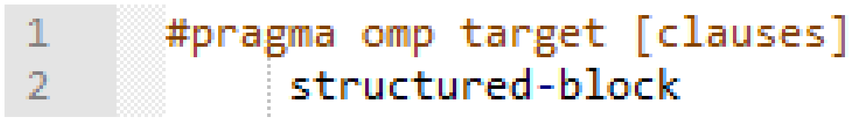
- Since version 4.0 , OpenMP supports heterogeneous systems. OpenMP uses TARGET construct to offload execution from the host to the target device(s), and hence the directive name
  - In addition, the associated data needs to be transferred to the device(s) as well. Once transferred, the target device owns the data and accesses by the host during the execution of the target region is forbidden
- A host/device model is generally used by OpenMP for offloading:
  - normally there is only one single host: e.g. CPU
  - one or multiple target devices of the same kind: e.g. coprocessor, GPU, FPGA, ...
  - unless with unified shared memory, the host and device have separate memory address space

Device execution model

- The execution on the device is host-centric
    1. the host creates the data environments on the device(s)
    2. the host maps data to the device data environment
    3. the host offloads OpenMP target regions to the target device to be executed
    4. the host transfers data from the device to the host
    5. the host destroys the data environment on the device

TARGET construct

- The TARGET construct consists of a target directive and an execution region. It is used to transfer both the control flow from the host to the device and the data between the host and device.

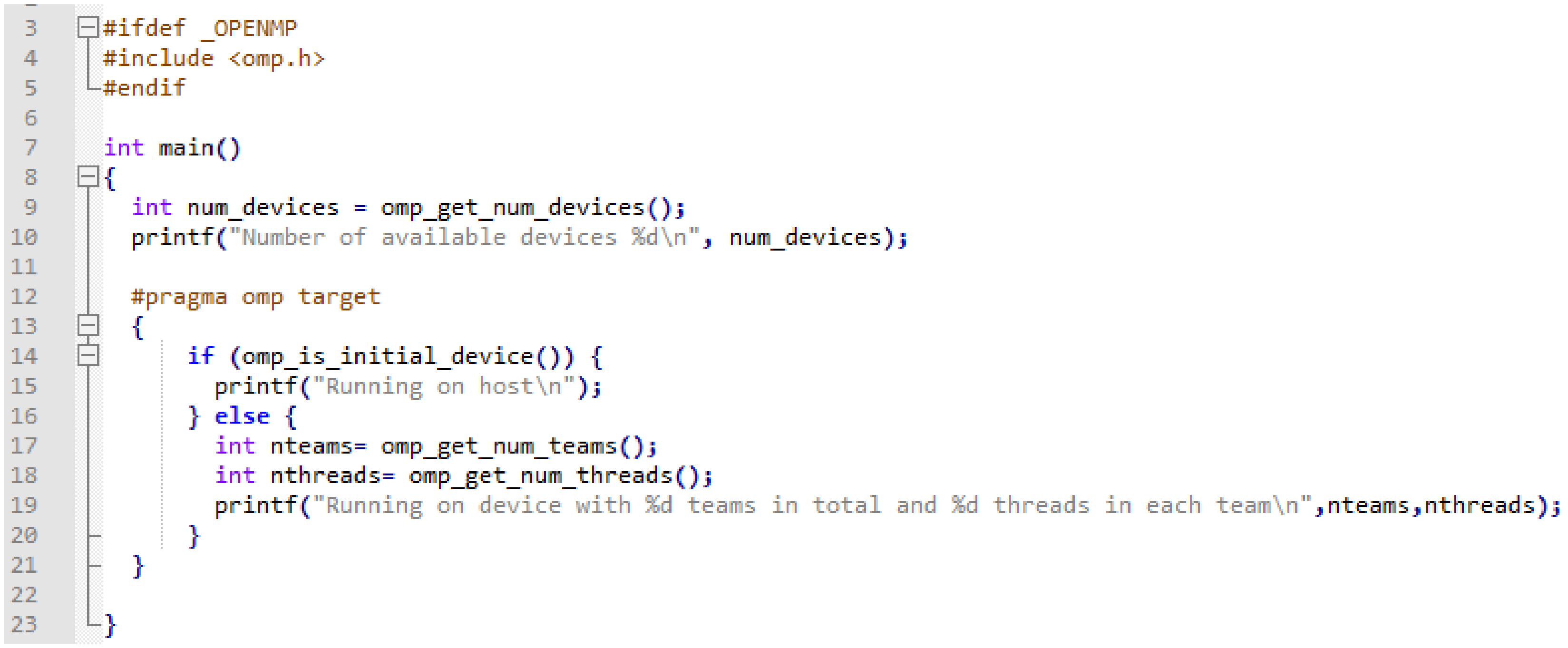


```
#pragma omp target [clauses]
      structured-block
```

clause:

    if([target:] scalar-expression)
    device(integer-expression)
    private(list)
    firstprivate(list)
    map([map-type:] list)
    is_device_ptr(list)
    defaultmap(tofrom:scalar)
    nowait
    depend(dependence-type : list)

TARGET construct



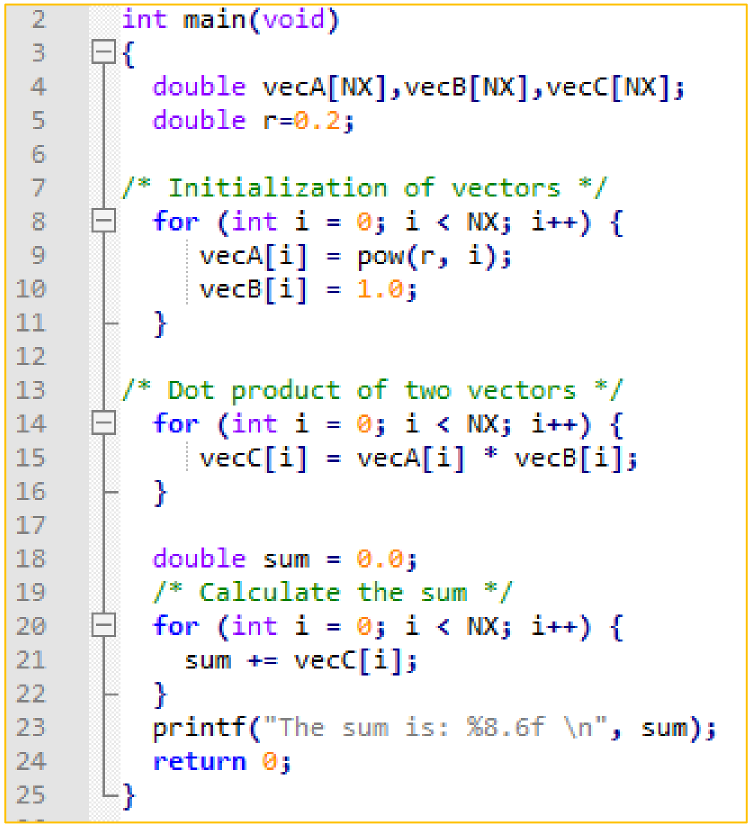
```
#ifdef _OPENMP
#include <omp.h>
#endif

int main()
{
  int num_devices = omp_get_num_devices();
  printf("Number of available devices %d\n", num_devices);

  #pragma omp target
  {
      if (omp_is_initial_device()) {
        printf("Running on host\n");
      } else {
        int nteams= omp_get_num_teams();
        int nthreads= omp_get_num_threads();
        printf("Running on device with %d teams in total and %d threads in each team\n",nteams,nthreads);
      }
  }

}
```

TARGET construct

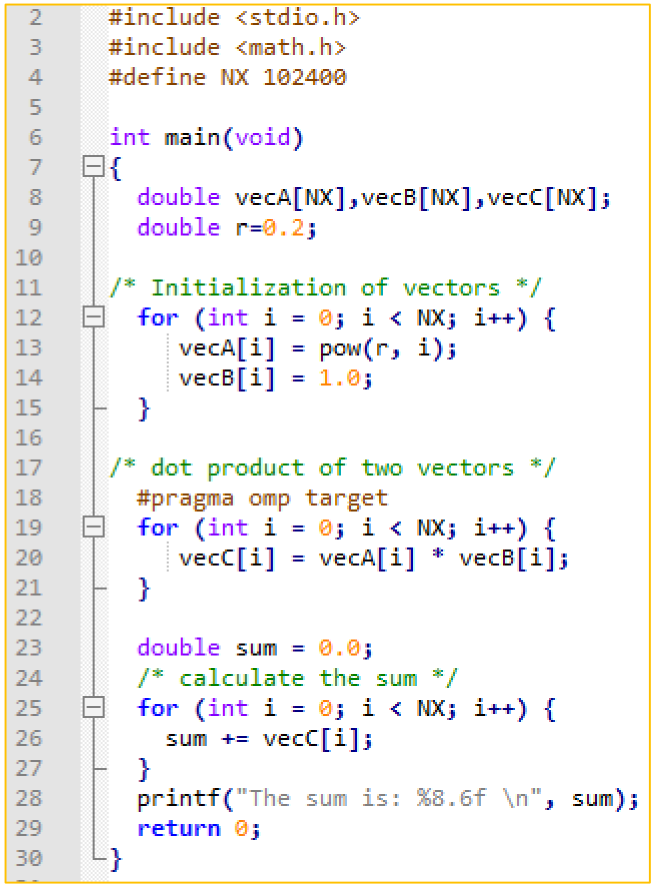


```c
int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
     vecA[i] = pow(r, i);
     vecB[i] = 1.0;
  }

/* Dot product of two vectors */
  for (int i = 0; i < NX; i++) {
     vecC[i] = vecA[i] * vecB[i];
  }

  double sum = 0.0;
  /* Calculate the sum */
  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```



```c
#include <stdio.h>
#include <math.h>
#define NX 102400

int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
     vecA[i] = pow(r, i);
     vecB[i] = 1.0;
  }

/* dot product of two vectors */
  #pragma omp target
  for (int i = 0; i < NX; i++) {
     vecC[i] = vecA[i] * vecB[i];
  }

  double sum = 0.0;
  /* calculate the sum */
  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```

Teams

- The **omp teams** directive creates a collection of thread teams. The master thread of each team executes the teams region.
- For a target region without specifying *teams*, a single team is created and its master thread executes the target region.
- The **omp teams** directive must be strictly nested inside the target region, that is, no code can exist between the **omp target** directive and the **omp teams** directive. You must strictly nest the following OpenMP regions inside the teams region:
  - omp distribute
  - omp distribute simd
  - omp distribute parallel for
  - omp distribute parallel for simd

DISTRIBUTE

- The DISTRIBUTE construct is a coarsely worksharing construct which distributes the loop iterations across the master threads in the teams, but no worksharing within the threads in one team
    - No implicit barrier at the end of the construct and no guarantee about the order the teams will execute.
- To further create threads within each team and distritute loop iterations across threads, we will use the PARALLEL FOR constructs.

DISTRIBUTE

- TEAMS DISTRIBUTE construct
  - Coarse-grained parallelism
  - Spawns multiple single-thread teams
  - No synchronization of threads in different teams
- PARALLEL FOR/DO construct
  - Fine-grained parallelism
  - Spawns many threads in one team
  - Threads can synchronize in a team

DISTRIBUTE

- TEAMS DISTRIBUTE construct
  - Coarse-grained parallelism
  - Spawns multiple single-thread teams
  - No synchronization of threads in different teams
- PARALLEL FOR/DO construct
  - Fine-grained parallelism
  - Spawns many threads in one team
  - Threads can synchronize in a team

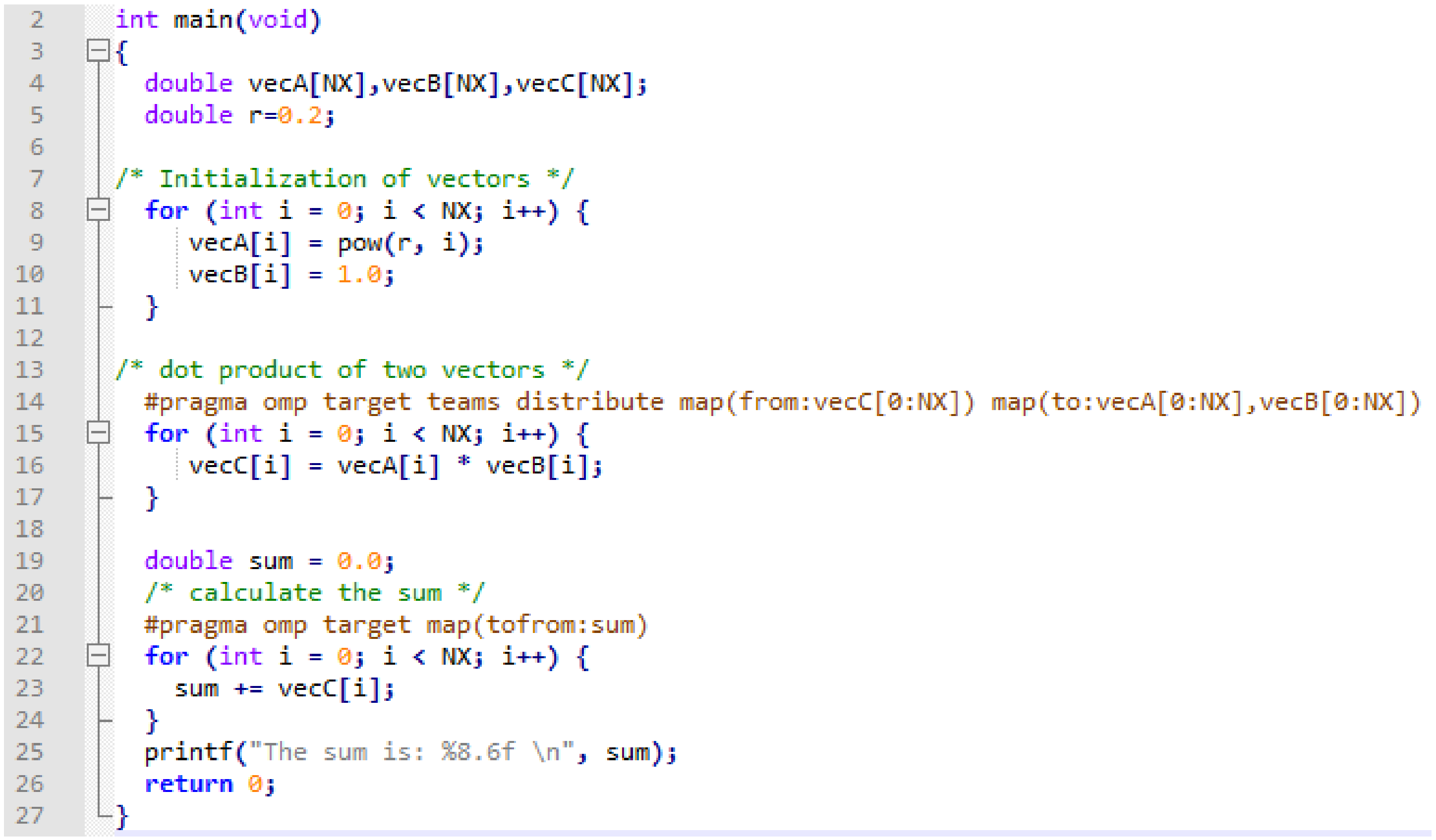
Data mapping

- Due to distinct memory spaces on host and device, transferring data becomes inevitable. A combination of both explicit and implicit data mapping is used.
- The MAP clause on a device construct explicitly specifies how items are mapped from the host to the device data environment. The common mapped items consist of arrays(array sections), scalars, pointers, and structure elements.

Data mapping

- The various forms of the map clause are summarized in the following table

map([map-type]:list)	map clause
map(to:list)	On entering the region, variables in the list are initialized on the device using the original values from the host
map(from:list)	At the end of the target region, the values from variables in the list are copied into the original variables on the host. On entering the region, the initial value of the variables on the device is not initialized
map(tofrom:list)	the effect of both a map-to and a map-from
map(alloc:list)	On entering the region, data is allocated and uninitialized on the device
map(list)	equivalent to ``map(tofrom:list)``



```c
int main(void)
{
  double vecA[NX],vecB[NX],vecC[NX];
  double r=0.2;

/* Initialization of vectors */
  for (int i = 0; i < NX; i++) {
     vecA[i] = pow(r, i);
     vecB[i] = 1.0;
  }

/* dot product of two vectors */
  #pragma omp target teams distribute map(from:vecC[0:NX]) map(to:vecA[0:NX],vecB[0:NX])
  for (int i = 0; i < NX; i++) {
     vecC[i] = vecA[i] * vecB[i];
  }

  double sum = 0.0;
  /* calculate the sum */
  #pragma omp target map(tofrom:sum)
  for (int i = 0; i < NX; i++) {
    sum += vecC[i];
  }
  printf("The sum is: %8.6f \n", sum);
  return 0;
}
```

References

BITS Pilani

- Chapter 6, Multicore and GPU Programming An Integrated Approach, Gerassimos Barlas, Morgan Kaufmann

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You