StarPU task-based programming hands-on session

Introduction to task-based programming with StarPU

STORM Team
Inria – LaBRI
starpu-devel@inria.fr
1

Introduction to task-based programming with StarPU
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];

/* ... fill vector ... */
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
                           (uintptr_t)vector, NX, sizeof(vector[0]));
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
   (uintptr_t)vector, NX, sizeof(vector[0]));

starpu_task_insert(
   &scal_cl,
   STARPU_RW, vector_handle,
   STARPU_VALUE, &factor, sizeof(factor), 0);
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
                           (uintptr_t)vector, NX, sizeof(vector[0]));

starpu_task_insert(
    &scal_cl,
    STARPU_RW, vector_handle,
    STARPU_VALUE, &factor, sizeof(factor),
    0);

starpu_task_wait_for_all();
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0, 
    (uintptr_t)vector, NX, sizeof(vector[0]));

starpu_task_insert(
    &scal_cl, 
    STARPU_RW, vector_handle, 
    STARPU_VALUE, &factor, sizeof(factor), 
    0);

starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);

/* ... display vector ... */
```
Terminology

- Codelet
- Task
- Data handle
Definition: A Codelet

A Codelet...  
- ... relates an abstract computation kernel to its implementation(s)  
- ... can be instantiated into one or more tasks  
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more tasks
- ... defines characteristics common to a set of tasks

```plaintext
scal_cl
```

Task 1: will perform a 'scal' kernel
Definition: A Codelet

A Codelet...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **tasks**
Definition: A Codelet

A **Codelet**... 
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **tasks**

![Diagram of Codelet and Tasks](image-url)
Definition: A Task

A Task . . .

- . . . is an instantiation of a **Codelet**
- . . . atomically executes a kernel from its beginning to its end
- . . . receives some input
- . . . produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

[Diagram of Codelet] scal_cl
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

... is an instantiation of a **Codelet**

... atomically executes a kernel from its beginning to its end

... receives some input

... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Task

A Task . . .

- . . . is an instantiation of a **Codelet**
- . . . atomically executes a kernel from its beginning to its end
- . . . receives some input
- . . . produces some output
Definition: A Task

A Task...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output
Definition: A Data Handle

A Data Handle...
- ... designates a piece of data managed by StarPU
- ... is typed (vector, matrix, etc.)
- ... can be passed as input/output for a Task
Elementary API

- Declaring a codelet
- Declaring and Managing Data
- Writing a Kernel Function
- Submitting a task
- Waiting for submitted tasks
Declaring a Codelet

Define a struct `starpu_codelet`

```c
struct starpu_codelet scal_cl = {
    ... 
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func },
    ...
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`
- Declare the number of data pieces used by the kernel
  - Here: A single vector

```
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func },
    .nbuffers = 1,
    ...
};
```
Declaring a Codelet

Define a `struct starpu_codelet`

- Plug the kernel function
  - Here: `scal_cpu_func`
- Declare the number of data pieces used by the kernel
  - Here: A single vector
- Declare how the kernel accesses the piece of data
  - Here: The vector is scaled in-place, thus R/W

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Declaring and Managing Data

Put data under StarPU control
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```c
float vector[NX];
/* ... fill data ... */
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control

```c
float vector[NX];
/* ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
  (uintptr_t)vector, NX, sizeof(vector[0]));
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle

```c
float vector[NX];
/∗ ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
    (uintptr_t)vector, NX, sizeof(vector[0]));

/∗ ... use the vector through the handle ... */
```
Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle
- Unregister the piece of data
  - The handle is destroyed
  - The vector is now back under user control

```c
float vector[NX];
/* ... fill data ... */

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle, 0,
                          (uintptr_t)vector, NX, sizeof(vector[0]));

/* ... use the vector through the handle ... */

starpu_data_unregister(vector_handle);
```
Writing a Kernel Function

- Every kernel function has the same C prototype

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];

    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    ...
}
```
Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector’s handle
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument
- Compute the vector scaling

```c
void scal_cpu_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);

    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor);

    unsigned i;
    for (i = 0; i < n; i++)
        vector[i] *= factor;
}
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure

```c
1 starpu_task_insert(&scal_cl
2     ... ) ;
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data

```
    starpu_task_insert(&scal_cl,
                       STARPU_RW, vector_handle,
                       ...); 
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data

```c
starpu_task_insert(&scal_cl,
STARPURW, vector_handle,
STARPUVALUE, &factor, sizeof(factor),
...);
```
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

```c
starpu_task_insert(&scal_cl ,
    STARPU_RW , vector_handle ,
    STARPU_VALUE , &factor , sizeof(factor) ,
    0);
```
Submitting a task

The `starpu_task_insert` call
  - **Inserts** a task in the StarPU DAG

Arguments
  - The codelet structure
  - The StarPU-managed data
  - The small-size inline data
  - 0 to mark the end of arguments

Notes
  - The task is submitted non-blocking
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
- ... following the natural order of the program
Submitting a task

The `starpu_task_insert` call

- **Inserts** a task in the StarPU DAG

**Arguments**

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

**Notes**

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks’ data...
- ... following the **natural** order of the program
- This is the **Sequential Task Flow Paradigm**
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

```c
/* non-blocking task submits */
starpu_task_insert(...) ;
starpu_task_insert(...) ;
starpu_task_insert(...) ;
starpu_task_insert(...) ;
... 
```
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```c
/* non-blocking task submits */
starpu_task_insert(...);
starpu_task_insert(...);
starpu_task_insert(...);
```
...
Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```c
/* non-blocking task submits */
starpu_task_insert(...);
starpu_task_insert(...);
starpu_task_insert(...);
...

/* wait for all task submitted so far */
starpu_task_wait_for_all();
```
Basic Example: Scaling a Vector

```c
float factor = 3.14;
float vector[NX];
starpu_data_handle_t vector_handle;

/* ... fill vector ... */

starpu_vector_data_register(&vector_handle, 0,
   (uintptr_t)vector, NX, sizeof(vector[0]))
;

starpu_task_insert(
   &scal_cl,
   STARPU_RW, vector_handle,
   STARPU_VALUE, &factor, sizeof(factor), 0);

starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);

/* ... display vector ... */
```
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels

```c
struct starpu_codelet scal_cl = {
    .cpu_func = { scal_cpu_func,
                 scal_sse_cpu_func, scal_avx_cpu_func },
    .nbuffers = 1,
    .modes = { STARPU_RW },
};
```
Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels
- Kernels implementations for accelerator devices
  - OpenCL, NVidia Cuda kernels

```c
struct starpu_codelet scal_cl = {
  .cpu_func   = { scal_cpu_func ,
                 scal_sse_cpu_func , scal_avx_cpu_func },
  .opencl_func = { scal_cpu_opencl },
  .cuda_func   = { scal_cpu_cuda  },
  .nbuffers    = 1,
  .modes       = { STARPU_RW },
};
```
Writing a Kernel Function for CUDA
Writing a Kernel Function for CUDA

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)
    ...
}
```
Writing a Kernel Function for **CUDA**

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    unsigned threads_per_block = 64;
    unsigned nbblocks = (n+threads_per_block-1)/threads_per_block;

    ...
}
```
Writing a Kernel Function for CUDA

```c
extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    unsigned threads_per_block = 64;
    unsigned nblocks = (n + threads_per_block - 1) / threads_per_block;

    vector_mult_cuda<<<nblocks, threads_per_block, 0, starpu_cuda_get_local_stream()>>>(n, vector, factor);
    cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```
Writing a Kernel Function for **CUDA**

```c
static __global__ void vector_mult_cuda(unsigned n, float *vector, float factor) {
    unsigned i = blockIdx.x * blockDim.x + threadIdx.x;

    ...
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    unsigned threads_per_block = 64;
    unsigned nblocks = (n + threads_per_block - 1) / threads_per_block;

    vector_mult_cuda<<<nblocks, threads_per_block, 0, 
    starpu_cuda_get_local_stream()>>>(n, vector, factor);

cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```
Writing a Kernel Function for CUDA

```c
static __global__ void vector_mult_cuda(unsigned n,
                                        float *vector, float factor){
    unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        vector[i] *= factor;
}

extern "C" void scal_cuda_func(void *buffers[], void *cl_arg) {
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    unsigned threads_per_block = 64;
    unsigned nblocks = (n+threads_per_block-1)/threads_per_block;

    vector_mult_cuda<<<nblocks, threads_per_block,0, 
                 starpu_cuda_get_local_stream()>>>(n, vector, factor);
    cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```
Writing a Kernel Function for OpenCL

```
__kernel void vector_mult_opencl(__global float *val,
                                unsigned n, float factor) {
    for (unsigned i = 0; i < n; i++)
        val[i] *= factor;
}

extern "C" void scal_opencl_func(void *buffers[], void *cl_arg)
{
    struct starpu_vector_interface *vector_handle = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
    float factor;
    starpu_codelet_unpack_args(cl_arg, &factor)

    cl_kernel kernel;
    cl_command_queue queue;
    starpu_opencl_load_kernel(&kernel, &queue, &opencl_program,
                               "vector_mult_opencl", devid);
    clSetKernelArg(kernel, 0, sizeof(val), &val);
    ...
    clEnqueueNDRangeKernel(queue, kernel, 1, NULL, ...);
```
Declaring a codelet in **Fortran**

- Defined as a C_PTR allocated by StarPU

```fortran
1     TYPE(C_PTR) :: scal_cl = C_NULL_PTR
2
3     scal_cl = fstarpu_codelet_allocate()
```
Declaring a codelet in Fortran

- Defined as a C_PTR allocated by StarPU
- Plug the kernel function

```fortran
TYPE(C_PTR):: scal_cl = C_NULL_PTR
scal_cl = fstarpu_codelet_allocate()
CALL fstarpu_codelet_add_cpu_func(scal_cl,
   C_FUNLOC(scal_cpu_func))
```
Declaring a codelet in Fortran

- Defined as a C_PTR allocated by StarPU
- Plug the kernel function
- Declare data

```fortran
TYPE(C_PTR):: scal_cl = C_NULL_PTR
scal_cl = fstarpu_codelet_allocate()
CALL fstarpu_codelet_add_cpu_func(scal_cl, C_FUNLOC(scal_cpu_func))
CALL fstarpu_codelet_add_buff(scal_cl, FSTARPU_RW)
```
Writing a Kernel Function in Fortran

- Every kernel function has the same Fortran prototype

```fortran
recursive subroutine scal_cpu_func(buffers, cl_args) bind(c)
type(c_ptr), value, intent(in) :: buffers, cl_args
...
```
Writing a Kernel Function in Fortran

- Every kernel function has the same Fortran prototype
- Get vector's number of elements and base pointer

```fortran
recursive subroutine scal_cpu_func(buffers, cl_args) bind(c)
  type(c_ptr), value, intent(in) :: buffers, cl_args

  integer :: n
  real(8), dimension(:), pointer :: val

  n = fstarpu_vector_get_nx(buffers, 0)
  call c_f_pointer(fstarpu_vector_get_ptr(buffers, 0), &
                  val, shape=[n])

  ...
```
Writing a Kernel Function in Fortran

- Every kernel function has the same Fortran prototype
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument

```fortran
recursive subroutine scal_cpu_func(buffers, cl_args) bind(c)
type(c_ptr), value, intent(in) :: buffers, cl_args

integer :: n
real(8), dimension(:), pointer :: val
real(8), target :: factor

n = fstarpu_vector_get_nx(buffers, 0)
call c_f_pointer(fstarpu_vector_get_ptr(buffers, 0), &
    val, shape=[n])
call fstarpu_unpack_arg(cl_args, (/ c_loc(factor) /))
...
```
Writing a Kernel Function in Fortran

- Every kernel function has the same Fortran prototype
- Get vector’s number of elements and base pointer
- Get the scaling factor as inline argument
- Compute the vector scaling

```
recursive subroutine scal_cpu_func(buffers, cl_args) bind(c)
type(c_ptr), value, intent(in) :: buffers, cl_args

integer :: n
real(8), dimension(:,), pointer :: val
real(8), target :: factor

n = fstarpu_vector_get_nx(buffers, 0)
call c_f_pointer(fstarpu_vector_get_ptr(buffers, 0), &
    val, shape=[n])
call fstarpu_unpack_arg(cl_args, (/ c_loc(factor) /))

do i = 1, n
    val(i) = val(i)*factor
end do
```
Registering an array to StarPU from Fortran

```fortran
real(8), dimension(:), allocatable, target :: array
allocate(array(NX))
array = (/ (1.0, i=1, NX) /)

type(c_ptr) :: vector_handle

! StarPU vector data registration
! --------------------------------
call fstarpu_vector_data_register(vector_handle, &
   0, c_loc(array), NX, c_sizeof(array(0)))
```
Submitting a task from Fortran

```fortran
real(kind=c_double), target :: factor

call fstarpu_task_insert(/ &
    scal_cl, &
    FSTARPU_RW &
    FSTARPU_VALUE, c_loc(factor) &
    FSTARPU_SZ_C_DOUBLE, &
    C_NULL_PTR /))
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
float matrix[NX*NY];
starpu_data_handle_t handle;

starpu_matrix_data_register(&handle, 0, (uintptr_t)matrix, NX, NX/10, NY/10, sizeof(matrix[0]));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```c
...  
starpu_data_handle_t handle;  
starpu_bcsr_data_register(&handle, 0, NNZ, NROW,  
    (uintptr_t) bcsr_matrix_data,  
    bcsr_matrix_indices, bcsr_matrix_rowptr,  
    first_entry,  
    BLOCK_NROW, BLOCK_NCOL, sizeof(double));
```
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix
- Extensible data type set
  - You can write your own, specifically tailored data type
Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix
- Extensible data type set
  - You can write your own, specifically tailored data type
- Only the byte size and the shape of data matter, not the actual element type (integer, float, double precision float, ...)
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

**Partition**

```c
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
  .filter_func = starpu_vector_filter_block,
  .nchildren = NB_PARTS
};
starpu_data_partition(handle, &filter);

/* Data can only be accessed through sub-handles now */
```
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

Partition $\rightarrow$ Use

```c
for (i=0; i<starpu_data_get_nb_children(handle); i++) {
    /* Get subdata number i */
    starpu_data_handle_t sub_handle =
        starpu_data_get_sub_data(handle, 1, i);

    starpu_task_insert(
        &scal_cl,
        STARPU_RW, sub_handle,
        STARPU_VALUE, &factor, sizeof(factor),
        0);
}
```
Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of filter

Partition → Use → Unpartition

```c
/* Wait for submitted tasks to complete */
starpu_task_wait_for_all();

/* Unpartition data */
starpu_data_unpartition(handle, 0);

/* Data can now be accessed through 'handle' only */
```
Hands-on session 1
Advanced principles of StarPU
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps
  - Partition planning

```c
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector, NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
  .filter_func = starpu_vector_filter_block,
  .nchildren = NB_PARTS
};
starpu_data_handle_t children[NB_PARTS];
starpu_data_partition_plan(handle, &filter, children);

/* Data can only be accessed through sub-handles now */
```
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning
- Asynchronous partition enforcement

```c
starpu_task_insert(&scal_cl,
    STARPU_RW, handle,
    STARPU_VALUE, &factor1, sizeof(factor1), 0);
starpu_data_partition_submit(handle, NB_PARTS, children);
for (i=0; i<NB_PARTS; i++) {
    starpu_task_insert(&scal_cl,
        STARPU_RW, children[i],
        STARPU_VALUE, &factor2, sizeof(factor2),
        0);
}
starpu_data_unpartition_submit(handle, NB_PARTS, children, node);
starpu_task_insert(&scal_cl,
    STARPU_RW, handle,
    STARPU_VALUE, &factor3, sizeof(factor3), 0);
```
Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning
- Asynchronous partition enforcement, or automatic!

```c
starpu_task_insert(&scal_cl,
    STARPU_RW, handle,
    STARPU_VALUE, &factor1, sizeof(factor1), 0);

// starpu_data_partition_submit(handle, NB_PARTS, children);
for (i=0; i<NB_PARTS; i++) {
    starpu_task_insert(&scal_cl,
        STARPU_RW, children[i],
        STARPU_VALUE, &factor2, sizeof(factor2),
        0);
}

// starpu_data_unpartition_submit(handle, NB_PARTS, children, node);

starpu_task_insert(&scal_cl,
    STARPU_RW, handle,
    STARPU_VALUE, &factor3, sizeof(factor3), 0);
```
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero

```c
void bzero_cpu(void *descr[], void *cl_arg) {
    double *v_zero = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
    *v_zero = 0.0;
}

struct starpu_codelet bzero_cl = {
    .cpu_funcs = { bzero_cpu },
    .nbuffers = 1
};
```
Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op

```c
void accumulate_cpu(void *descr[], void *cl_arg) {
    double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
    double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
    *v_dst = *v_dst + *v_src;
}

struct starpu_codelet accumulate_cl = {
    .cpu_funcs = { accumulate_cpu },
    .nbuffers = 1
};
```
**Reduction**

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op → **Reduce task contributions**

```c
starpu_variable_data_register(&accum_handle, -1,
    NULL, sizeof(type));
starpu_data_set_reduction_methods(accum_handle,
    &accumulate_cl, &bzero_cl);

for (b = 0; b < nblocks; b++)
    starpu_task_insert(&dot_kernel_cl,
        STARPU_REDUX, accum_handle,
        STARPU_R, starpu_data_get_sub_data(v1, 1, b),
        STARPU_R, starpu_data_get_sub_data(v2, 1, b),
        0);
```
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - Too strong for some kind of workloads
  - N-body, unstructured meshes
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - Too strong for some kind of workloads
  - N-body, unstructured meshes
Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - Too strong for some kind of workloads
  - N-body, unstructured meshes

- **Commutate**: allows a set of tasks to modify a buffer in any order

```c
starpu_task_insert(&cl1,
    STARPU_R, handle0,
    STARPU_RW, handle,
    0);

starpu_task_insert(&cl2,
    STARPU_R, handle1,
    STARPU_RW | STARPU_COMMUTE, handle,
    0);

starpu_task_insert(&cl2,
    STARPU_R, handle2,
    STARPU_RW | STARPU_COMMUTE, handle,
    0);

starpu_task_insert(&cl3,
    STARPU_R, handle3,
    STARPU_RW, handle,
    0);
```
Performance models

- Provide estimated task duration
- For advanced task scheduling: tells about the future!
- For task performance feedback
Performance models

- Provide estimated task duration
- For advanced task scheduling: tells about the future!
- For task performance feedback

```c
struct starpu_perfmodel scal_cl_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "scal",
};

struct starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func },
    ...
    .model = &scal_cl_model;
}
```
Performance models

- Provide estimated task duration
- For advanced task scheduling: tells about the future!
- For task performance feedback

```c
struct starpu_perfmodel scal_cl_model = {
    .type = STARPU_REGRESSION_BASED,
    .symbol = "scal",
};

struct starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func },
    ...
    .model = &scal_cl_model;
```


Performance models

- Provide estimated task duration
- For advanced task scheduling: tells about the future!
- For task performance feedback

```c
struct starpu_perfmodel scal_cl_model = {
    .type = STARPU_NL_REGRESSION_BASED,
    .symbol = "scal",
};
struct starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func },
    ...
    .model = &scal_cl_model;
}
Performance models

- Provide estimated task duration
- For advanced task scheduling: tells about the future!
- For task performance feedback

```c
struct starpu_perfmodel scal_cl_model = {
    .type = STARPU_MULTIPLE_REGRESSION_BASED,
    .symbol = "scal",
};
struct starpu_codelet scal_cl = {
    .cpu_funcs = { scal_cpu_func },
    ...
    .model = &scal_cl_model;
}
```
Feedback mechanisms

Online Tools
- Statistics
- Visual debugging

Offline Tools
- Performance models
- Trace-based analysis
Visual debugging: Temanejo

A debugger at the task level
- Visualize task graph
- Add breakpoints
- Execute task-by-task
- ...
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool `starpu_perfmodel_display`
- History-based models
- Regression-based models

```
$ starpu_perfmodel_display --s starpu_slu_lu_model_11

performance model for cpu0_parallel1_impl0

<table>
<thead>
<tr>
<th># hash</th>
<th>size</th>
<th>mean (us)</th>
<th>stddev (us)</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa6d4ef7</td>
<td>4194304</td>
<td>3.055501e+05</td>
<td>5.840822e+04</td>
<td>48</td>
</tr>
</tbody>
</table>
```
Offline Feedback – Kernel Model Characteristics

$ starpu_perfmodel_plot -s starpu_slu_lu_model_11
Offline Feedback – Kernel Model Regression Fitness

$ starpu_perfmodel_plot -s non_linear_memset_regression_based
Offline Feedback – Kernel Model Efficiency

$\texttt{starpu_perfmodel_plot -f -s chol\_model\_11}$

![Graph showing model for codelet chol_model_11.type](image)
Offline Feedback – Kernel Model Power Efficiency

`$ starpu_perfmodel_plot -f -se memset memset_energy`
Offline Feedback – Synthetic Kernels’ Behaviour

$ starpu_fxt_data_trace /tmp/prof_file_user_0
Offline Trace-Based Feedback

- FxT trace collection
- Trace analysis and display
  - ViTE Gantt
  - Graphviz DAG
  - StarVZ Gantt / R plots
Offline Feedback – Trace Analysis

Automatically generated

- Dependency graph (DAG)
- Activity diagramm (GANTT)
  - Visualize with ViTE

![ViTE visualization](image-url)
Offline Feedback – Trace Analysis
Distributed task graph

1D blocked stencil example, sequential version

```
for (loop = 0; loop < NLOOPS; loop++) {

    for (i = 0; i < n; i++)
        update(&blocks[i-1], /* R */
               &blocks[i],    /* RW */
               &blocks[i+1]); /* R */

}
```
Distributed task graph

1D blocked stencil example, MPI version

- Distribute for i iterations, receive/send contributions

```c
for (loop = 0; loop < NLOOPS; loop++) {
    int first = me * num_blocks_per_rank;
    int last = (me+1) * num_blocks_per_rank - 1;

    MPI_Recv(&blocks[first-1], me-1);
    MPI_Recv(&blocks[last+1], me+1);

    for (i = first; i <= last; i++)
        update(&blocks[i-1], /* R */
               &blocks[i], /* RW */
               &blocks[i+1]); /* R */

    MPI_Send(&blocks[first], me-1);
    MPI_Send(&blocks[last], me+1);
}
```
Distributed task graph

1D blocked stencil example, **task version, explicit**

- Submit send/recv and update tasks \(\rightarrow\) tasks+communications graph

```c
for (loop = 0; loop < NLOOPS; loop++) {  
    int first = me * num_blocks_per_rank ;  
    int last = (me+1) * num_blocks_per_rank - 1 ;

    starpu_mpi_irecv_submit ( handles [ first ] , me-1) ;  
    starpu_mpi_irecv_submit ( handles [ last +1] , me+1) ;

    for ( i = first ; i <= last ; i++)  
        starpu_task_insert (&cl_update , STARPU_R , handles [ i -1] ,  
                              STARPU_RW , handles [ i ] ,  
                              STARPU_R , handles [ i +1] ,0) ;

    starpu_mpi_isend_submit ( handles [ first ] , me-1) ;  
    starpu_mpi_isend_submit ( handles [ last ] , me+1) ;
}

starpu_task_wait_for_all () ;
```
Distributed task graph

1D blocked stencil example, task version, implicit

- Let communication tasks get inferred from data mapping

```c
for (loop = 0; loop < NLOOPS; loop++) {
    for (i = 0; i < n; i++)
        starpu_mpi_task_insert(&cl_update,
                                  STARPU_R, handles[i-1],
                                  STARPU_RW, handles[i],
                                  STARPU_R, handles[i+1],0);
}
starpu_task_wait_for_all();
```
Hands-on session 2