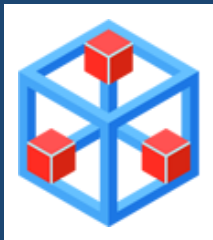


Inria
INVENTORS FOR THE DIGITAL WORLD



StarPU task-based programming hands-on session

Introduction to task-based programming with StarPU

STORM
STATIC OPTIMIZATIONS – RUNTIME METHODS

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

1

Introduction to task-based programming with StarPU

Basic Example: Scaling a Vector

```
1 float factor = 3.14;  
2 float vector[NX];  
3  
4  
5 /* ... fill vector ... */
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;  
2 float vector[NX];  
3 starpu_data_handle_t vector_handle;  
4  
5 /* ... fill vector ... */
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
8                             (uintptr_t)vector, NX, sizeof(vector[0]));
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
8                             (uintptr_t)vector, NX, sizeof(vector[0]));
9
10 starpu_task_insert(
11     &scal_cl,
12     STARPU_RW, vector_handle,
13     STARPU_VALUE, &factor, sizeof(factor),
14     0);
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
8                             (uintptr_t)vector, NX, sizeof(vector[0]));
9
10 starpu_task_insert(
11     &scal_cl,
12     STARPU_RW, vector_handle,
13     STARPU_VALUE, &factor, sizeof(factor),
14     0);
15
16 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
8                             (uintptr_t)vector, NX, sizeof(vector[0]));
9
10 starpu_task_insert(
11     &scal_cl,
12     STARPU_RW, vector_handle,
13     STARPU_VALUE, &factor, sizeof(factor),
14     0);
15
16 starpu_task_wait_for_all();
17 starpu_data_unregister(vector_handle);
18
19 /* ... 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**

Codelet
scal_cl



Definition: A Codelet

A Codelet...

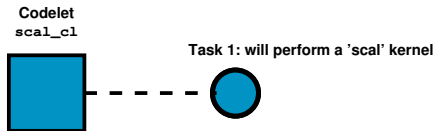
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **tasks**



Definition: A Codelet

A Codelet...

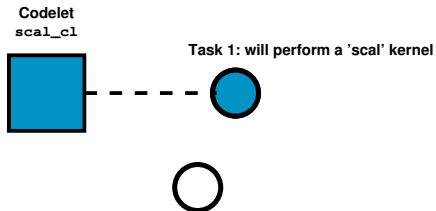
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **tasks**



Definition: A Codelet

A Codelet...

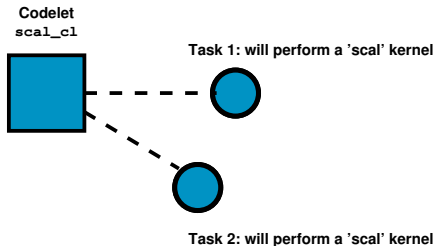
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **tasks**



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**



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...

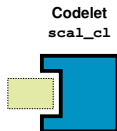
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A Task...

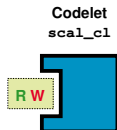
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A Task...

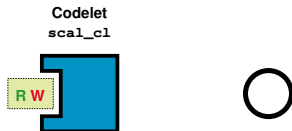
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A Task...

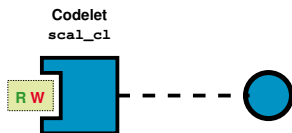
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



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...

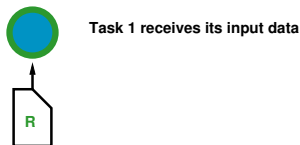
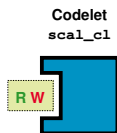
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A Task...

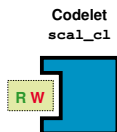
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A Task...

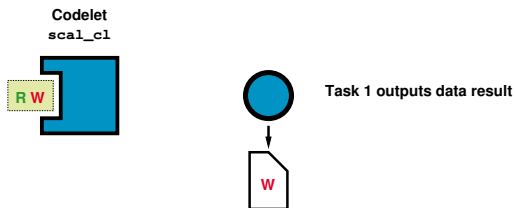
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



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`

```
1 struct starpu_codelet scal_cl = {  
2     ...  
3 };
```

Declaring a Codelet

Define a **struct** `starpu_codelet`

- Plug the kernel function
 - Here: `scal_cpu_func`

```
1 struct starpu_codelet scal_cl = {  
2     .cpu_func = { scal_cpu_func },  
3     ...  
4 };
```

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

```
1 struct starpu_codelet scal_cl = {  
2     .cpu_func = { scal_cpu_func },  
3     .nbuffers = 1,  
4     ...  
5 };
```

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

```
1 struct starpu_codelet scal_cl = {  
2     .cpu_func = { scal_cpu_func },  
3     .nbuffers = 1,  
4     .modes = { STARPU_RW },  
5 };
```

Declaring and Managing Data

Put data under StarPU control

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```
1 float vector[NX];  
2 /* ... 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

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
6                             (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

```
1 float vector[NX];  
2 /* ... fill data ... */  
3  
4 starpu_data_handle_t vector_handle;  
5 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,  
6                             (uintptr_t)vector, NX, sizeof(vector[0]));  
7  
8 /* ... 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**

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
6                             (uintptr_t)vector, NX, sizeof(vector[0]));
7
8 /* ... use the vector through the handle ... */
9
10 starpu_data_unregister(vector_handle);
```

Writing a Kernel Function

- Every kernel function has the same C prototype

```
1 void scal_cpu_func(void *buffers [], void *cl_arg) {  
2     ...  
3 }
```

Writing a Kernel Function

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

```
1 void scal_cpu_func(void *buffers [], void *cl_arg) {  
2     struct starpu_vector_interface *vector_handle = buffers[0];  
3  
4     ...  
5 }
```

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

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {  
2     struct starpu_vector_interface *vector_handle = buffers[0];  
3  
4     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);  
5     float *vector  = STARPU_VECTOR_GET_PTR(vector_handle);  
6  
7     ...  
8 }
```

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

```
1 void scal_cpu_func(void *buffers [], void *cl_arg) {  
2     struct starpu_vector_interface *vector_handle = buffers [0];  
3  
4     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);  
5     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);  
6  
7     float factor;  
8     starpu_codelet_unpack_args(cl_arg , &factor)  
9  
10    ...  
11 }
```


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**

```
1 void scal_cpu_func(void *buffers [], void *cl_arg) {
2     struct starpu_vector_interface *vector_handle = buffers[0];
3
4     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
5     float *vector   = STARPU_VECTOR_GET_PTR(vector_handle);
6
7     float factor;
8     starpu_codelet_unpack_args(cl_arg, &factor)
9
10    unsigned i;
11    for (i = 0; i < n; i++)
12        vector[i] *= factor;
13 }
```

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

```
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

```
1 starpu_task_insert(&scal_cl ,  
2                   STARPU_RW , vector_handle ,  
3                   ... ) ;
```

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

```
1 starpu_task_insert(&scal_cl ,  
2                   STARPU_RW , vector_handle ,  
3                   STARPU_VALUE , &factor , sizeof(factor) ,  
4                   ... ) ;
```

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

```
1 starpu_task_insert(&scal_cl ,  
2                   STARPU_RW , vector_handle ,  
3                   STARPU_VALUE , &factor , sizeof(factor) ,  
4                   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-blockingly**

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

```
1 /* non-blocking task submits */  
2 starpu_task_insert (...);  
3 starpu_task_insert (...);  
4 starpu_task_insert (...);  
5 ...
```

Waiting for Submitted Task Completion

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

```
1 /* non-blocking task submits */  
2 starpu_task_insert (...);  
3 starpu_task_insert (...);  
4 starpu_task_insert (...);  
5 ...
```

Waiting for Submitted Task Completion

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

```
1 /* non-blocking task submits */
2 starpu_task_insert (...);
3 starpu_task_insert (...);
4 starpu_task_insert (...);
5 ...
6
7 /* wait for all task submitted so far */
8 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
4
5 /* ... fill vector ... */
6
7 starpu_vector_data_register(&vector_handle, STARPU_MAIN_RAM,
8                             (uintptr_t)vector, NX, sizeof(vector[0]))
9                             ;
10
11 starpu_task_insert(
12     &scal_cl,
13     STARPU_RW, vector_handle,
14     STARPU_VALUE, &factor, sizeof(factor),
15     0);
16
17 starpu_task_wait_for_all();
18 starpu_data_unregister(vector_handle);
19
20 /* ... 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

```
1 struct starpu_codelet scal_cl = {  
2     .cpu_func = { scal_cpu_func ,  
3                 scal_sse_cpu_func , scal_avx_cpu_func },  
4     .nbuffers = 1,  
5     .modes = { STARPU_RW },  
6 };
```

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

```
1 struct starpu_codelet scal_cl = {
2     .cpu_func = { scal_cpu_func ,
3                 scal_sse_cpu_func , scal_avx_cpu_func },
4     .opencl_func = { scal_cpu_opencl },
5     .cuda_func = { scal_cpu_cuda },
6     .nbuffers = 1,
7     .modes = { STARPU_RW },
8 };
```

Writing a Kernel Function for **CUDA**

Writing a Kernel Function for **CUDA**

```
1
2
3
4
5
6
7
8 extern "C" void scal_cuda_func(void *buffers [], void *cl_arg)
9     {
10    struct starpu_vector_interface *vector_handle = buffers [0];
11    unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
12    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
13    float factor;
14    starpu_codelet_unpack_args(cl_arg , &factor)
15
16    ...
17
18
19
20
21
22 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8 extern "C" void scal_cuda_func(void *buffers [], void *cl_arg)
9 {
10     struct starpu_vector_interface *vector_handle = buffers[0];
11     unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
12     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
13     float factor;
14     starpu_codelet_unpack_args(cl_arg, &factor)
15
16     unsigned threads_per_block = 64;
17     unsigned nblocks = (n+threads_per_block-1)
18                       /threads_per_block;
19     ...
20
21
22 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8 extern "C" void scal_cuda_func(void *buffers [], void *cl_arg)
9 {
10     struct starpu_vector_interface *vector_handle = buffers [0];
11     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
12     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
13     float factor;
14     starpu_codelet_unpack_args(cl_arg , &factor)
15
16     unsigned threads_per_block = 64;
17     unsigned nblocks = (n+threads_per_block-1)
18                       /threads_per_block;
19
20     vector_mult_cuda<<<<nblocks , threads_per_block , 0 ,
21                       starpu_cuda_get_local_stream ()>>>(n, vector , factor);
22     cudaStreamSynchronize(starpu_cuda_get_local_stream ());
23 }
```

Writing a Kernel Function for CUDA

```
1 static __global__ void vector_mult_cuda(unsigned n,  
2                                         float *vector, float factor){  
3     unsigned i = blockIdx.x*blockDim.x + threadIdx.x;  
4  
5     ...  
6 }  
7  
8 extern "C" void scal_cuda_func(void *buffers [], void *cl_arg)  
9 {  
10    struct starpu_vector_interface *vector_handle = buffers[0];  
11    unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);  
12    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);  
13    float factor;  
14    starpu_codelet_unpack_args(cl_arg, &factor)  
15  
16    unsigned threads_per_block = 64;  
17    unsigned nblocks = (n+threads_per_block-1)  
18                    /threads_per_block;  
19  
20    vector_mult_cuda<<<nblocks, threads_per_block, 0,  
21                    starpu_cuda_get_local_stream()>>>(n, vector, factor);  
22    cudaStreamSynchronize(starpu_cuda_get_local_stream());  
23 }
```

Writing a Kernel Function for CUDA

```
1 static __global__ void vector_mult_cuda(unsigned n,
2                                         float *vector, float factor){
3     unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
4     if (i < n)
5         vector[i] *= factor;
6 }
7
8 extern "C" void scal_cuda_func(void *buffers [], void *cl_arg)
9 {
10    struct starpu_vector_interface *vector_handle = buffers[0];
11    unsigned n = STARPU_VECTOR_GET_NX(vector_handle);
12    float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
13    float factor;
14    starpu_codelet_unpack_args(cl_arg, &factor)
15
16    unsigned threads_per_block = 64;
17    unsigned nblocks = (n+threads_per_block-1)
18                      /threads_per_block;
19
20    vector_mult_cuda<<<nblocks, threads_per_block, 0,
21                    starpu_cuda_get_local_stream()>>>(n, vector, factor);
22    cudaStreamSynchronize(starpu_cuda_get_local_stream());
23 }
```


Writing a Kernel Function for OpenCL

```
1  __kernel void vector_mult_opencl(__global float *val ,
2                                  unsigned n, float factor) {
3      for (unsigned i = 0; i < n; i++)
4          val[i] *= factor;
5  }
6
7
8  extern "C" void scal_opencl_func(void *buffers [], void *cl_arg
9                                  ){
10     struct starpu_vector_interface *vector_handle = buffers[0];
11     unsigned n      = STARPU_VECTOR_GET_NX(vector_handle);
12     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
13     float factor;
14     starpu_codelet_unpack_args(cl_arg , &factor)
15
16     cl_kernel kernel;
17     cl_command_queue queue;
18     starpu_opencl_load_kernel(&kernel , &queue , &opencl_program ,
19                             name, "vector_mult_opencl" , devid);
20     clSetKernelArg(kernel , 0, sizeof(val) , &val);
21     ...
22     clEnqueueNDRangeKernel(queue , kernel , 1, NULL, ...);
23 }
```

Declaring a codelet in **Fortran**

- Defined as a C_PTR allocated by StarPU

```
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

```
1 TYPE(C_PTR) :: scal_cl = C_NULL_PTR
2
3 scal_cl = fstarpu_codelet_allocate()
4
5 CALL fstarpu_codelet_add_cpu_func(scal_cl ,
6   C_FUNLOC(scal_cpu_func))
```

Declaring a codelet in Fortran

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

```
1 TYPE(C_PTR) :: scal_cl = C_NULL_PTR
2
3 scal_cl = fstarpu_codelet_allocate()
4
5 CALL fstarpu_codelet_add_cpu_func(scal_cl ,
6     C_FUNLOC(scal_cpu_func))
7
8 CALL fstarpu_codelet_add_buff(scal_cl , FSTARPU_RW)
```

Writing a Kernel Function in Fortran

- Every kernel function has the same Fortran prototype

```
1 recursive subroutine scal_cpu_func( buffers , cl_args ) bind(c)  
2   type(c_ptr), value , intent(in) :: buffers , cl_args  
3  
4   ...
```

Writing a Kernel Function in Fortran

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

```
1 recursive subroutine scal_cpu_func( buffers , cl_args ) bind(c)
2   type(c_ptr), value , intent(in) :: buffers , cl_args
3
4   integer :: n
5   real(8) , dimension(:) , pointer :: val
6
7
8   n = fstarpu_vector_get_nx( buffers , 0)
9   call c_f_pointer( fstarpu_vector_get_ptr( buffers , 0) , &
10    val , shape=[n])
11
12   ...
```

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

```
1 recursive subroutine scal_cpu_func( buffers , cl_args ) bind(c)
2   type(c_ptr), value , intent(in) :: buffers , cl_args
3
4   integer :: n
5   real(8) , dimension(:) , pointer :: val
6   real(8) , target :: factor
7
8   n = fstarpu_vector_get_nx( buffers , 0)
9   call c_f_pointer( fstarpu_vector_get_ptr( buffers , 0) , &
10    val , shape=[n])
11   call fstarpu_unpack_arg( cl_args , (/ c_loc( factor) /))
12
13   ...
```

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

```
1 recursive subroutine scal_cpu_func( buffers , cl_args ) bind(c)
2   type(c_ptr), value , intent(in) :: buffers , cl_args
3
4   integer :: n
5   real(8) , dimension(:) , pointer :: val
6   real(8) , target :: factor
7
8   n = fstarpu_vector_get_nx( buffers , 0)
9   call c_f_pointer( fstarpu_vector_get_ptr( buffers , 0) , &
10    val , shape=[n] )
11   call fstarpu_unpack_arg( cl_args , (/ c_loc( factor ) /))
12
13   do i = 1 , n
14     val(i) = val(i)*factor
15   end do
```


Registering an array to StarPU from Fortran

```
1  real(8), dimension (:), allocatable, target :: array
2  allocate(array(NX))
3  array = (/ (1.0, i=1, NX) /)
4
5  type(c_ptr) :: vector_handle
6  call fstarpu_vector_data_register(vector_handle, &
7  STARPU_MAIN_RAM, c_loc(array), NX, c_sizeof(array(0)))
```

Submitting a task from Fortran

```
1  real(kind=c_double), target :: factor
2
3  call fstarpu_task_insert((/ scal_cl , &
4     FSTARPU_RW, vector_handle , &
5     FSTARPU_VALUE, c_loc(factor), FSTARPU_SZ_C_DOUBLE, &
6     C_NULL_PTR /))
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```
1 int vector [NX];  
2 starpu_data_handle_t handle ;  
3  
4 starpu_vector_data_register(&handle , STARPU_MAIN_RAM,  
5                             (uintptr_t)vector ,  
6                             NX, sizeof(vector [0]));
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```
1 float matrix[NX*NY];  
2 starpu_data_handle_t handle;  
3  
4 starpu_matrix_data_register(&handle, STARPU_MAIN_RAM,  
5                             (uintptr_t)matrix,  
6                             NX, NX/10, NY/10,  
7                             sizeof(matrix[0]));
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- **BCSR sparse matrix**

```
1  ...
2  starpu_data_handle_t handle ;
3
4  starpu_bcsr_data_register(&handle , STARPU_MAIN_RAM,
5                          NNZ, NROW,
6                          (uintptr_t)bcsr_matrix_data ,
7                          bcsr_matrix_indices , bcsr_matrix_rowptr ,
8                          first_entry ,
9                          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

```
1 int vector[NX];
2 starpu_data_handle_t handle;
3 starpu_vector_data_register(&handle, STARPU_MAIN_RAM,
4                             (uintptr_t)vector,
5                             NX, sizeof(vector[0]));
6
7 /* Partition the vector in NB_PARTS sub-vectors */
8 struct starpu_data_filter filter = {
9     .filter_func = starpu_vector_filter_block,
10    .nchildren = NB_PARTS
11 };
12 starpu_data_partition(handle, &filter);
13
14 /* 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 → Use

```
1 for (i=0; i<starpu_data_get_nb_children(handle); i++) {
2     /* Get subdata number i */
3     starpu_data_handle_t sub_handle =
4         starpu_data_get_sub_data(handle, 1, i);
5
6     starpu_task_insert(
7         &scal_cl,
8         STARPU_RW, sub_handle,
9         STARPU_VALUE, &factor, sizeof(factor),
10        0);
11 }
```

Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of **filter**

Partition → Use → **Unpartition**

```
1 /* Wait for submitted tasks to complete */
2 starpu_task_wait_for_all();
3
4 /* Unpartition data */
5 starpu_data_unpartition(handle, 0);
6
7 /* Data can now be accessed through 'handle' only */
```

2

Hands-on session 1

3

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

```
1 int vector[NX];
2 starpu_data_handle_t handle;
3 starpu_vector_data_register(&handle, STARPU_MAIN_RAM,
4                             (uintptr_t)vector,
5                             NX, sizeof(vector[0]));
6
7 /* Partition the vector in NB_PARTS sub-vectors */
8 struct starpu_data_filter filter = {
9     .filter_func = starpu_vector_filter_block,
10    .nchildren = NB_PARTS
11 };
12 starpu_data_handle_t children[NB_PARTS];
13 starpu_data_partition_plan(handle, &filter, children);
```

Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning
- Asynchronous partition enforcement

```
1  starpu_task_insert(&scal_cl ,
2      STARPU_RW , handle ,
3      STARPU_VALUE , &factor1 , sizeof(factor1), 0);
4  starpu_data_partition_submit(handle, NB_PARTS, children);
5  for (i=0; i<NB_PARTS; i++) {
6      starpu_task_insert(&scal_cl ,
7          STARPU_RW , children[i],
8          STARPU_VALUE , &factor2 , sizeof(factor2),
9          0);
10 }
11 starpu_data_unpartition_submit(handle , NB_PARTS, children ,
12     node);
13 starpu_task_insert(&scal_cl ,
14     STARPU_RW , handle ,
15     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!** Can access any data.

```
1 starpu_task_insert(&scal_cl ,
2     STARPU_RW , handle ,
3     STARPU_VALUE , &factor1 , sizeof(factor1), 0);
4 //starpu_data_partition_submit(handle , NB_PARTS, children);
5 for (i=0; i<NB_PARTS; i++) {
6     starpu_task_insert(&scal_cl ,
7         STARPU_RW , children[i],
8         STARPU_VALUE , &factor2 , sizeof(factor2),
9         0);
10 }
11 //starpu_data_unpartition_submit(handle , NB_PARTS, children ,
12     node);
13 starpu_task_insert(&scal_cl ,
14     STARPU_RW , handle ,
15     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

```
1 void bzero_cpu(void *descr [], void *cl_arg) {
2     double *v_zero = (double *)STARPU_VARIABLE_GET_PTR(descr
3         [0]);
4     *v_zero = 0.0;
5 }
6 struct starpu_codelet bzero_cl = {
7     .cpu_funcs = { bzero_cpu },
8     .nbuffers = 1
9 };
```

Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op

```
1 void accumulate_cpu(void *descr [], void *cl_arg) {
2     double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr
3         [0]);
4     double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr
5         [1]);
6     *v_dst = *v_dst + *v_src;
7 }
8
9 struct starpu_codelet accumulate_cl = {
10     .cpu_funcs = { accumulate_cpu },
11     .nbuffers = 1
12 };
```

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

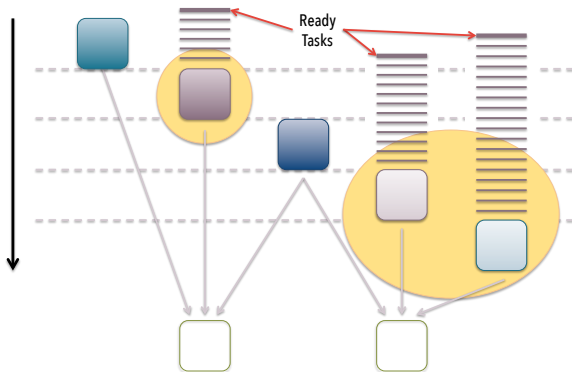
```
1 starpu_variable_data_register(&accum_handle, -1,
2                               NULL, sizeof(type));
3 starpu_data_set_reduction_methods(accum_handle,
4                                   &accumulate_cl, &bzero_cl);
5
6 for (b = 0; b < nblocks; b++)
7     starpu_task_insert(&dot_kernel_cl,
8                       STARPU_REDUX, accum_handle,
9                       STARPU_R, starpu_data_get_sub_data(v1, 1, b),
10                      STARPU_R, starpu_data_get_sub_data(v2, 1, b),
11                      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
- **Commute**: allows a set of tasks to modify a buffer in any order

```
1 starpu_task_insert(&c11 ,
2     STARPU_R , handle0 ,
3     STARPU_RW , handle ,
4     0);
5 starpu_task_insert(&c12 ,
6     STARPU_R , handle1 ,
7     STARPU_RW | STARPU_COMMUTE , handle ,
8     0);
9 starpu_task_insert(&c12 ,
10    STARPU_R , handle2 ,
11    STARPU_RW | STARPU_COMMUTE , handle ,
12    0);
13 starpu_task_insert(&c13 ,
14    STARPU_R , handle3 ,
15    STARPU_RW , handle ,
16    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

```
1 struct starpu_perfmodel scal_cl_model = {
2     .type = STARPU_HISTORY_BASED,
3     .symbol = "scal",
4 };
5 struct starpu_codelet scal_cl = {
6     .cpu_funcs = { scal_cpu_func },
7     ...
8     .model = &scal_cl_model;
9 }
```

Performance models

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

```
1 struct starpu_perfmodel scal_cl_model = {
2     .type = STARPU_REGRESSION_BASED,
3     .symbol = "scal",
4 };
5 struct starpu_codelet scal_cl = {
6     .cpu_funcs = { scal_cpu_func },
7     ...
8     .model = &scal_cl_model;
9 }
```

Performance models

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

```
1 struct starpu_perfmodel scal_cl_model = {
2     .type = STARPU_NL_REGRESSION_BASED,
3     .symbol = "scal",
4 };
5 struct starpu_codelet scal_cl = {
6     .cpu_funcs = { scal_cpu_func },
7     ...
8     .model = &scal_cl_model;
9 }
```

Performance models

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

```
1 struct starpu_perfmodel scal_cl_model = {
2     .type = STARPU_MULTIPLE_REGRESSION_BASED,
3     .symbol = "scal",
4 };
5 struct starpu_codelet scal_cl = {
6     .cpu_funcs = { scal_cpu_func },
7     ...
8     .model = &scal_cl_model;
9 }
```

Feedback mechanisms

Online Tools

- Statistics
- Visual debugging

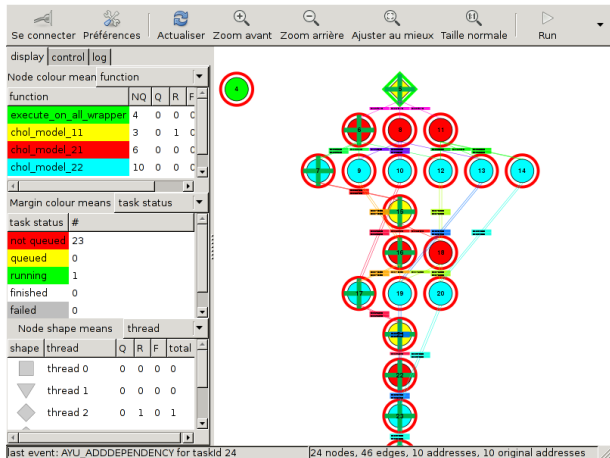
Offline Tools

- 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

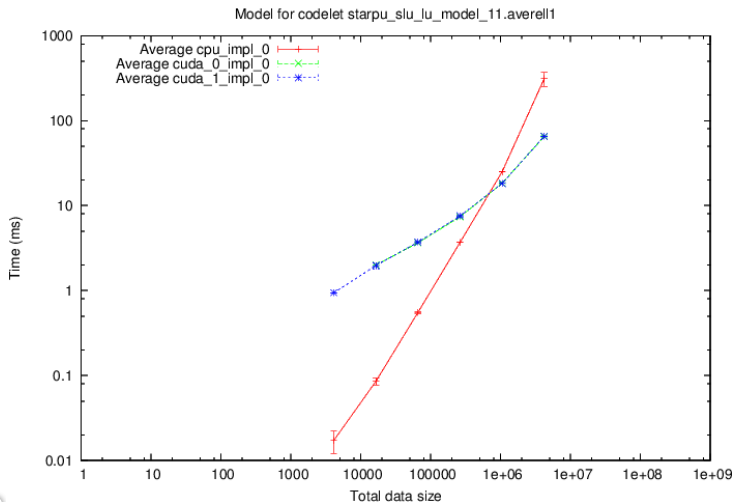
Display the codelet performance models recorded by StarPU

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

```
1 $ starpu_perfmodel_display -s starpu_slv_lu_model_11
2
3 performance model for cpu0_parallel1_impl0
4 # hash      size      mean (us)      stddev (us)      n
5 aa6d4ef7    4194304    3.055501e+05   5.804822e+04     48
```

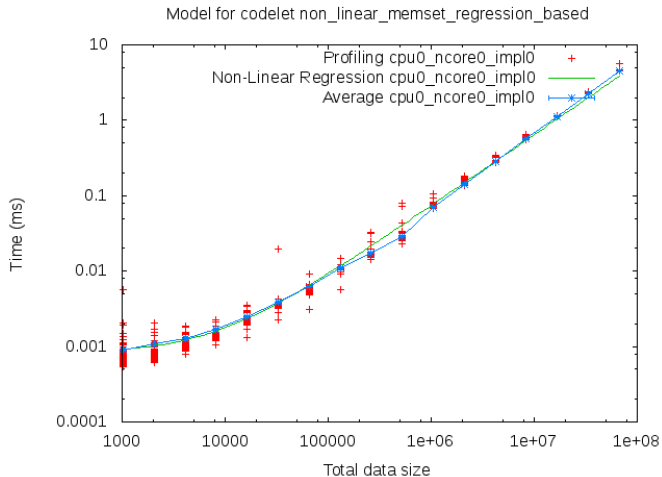
Offline Feedback – Kernel Model Characteristics

```
$ starpu_perfmodel_plot -s starpu_slu_lu_model_11
```



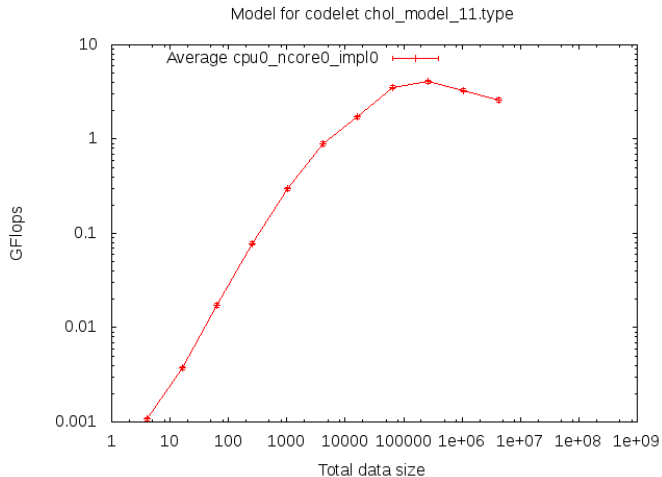
Offline Feedback – Kernel Model Regression Fitness

```
$ starpu_perfmodel_plot -s non_linear_memset_regression_based
```



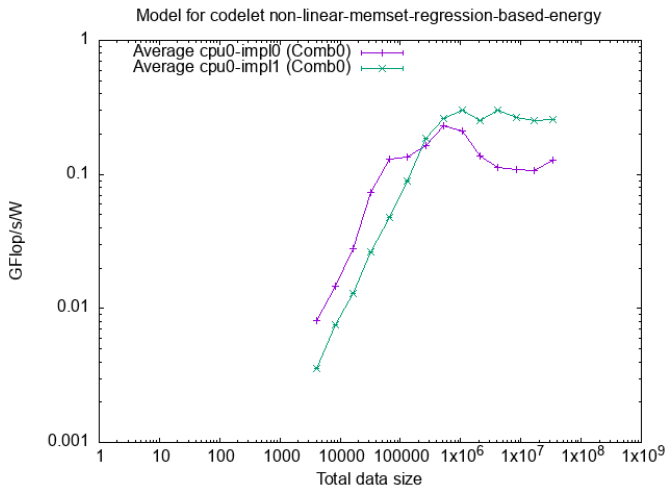
Offline Feedback – Kernel Model Efficiency

```
$ starpu_perfmodel_plot -f -s chol_model_11
```



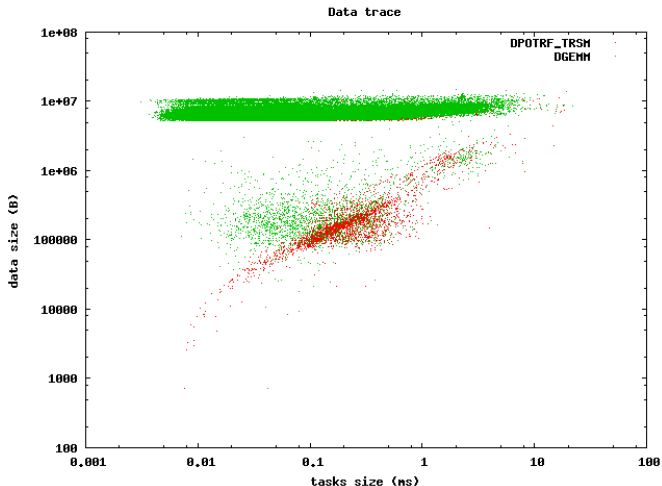
Offline Feedback – Kernel Model Power Efficiency

```
$ starpu_perfmodel_plot -f -se memset memset_energy
```



Offline Feedback – Synthetic Kernels' Behaviour

```
$ starpu_fxt_data_trace /tmp/prof_file_user_0
```



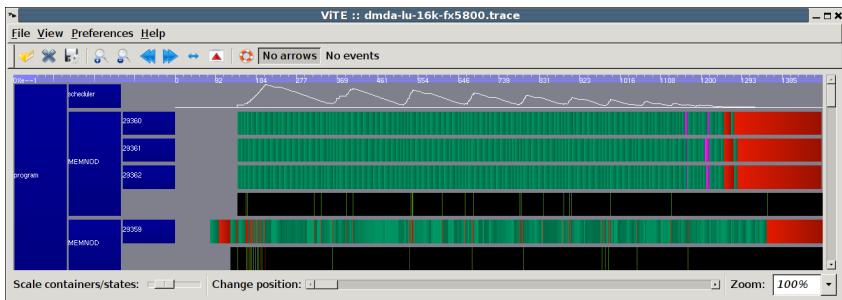
Offline Trace-Based Feedback

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

Offline Feedback – Trace Analysis

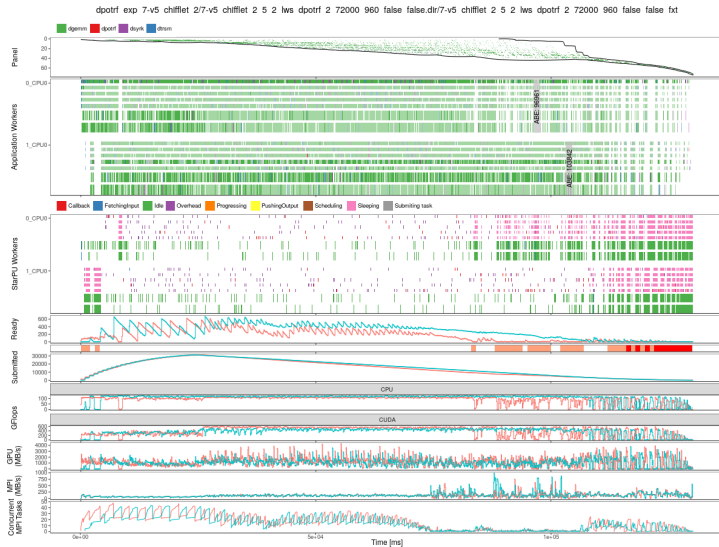
Automatically generated

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



Offline Feedback – Trace Analysis

StarVZ



Distributed task graph

1D blocked stencil example, sequential version

```
1  for (loop = 0; loop < NLOOPS; loop++) {  
2  
3  
4  
5  
6  
7  
8      for (i = 0; i < n; i++)  
9          update(&blocks[i-1], /* R */  
10                &blocks[i], /* RW */  
11                &blocks[i+1]); /* R */  
12  
13  
14 }
```

Distributed task graph

1D blocked stencil example, MPI version

- Distribute for i iterations, receive/send contributions

```
1 for (loop = 0; loop < NLOOPS; loop++) {
2   int first = me * num_blocks_per_rank;
3   int last = (me+1) * num_blocks_per_rank - 1;
4
5   MPI_Recv(&blocks[first-1], me-1);
6   MPI_Recv(&blocks[last+1], me+1);
7
8   for (i = first; i <= last; i++)
9     update(&blocks[i-1], /* R */
10           &blocks[i], /* RW */
11           &blocks[i+1]); /* R */
12
13   MPI_Send(&blocks[first], me-1);
14   MPI_Send(&blocks[last], me+1);
15 }
```

Distributed task graph

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

- Submit send/rcv and update tasks → tasks+communications graph

```
1 for (loop = 0; loop < NLOOPS; loop++) {
2     int first = me * num_blocks_per_rank;
3     int last = (me+1) * num_blocks_per_rank - 1;
4
5     starpu_mpi_irecv_submit(handles[first-1], me-1);
6     starpu_mpi_irecv_submit(handles[last+1], me+1);
7
8     for (i = first; i <= last; i++)
9         starpu_task_insert(&cl_update, STARPU_R, handles[i-1],
10                          STARPU_RW, handles[i],
11                          STARPU_R, handles[i+1], 0);
12
13     starpu_mpi_isend_submit(handles[first], me-1);
14     starpu_mpi_isend_submit(handles[last], me+1);
15 }
16 starpu_task_wait_for_all();
```

Distributed task graph

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

- Let communication tasks get inferred from data mapping

```
1 for (loop = 0; loop < NLOOPS; loop++) {
2
3
4
5
6
7
8     for (i = 0; i < n; i++)
9         starpu_mpi_task_insert(&cl_update ,
10                                STARPU_R,  handles[i-1],
11                                STARPU_RW, handles[i],
12                                STARPU_R,  handles[i+1],0);
13
14 }
15
16 starpu_task_wait_for_all();
```

4

Hands-on session 2