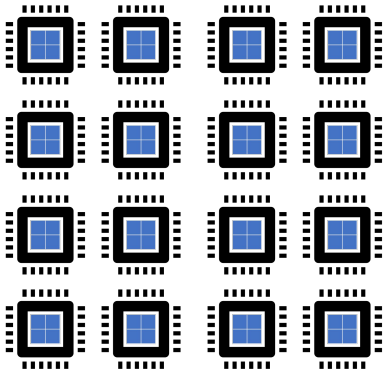
GPUs
bottlenecks in accelerated codes

l.bellentani@cineca.it
Tutorial on Profiling @ MHPC - ICTP

A new paradigm in parallel programming

A new paradigm in parallel programming

90s advent of MPI → distributed programming



Increasing density and frequency
of conventional CPU architecture

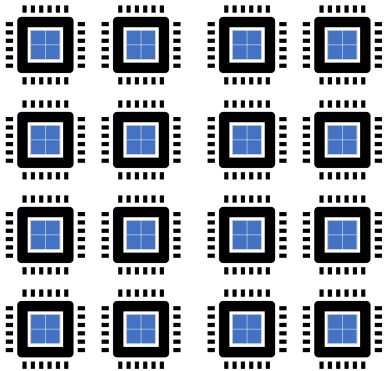
A new paradigm in parallel programming

2008

PETASCALE

Roadrunner supercomputer
Los Alamos National Laboratory (US)

10^{15} FLOP/s



Increasing density and frequency
of conventional CPU architecture

A new paradigm in parallel programming

2008

PETASCALE

Roadrunner supercomputer
Los Alamos National Laboratory (US)

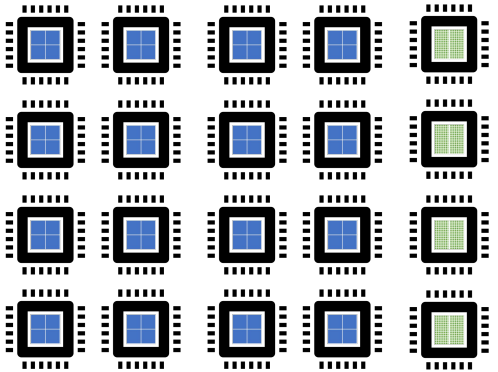
10^{15} FLOP/s

2022

EXASCALE

Frontier supercomputer
Oak Ridge National Laboratory (US)

10^{18} FLOP/s

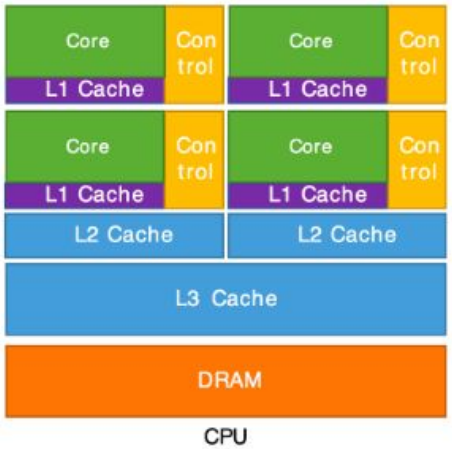


New hardware!
Graphic Processing Units (GPUs)

GPU vs CPU architecture

several strong cores

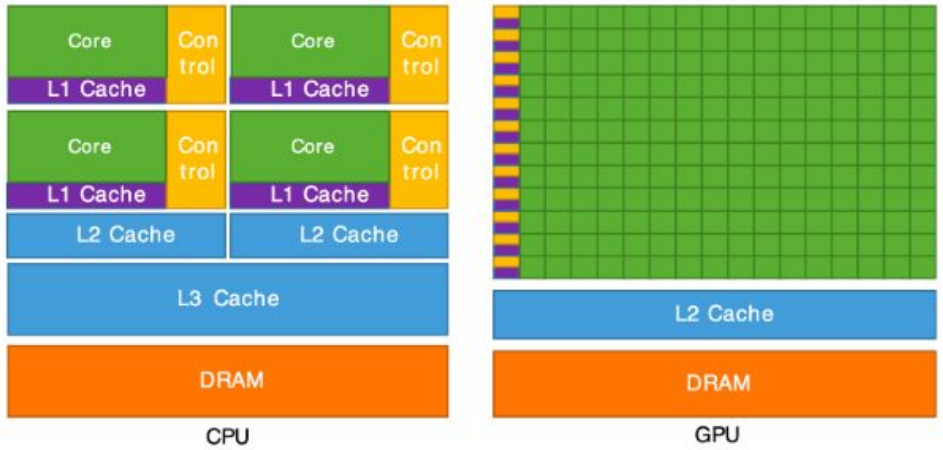
specialized processing



GPU vs CPU architecture

several strong cores

specialized processing



thousands of cores

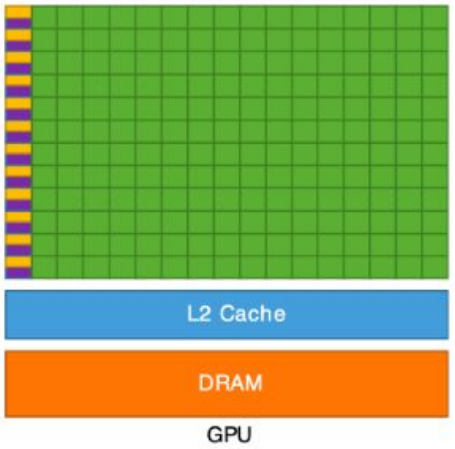
embarrassingly parallel processing

GPU vs CPU architecture

```
for(int i = 0; i < N; i++){
    a[i] = 2 * a[i];
```

a[0]	a[1]	a[2]	a[3]	...	a[N]
↑	↑	↑	↑	↑	↑
th0	th1	th2	th3	...	thN

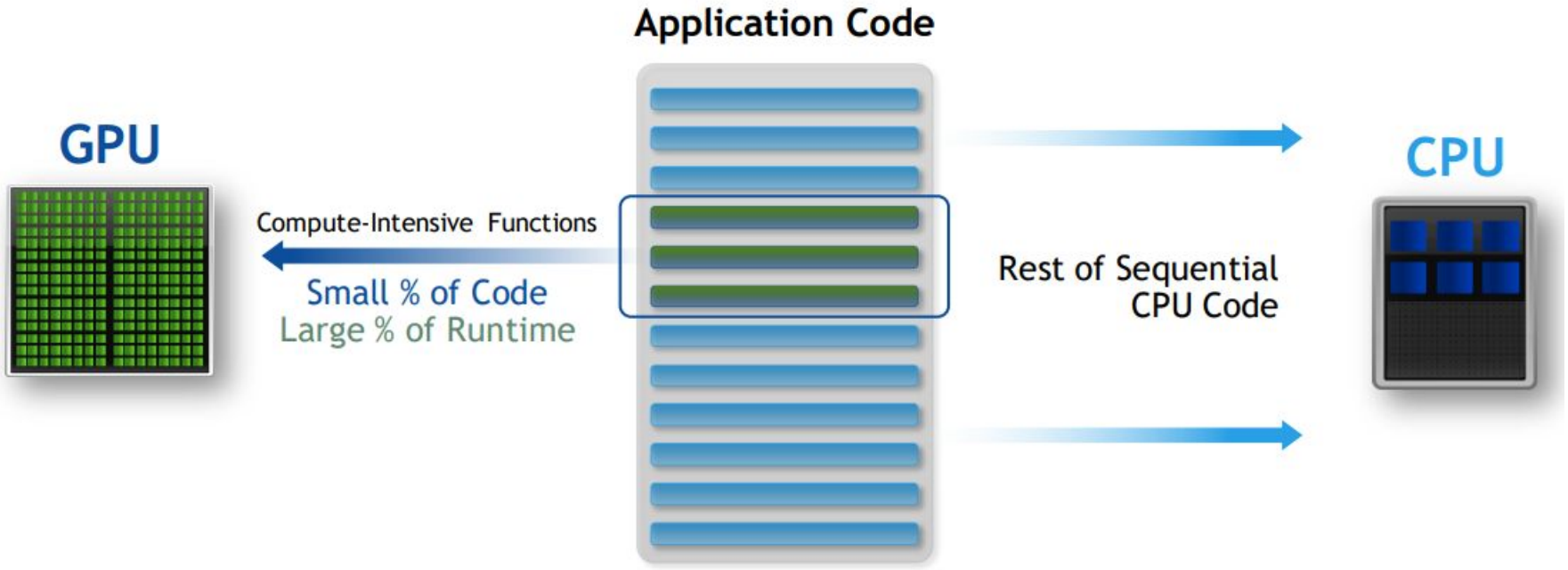
supports very large amount of parallelism



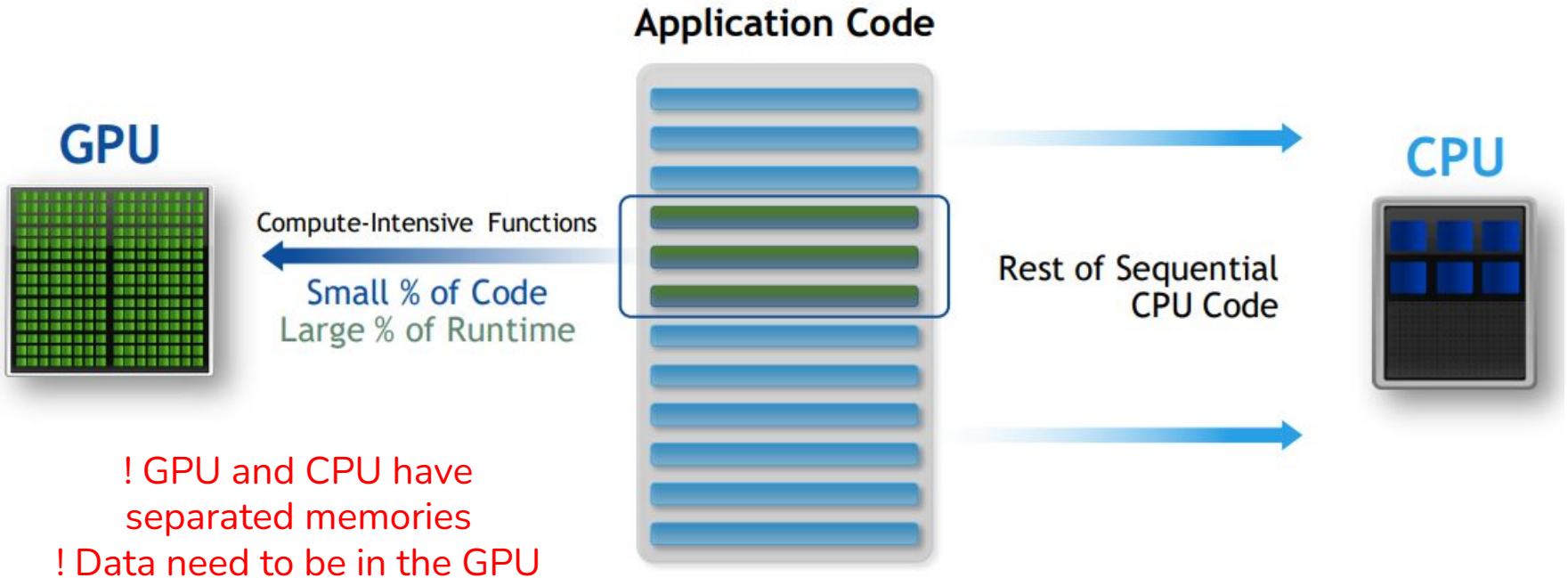
thousands of cores

embarrassingly parallel processing

Heterogeneous programming



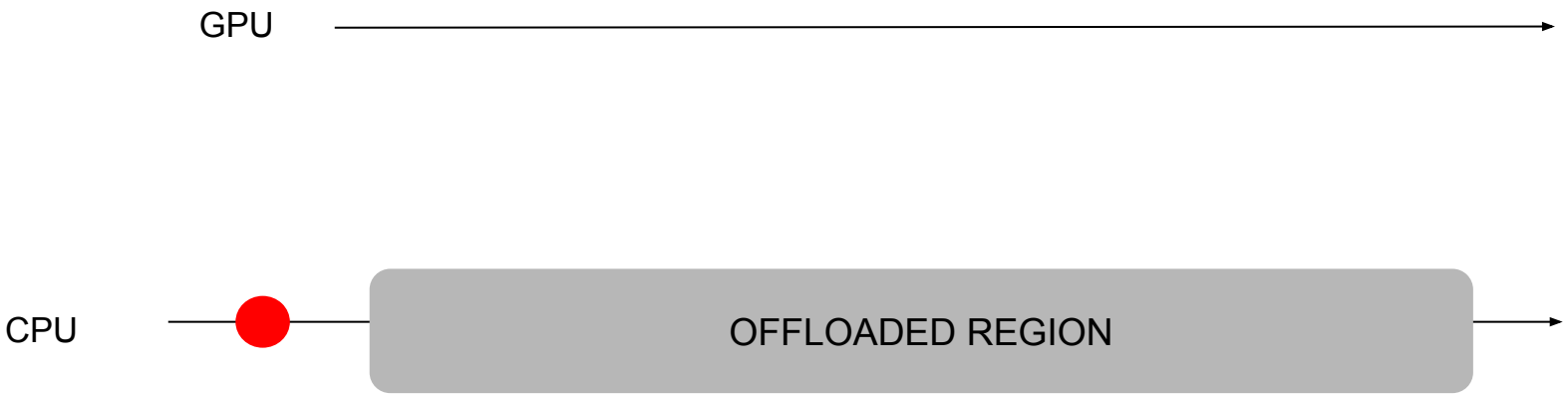
Heterogeneous programming



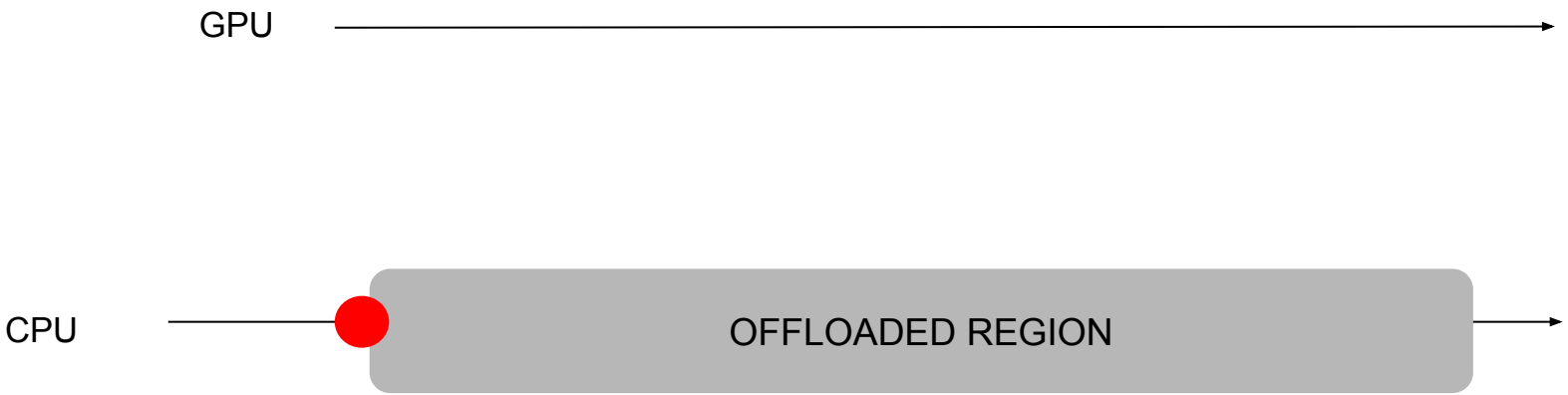
! GPU and CPU have separated memories

! Data need to be in the GPU memory

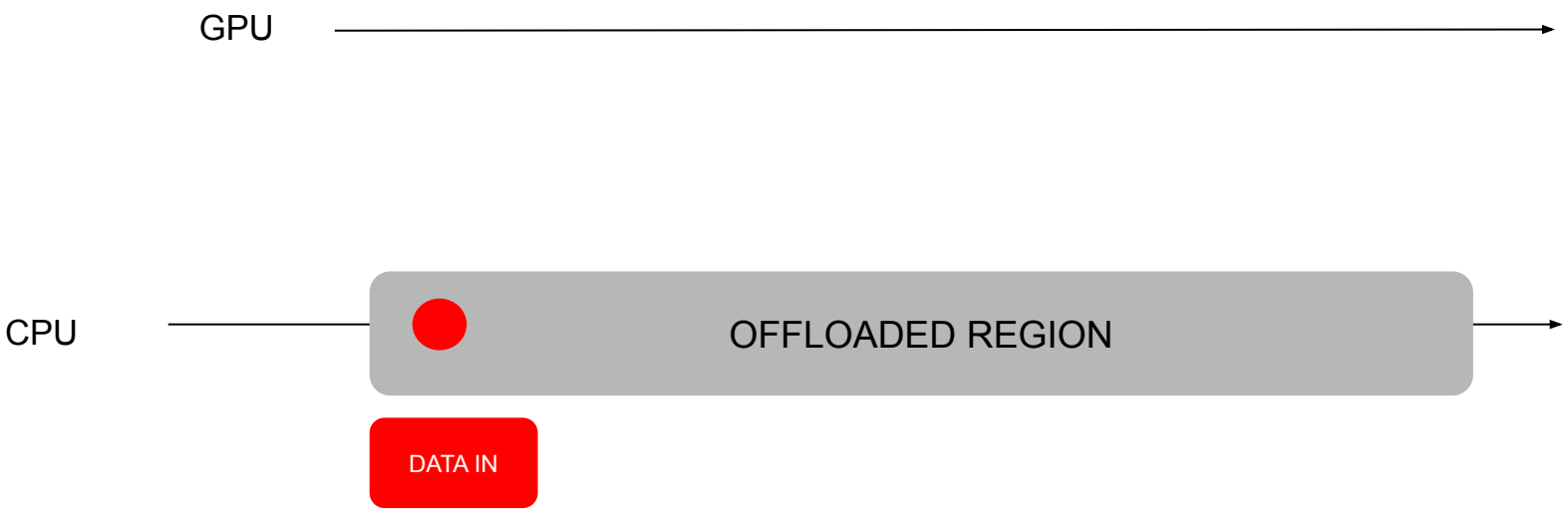
Traces in heterogeneous programs



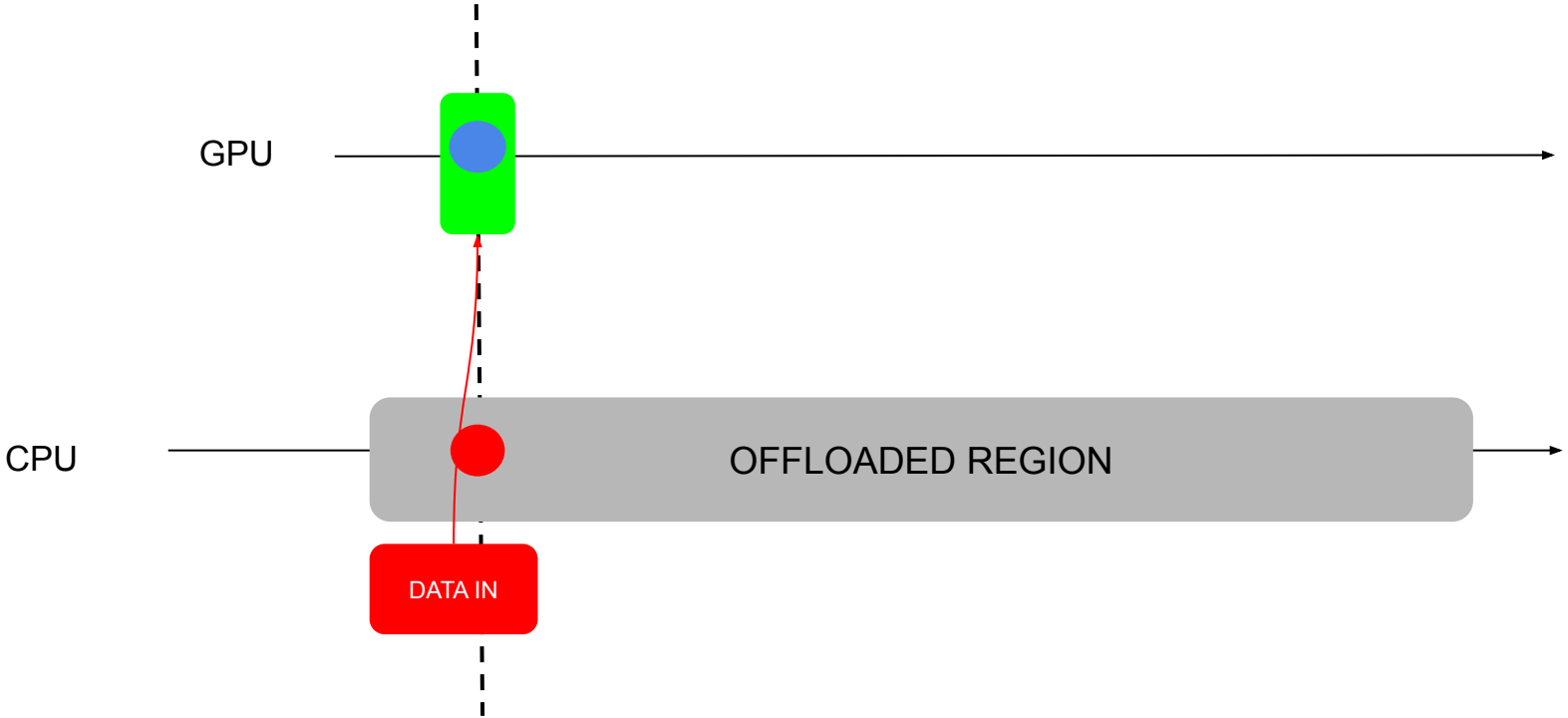
Traces in heterogeneous programs



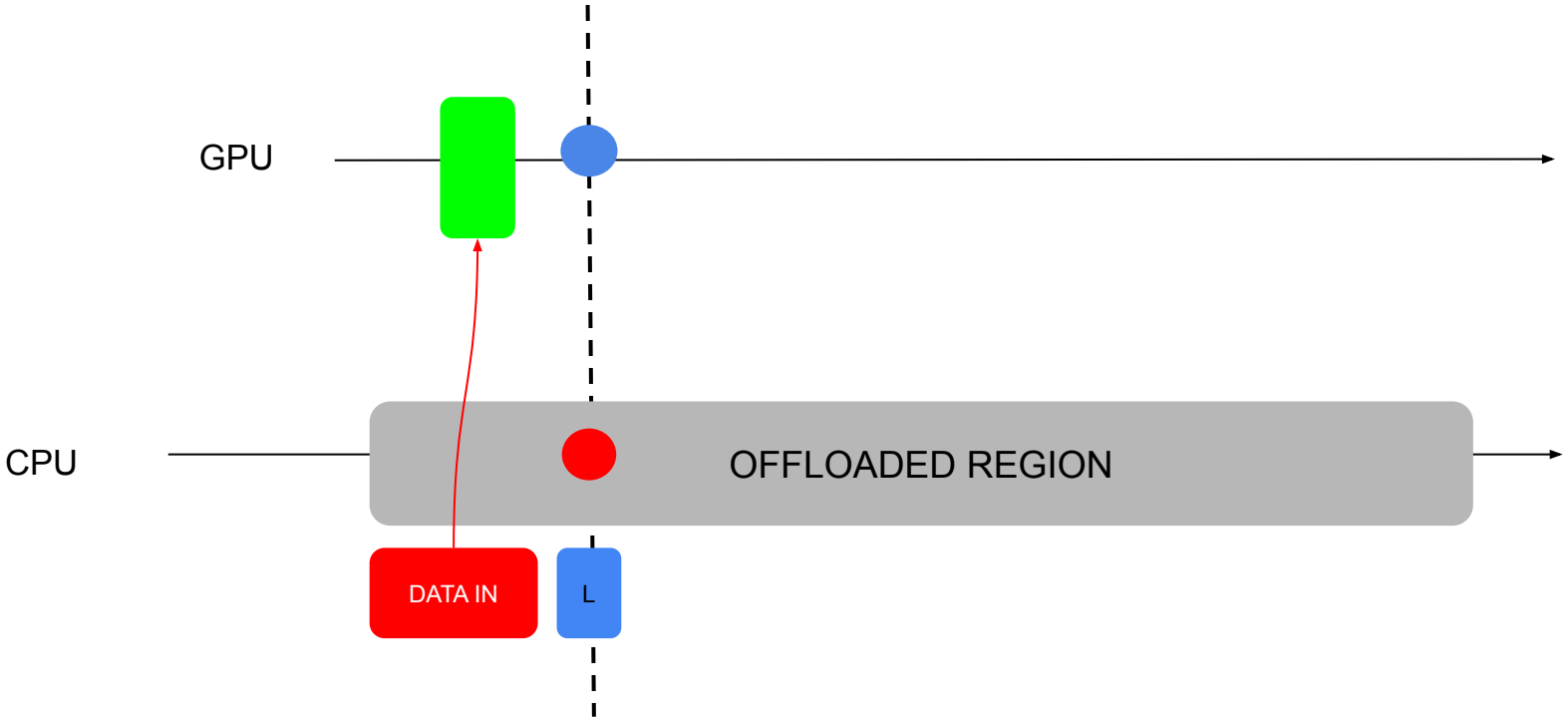
Traces in heterogeneous programs



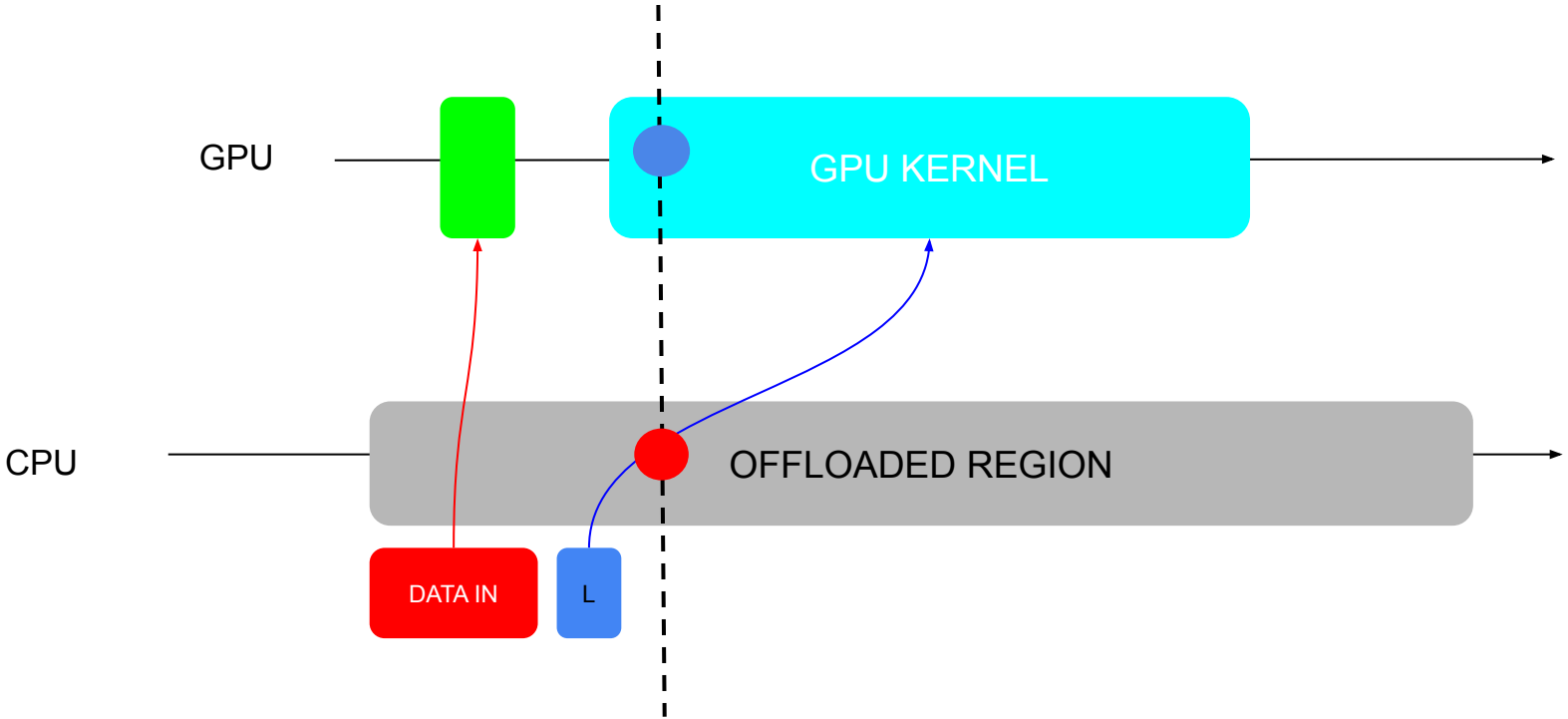
Traces in heterogeneous programs



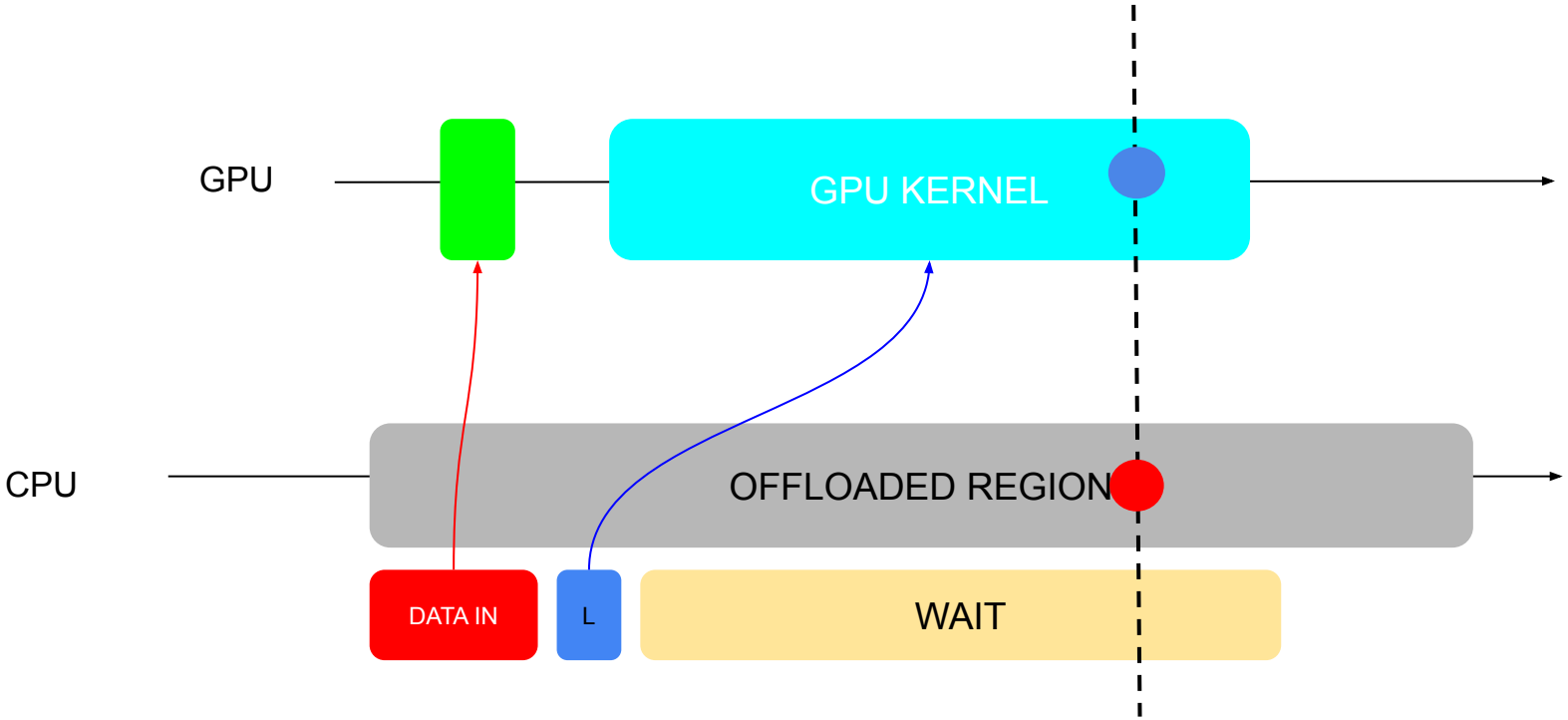
Traces in heterogeneous programs



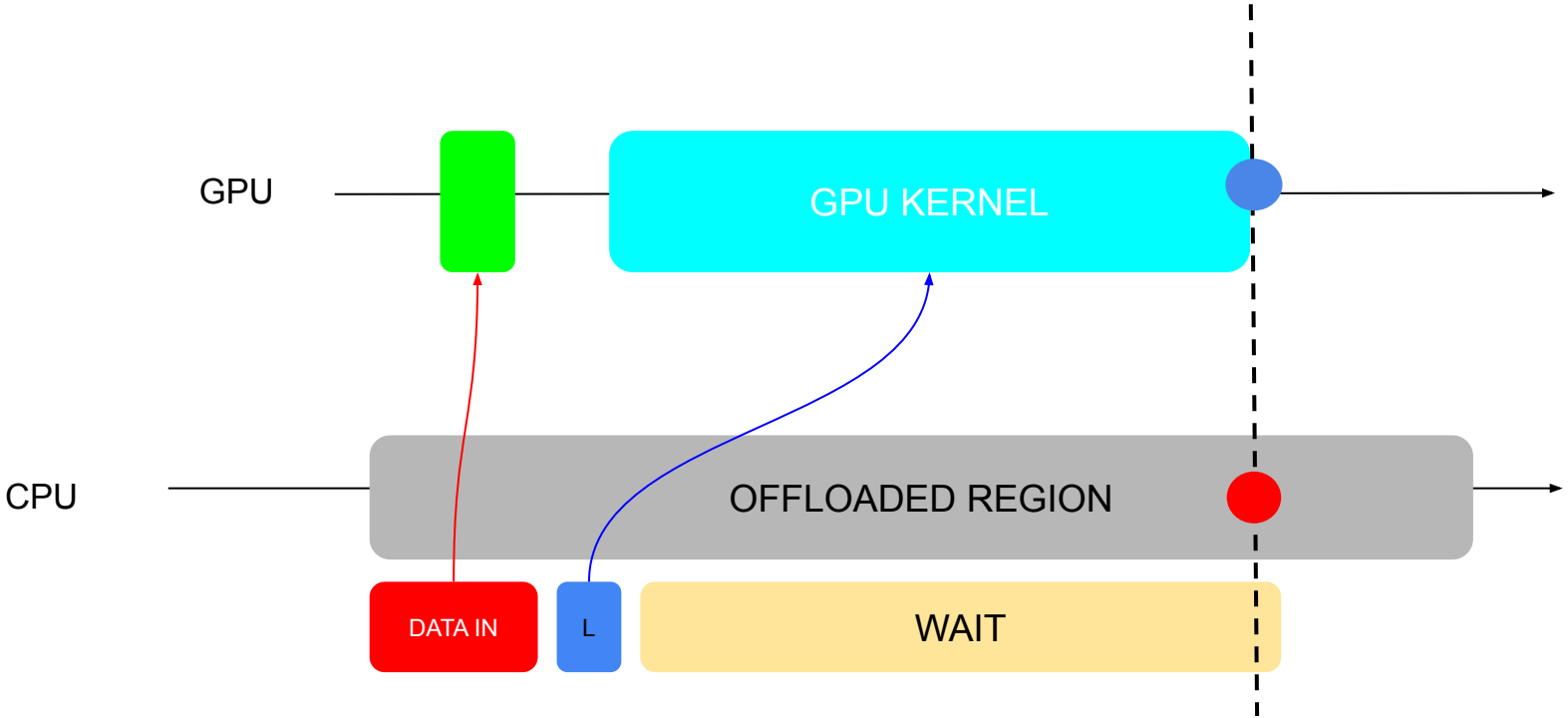
Traces in heterogeneous programs



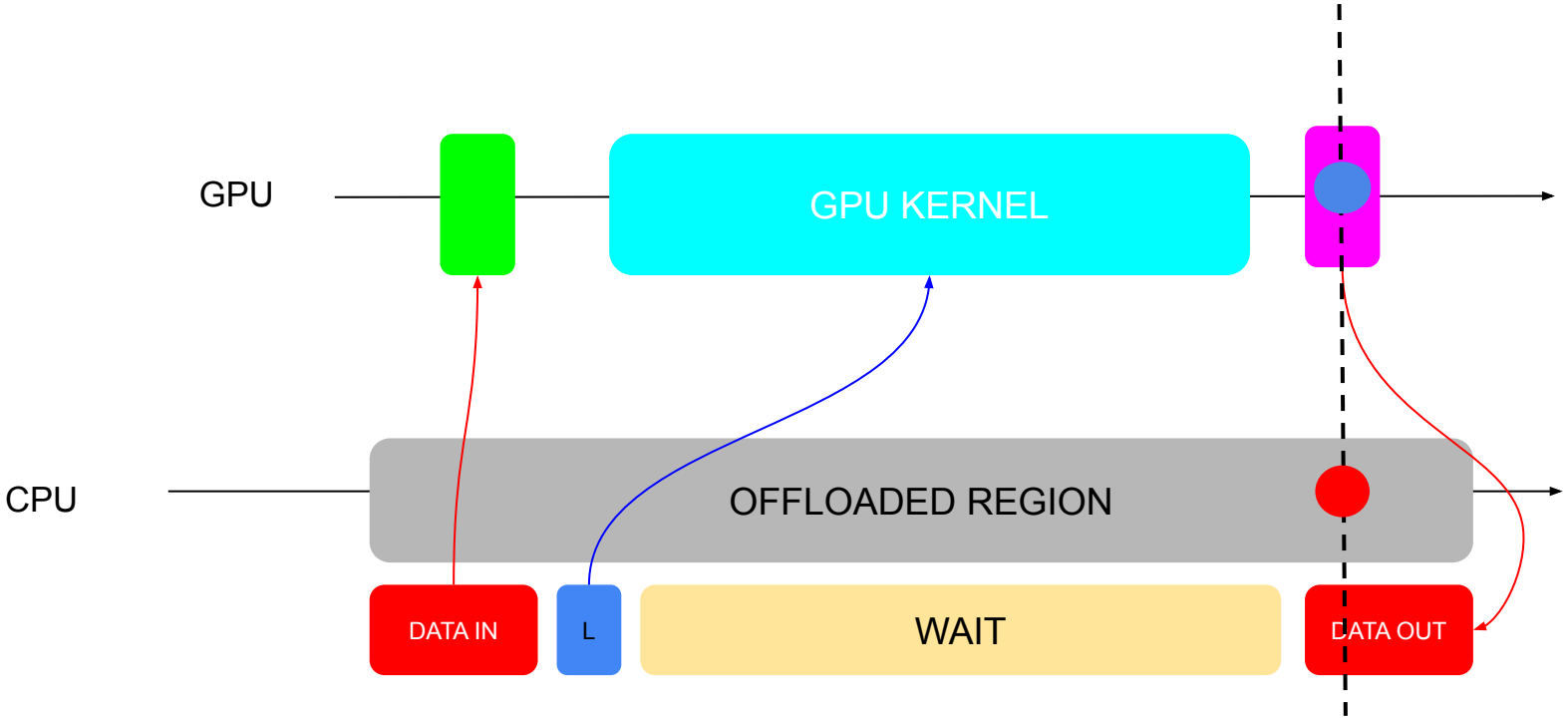
Traces in heterogeneous programs



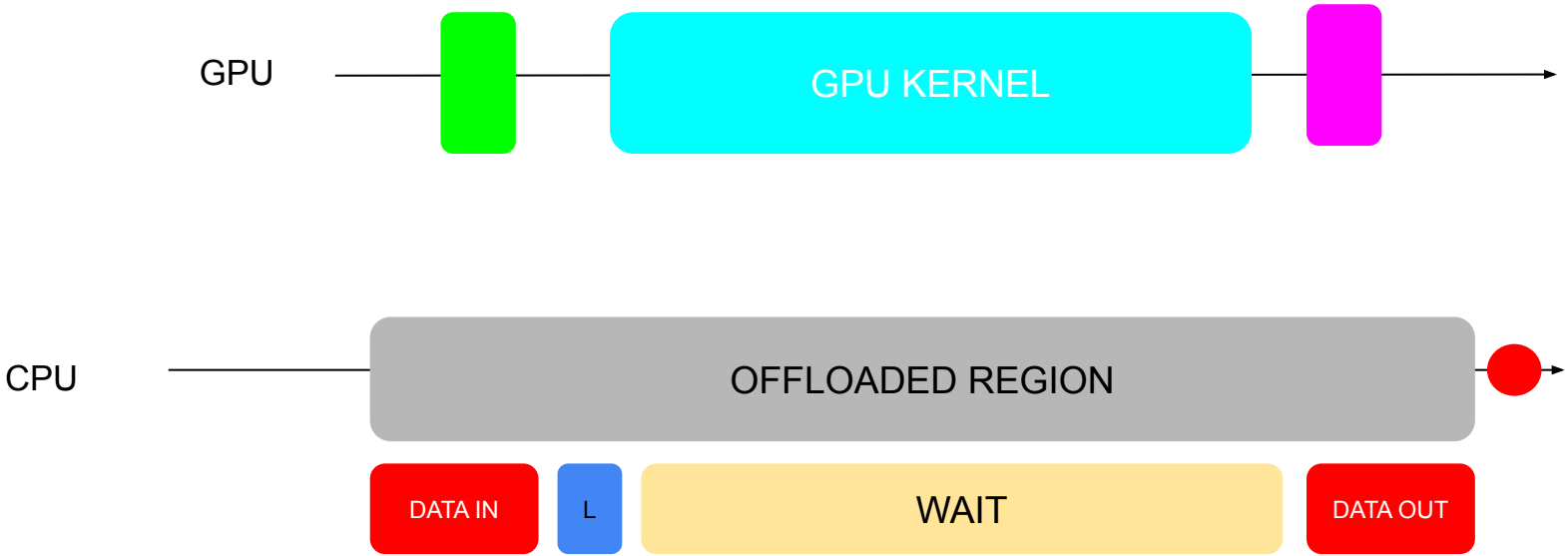
Traces in heterogeneous programs



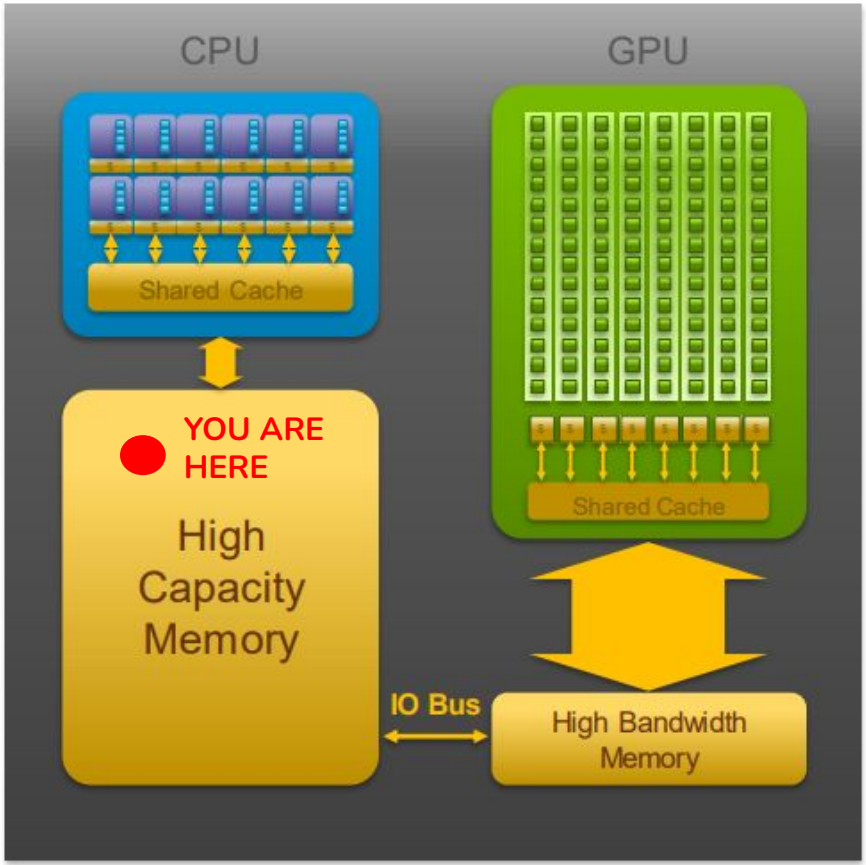
Traces in heterogeneous programs



Traces in heterogeneous programs

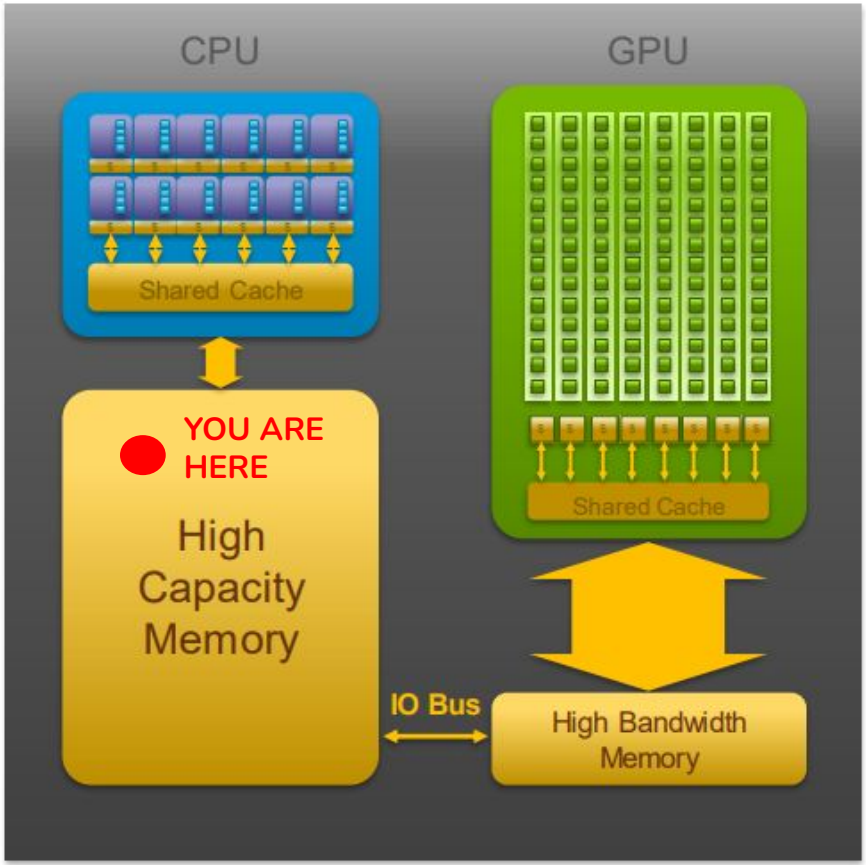


Heterogeneous programming



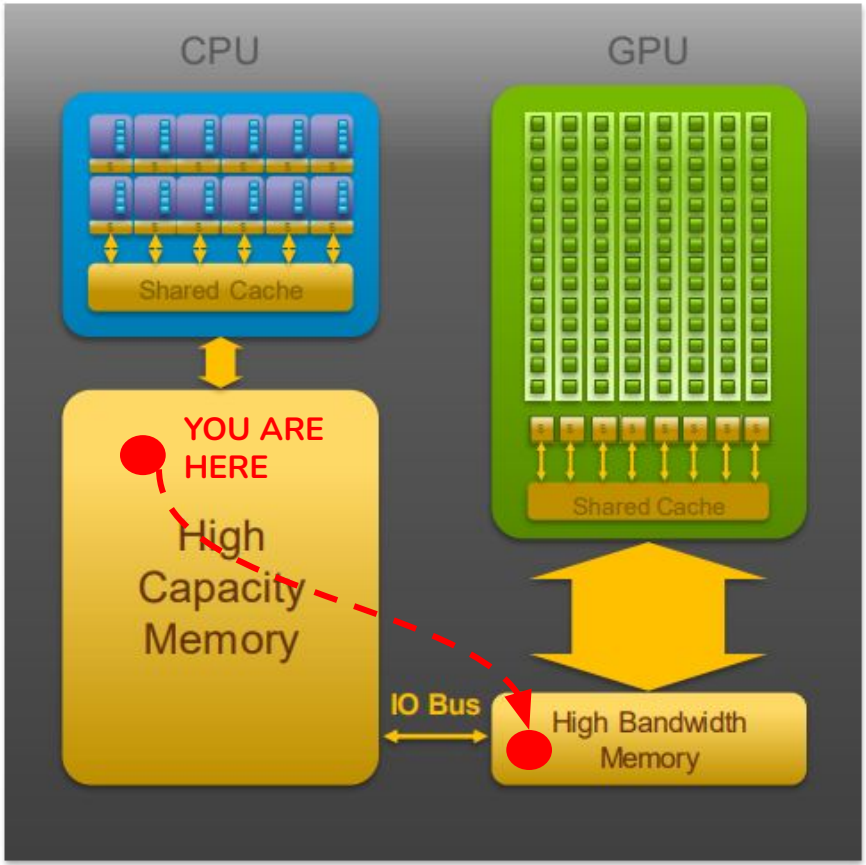
- The program starts on the CPU

Heterogeneous programming



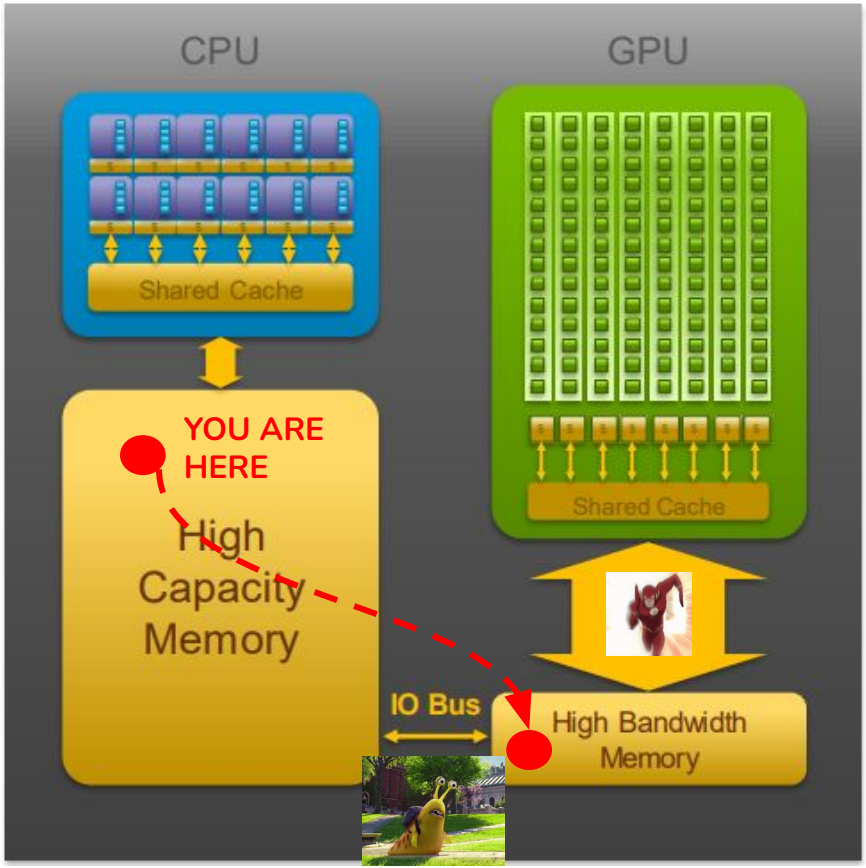
- The program starts on the CPU
- GPUs and CPUs have separated memories

Heterogeneous programming



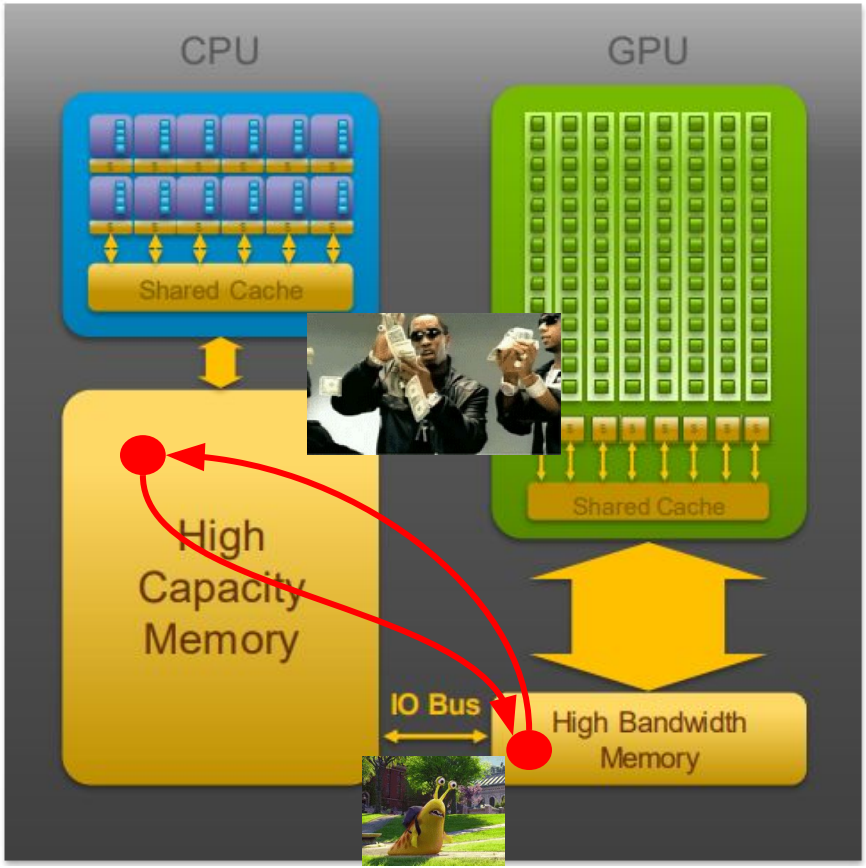
- The program starts on the CPU
- GPUs and CPUs have separated memories
- GPUs need data in their own memory

Heterogeneous programming



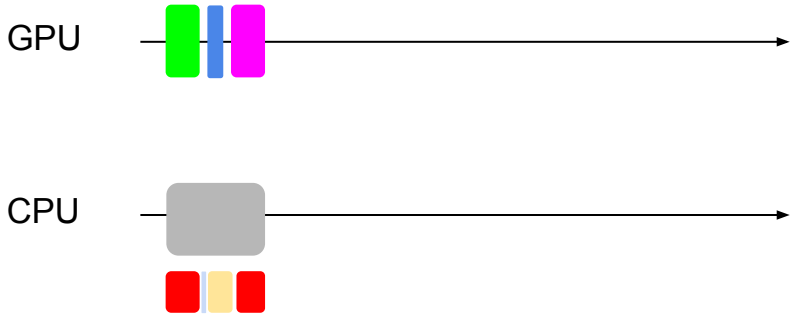
- The program starts on the CPU
- GPUs and CPUs have separated memories
- GPUs need data in their own memory
- IO bus is slow compared to GPU BW

Heterogeneous programming

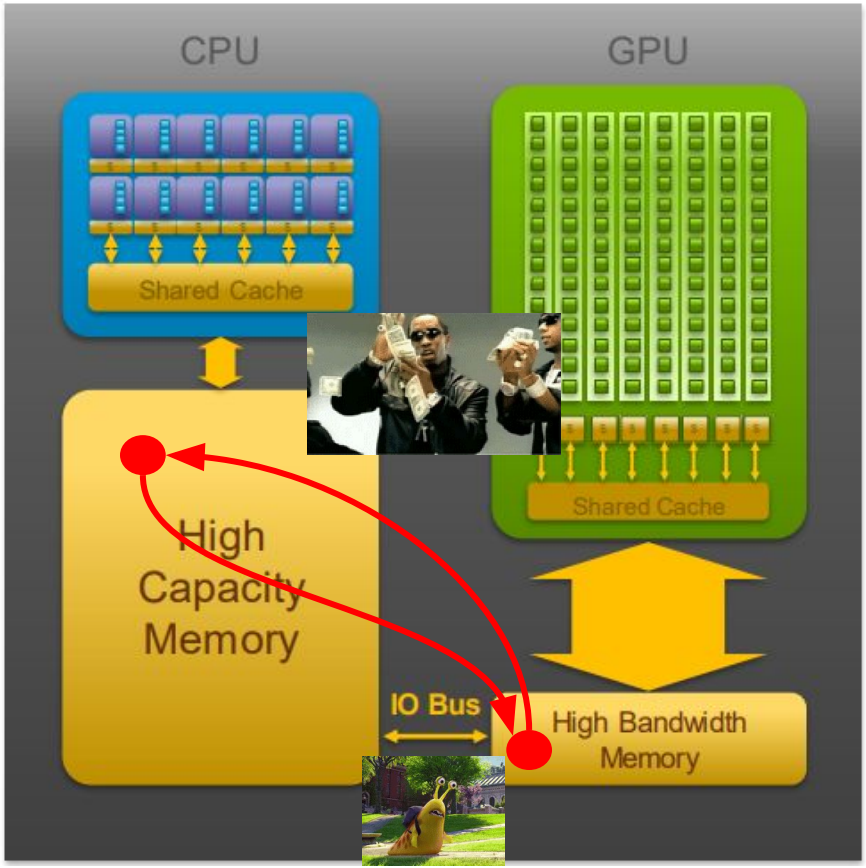


- The program starts on the CPU
- GPUs and CPUs have separated memories
- GPUs need data in their own memory
- IO bus is slow compared to GPU BW

MOVING DATA IS EXPENSIVE

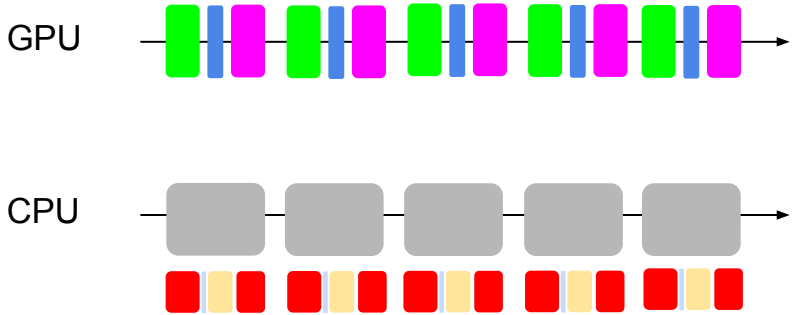


Heterogeneous programming

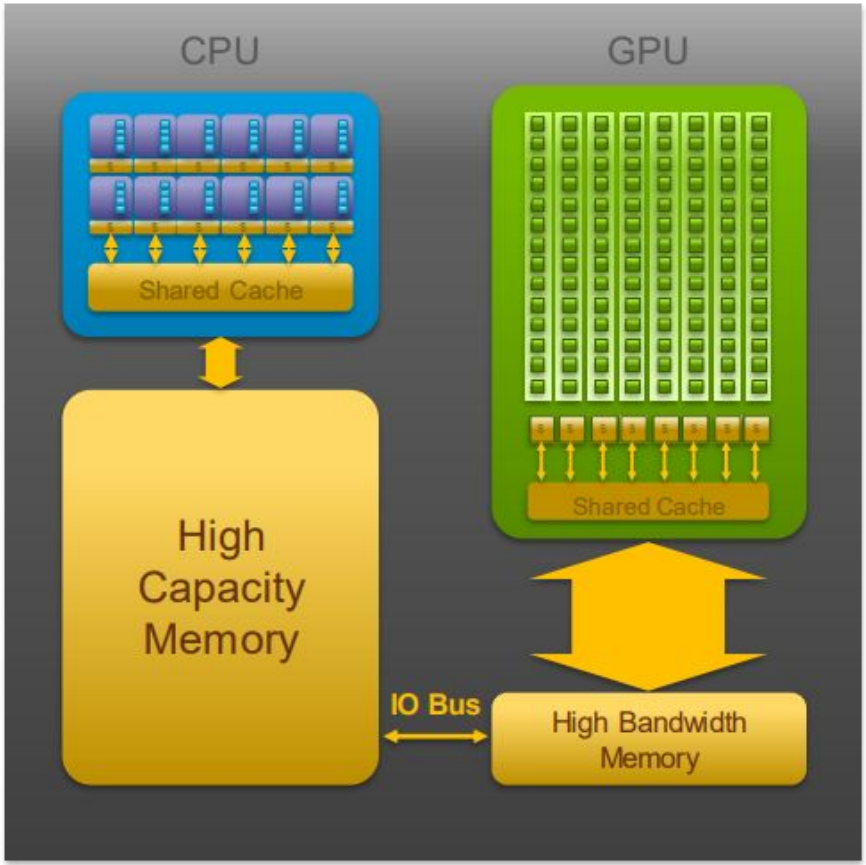


- The program starts on the CPU
- GPUs and CPUs have separated memories
- GPUs need data in their own memory
- IO bus is slow compared to GPU BW

MOVING DATA IS EXPENSIVE



Heterogeneous programming

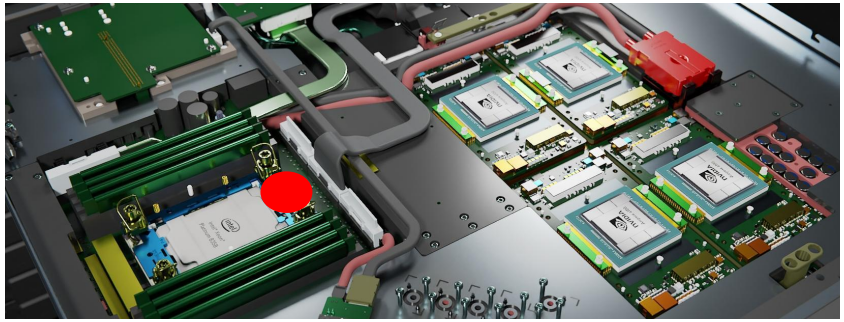


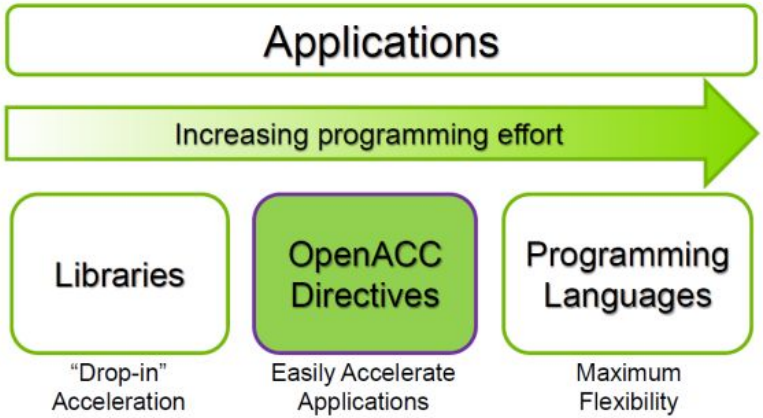
- The program starts on the CPU
- GPUs and CPUs have separated memories
- GPUs need data in their own memory
- IO bus is slow compared to GPU BW

MOVING DATA IS EXPENSIVE

IMPROVE DATA LOCALITY

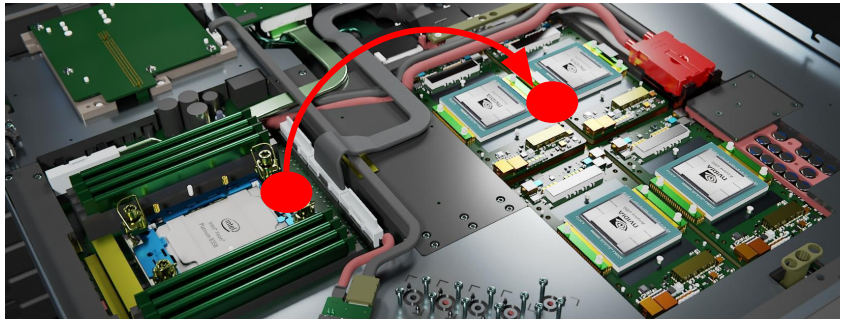
Heterogeneous programming



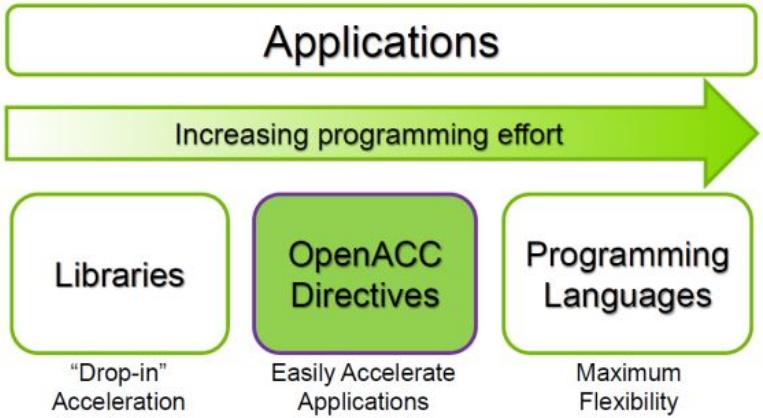


- The program starts on the CPU

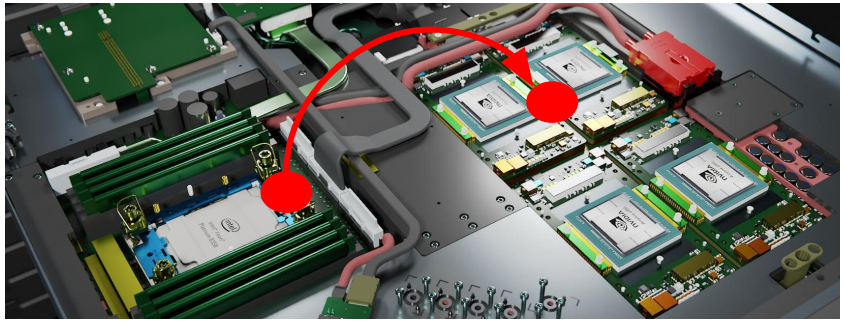
Heterogeneous programming



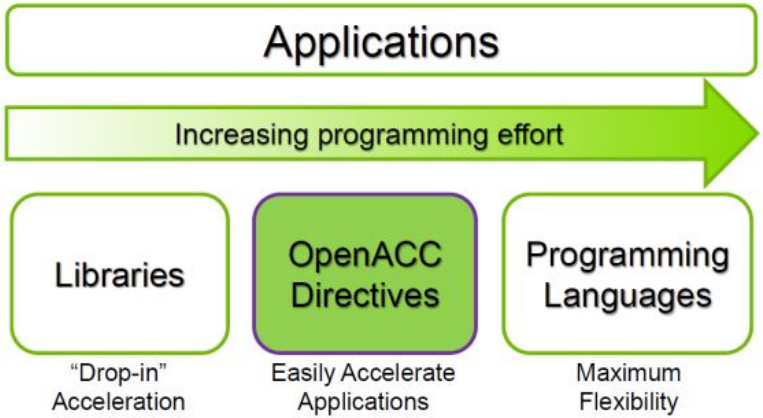
- The program starts on the CPU
- Many ways to offload kernels to GPUs



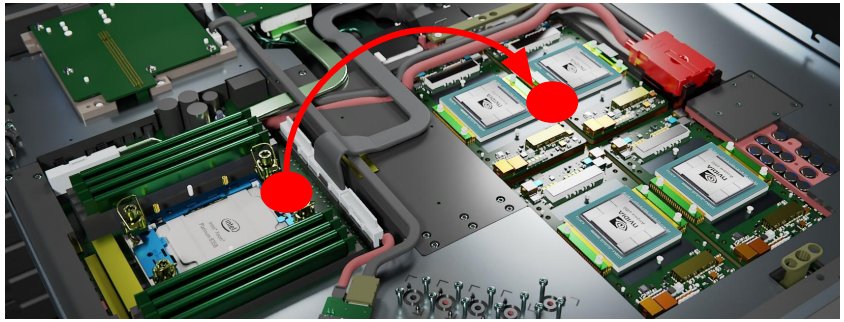
Heterogeneous programming



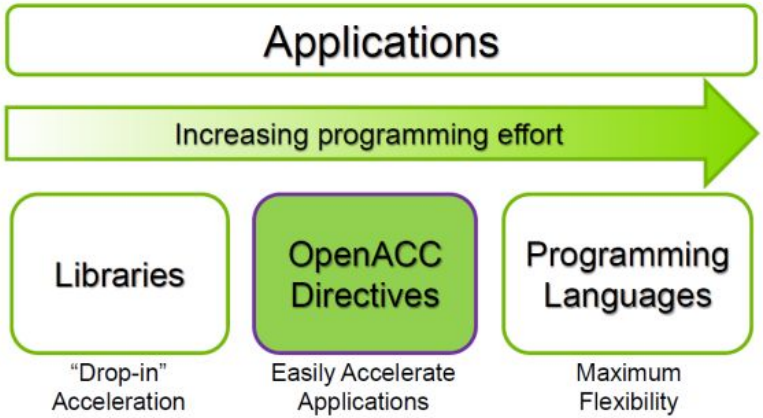
- The program starts on the CPU
- Many ways to offload kernels to GPUs

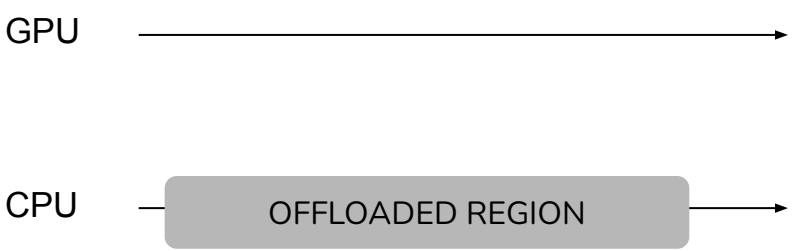


Heterogeneous programming

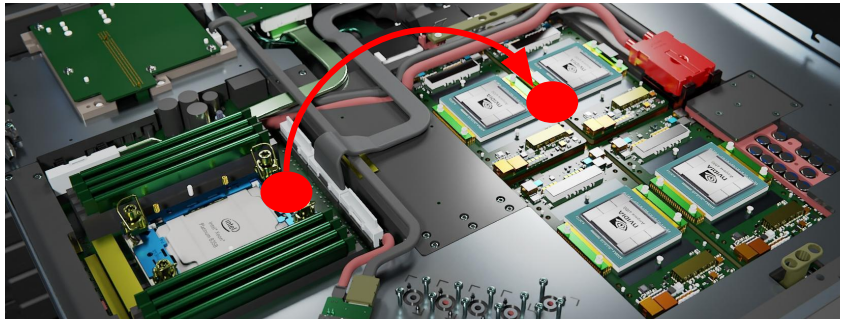


- The program starts on the CPU
- Many ways to offload kernels to GPUs
- There is a time needed to launch the kernels ("latency")

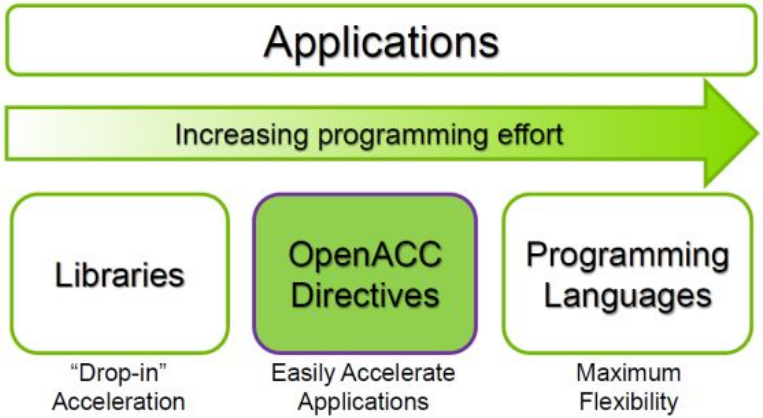


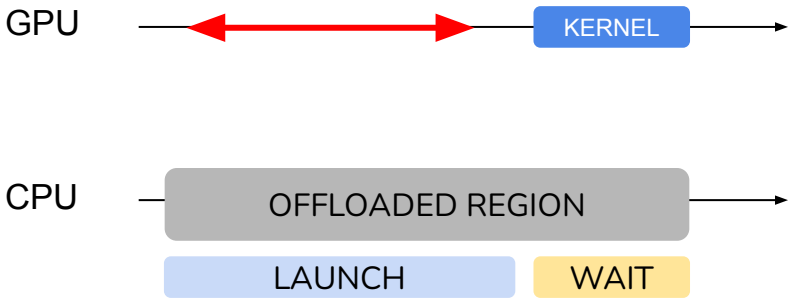


Heterogeneous programming

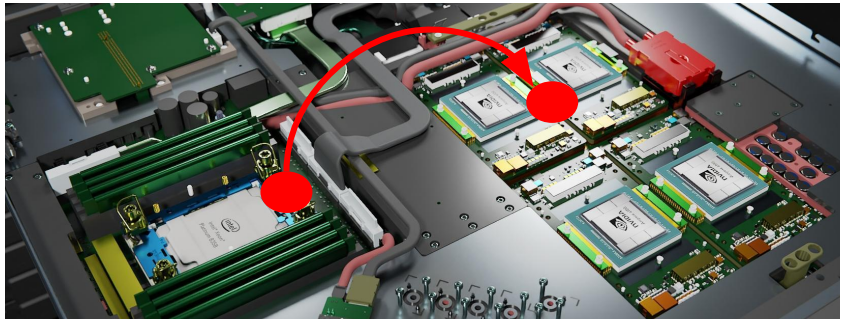


- The program starts on the CPU
- Many ways to offload kernels to GPUs
- There is a time needed to launch the kernels ("latency")

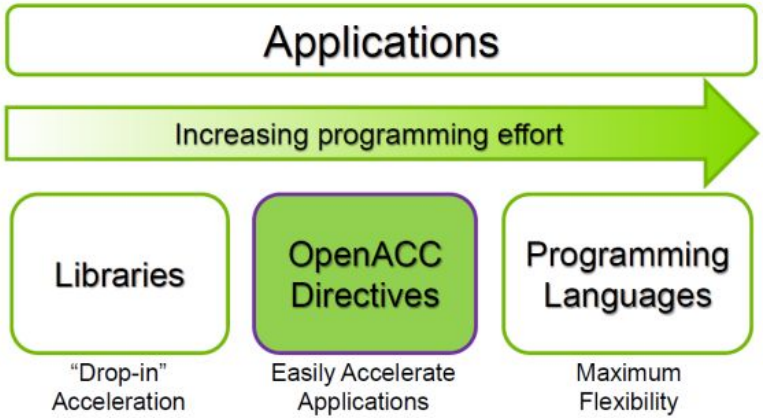


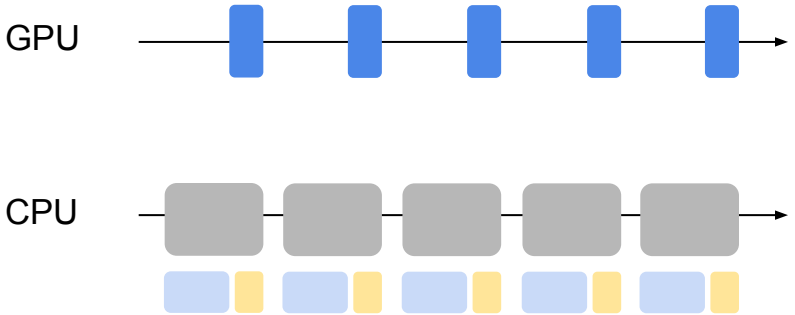


Heterogeneous programming

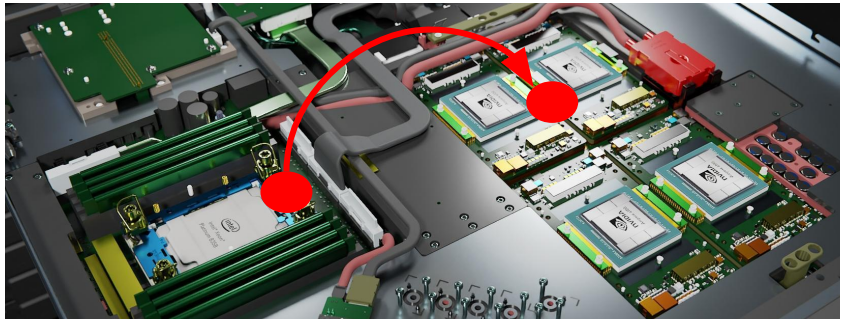


- The program starts on the CPU
- Many ways to offload kernels to GPUs
- There is a time needed to launch the kernels (“latency”)





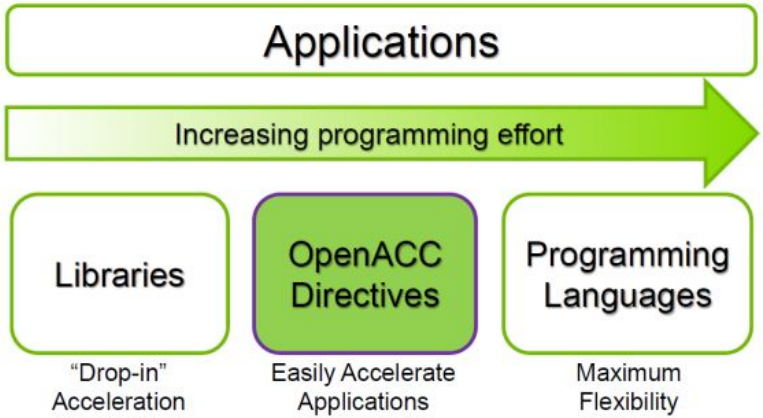
Heterogeneous programming



- The program starts on the CPU
- Many ways to offload kernels to GPUs
- There is a time needed to launch the kernels (“latency”)

KERNEL LAUNCH ADDS OVERHEAD

EXPOSE AS MUCH PARALLELISM AS POSSIBLE



NSight Systems
Tracing GPUs and CPUs

l.bellentani@cineca.it
Tutorial on Profiling @ MHPC - ICTP

NVIDIA profiling and tracing tools

MEASUREMENT

SAMPLING	GPU	TRACES
INSTRUMENTATION	CPU	SUMMARIES

- Instrument CUDA APIs
- Instrument OpenACC regions
- Instrument MPI, OpenMP for the CPU
- Instrument GPU kernels, GPU metrics
- Sample the CPU (usr routines, libraries)
- Network metrics (H2D, D2D, NIC metrics)

NVIDIA profiling and tracing tools

```
nsys [command_switch][optional command_switch_options][application] [optional application_options]
```

Command switches

- **profile**	All-in-one , needed for concurrent profiling sessions
- start - launch - cancel - shutdown - stop	Interactive mode
- export - stats - analyze	Postprocessing to export / textual summaries

Command switch options

- **--trace=**[nvtx,cuda,osrt],mpi,openmp,openacc,cublas,cusolver,...	Events to trace
- --cuda-memory-usage	Collect GPU memory usage
- --mpi-impl	Select mpi implementation (openmpi, mpich)
- --nic-metrics	Network bandwidth

NVIDIA profiling and tracing tools

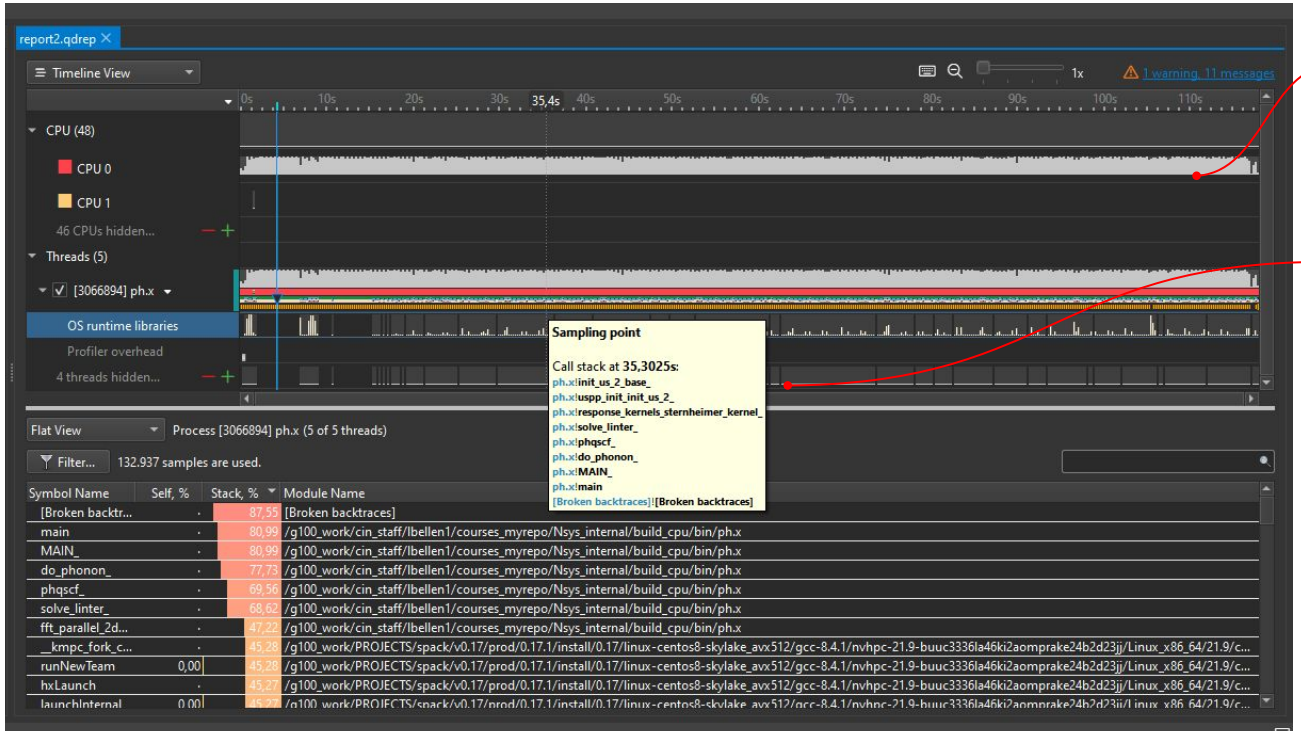
@CINECA cluster : to avoid /tmp/ saturation, do not forget wrapping nsys command with

```
rm -rf /tmp/nvidia
ln -s $TMPDIR /tmp/nvidia
nsys <...>
rm -rf /tmp/nvidia
```

https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.2%3A+MARCONI100+UserGuide#UG3.2:MARCONI100UserGuide-GPUprofilers(NvidiaNsightSystem)

FLAT PROFILES (sampling)

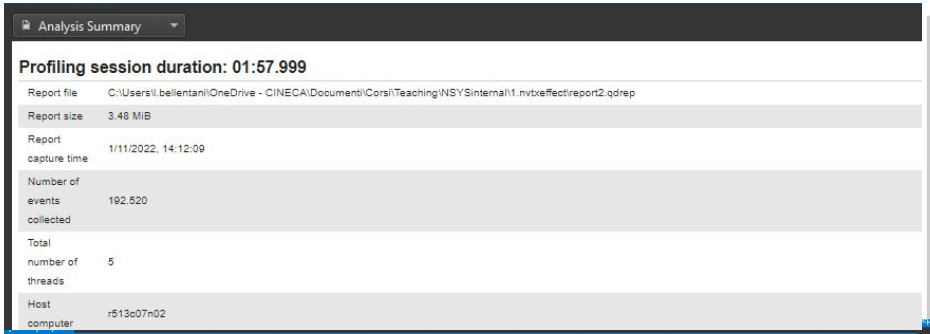
```
nsys profile --sample=cpu --backtrace=fp
```

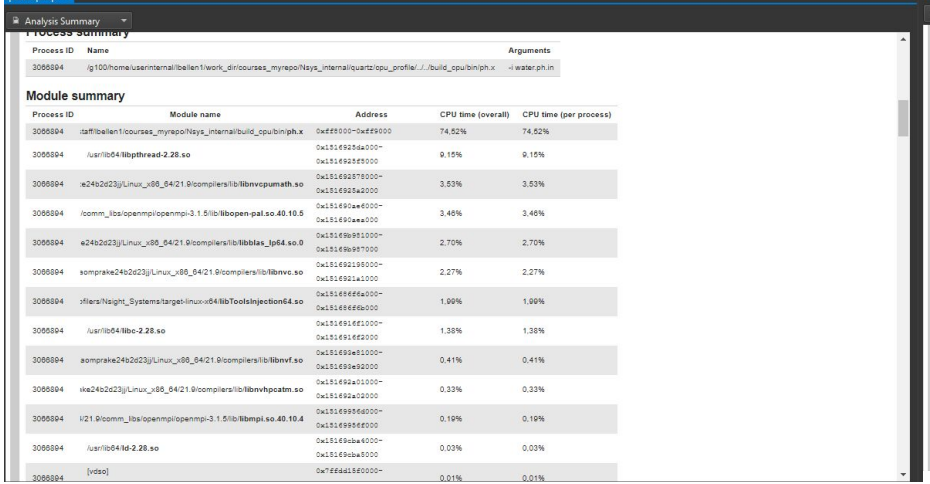


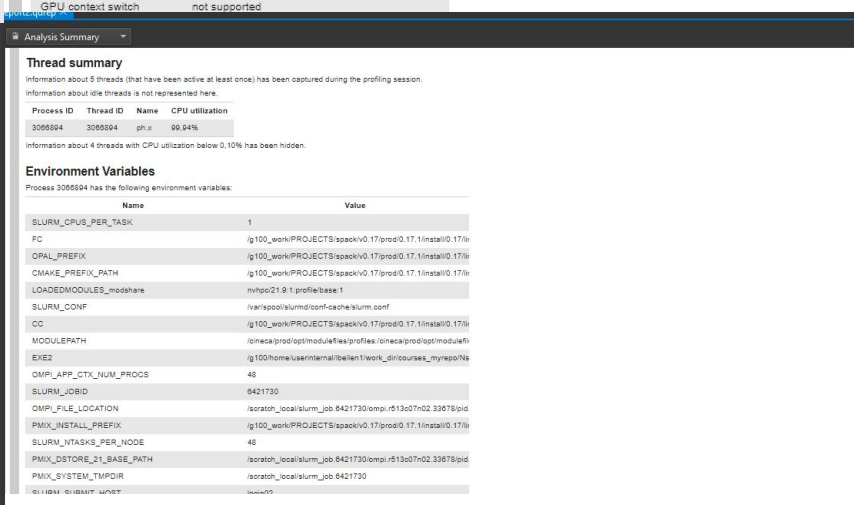
CPU bindling

Debug symbols are needed for backtraces!

Analysis summaries

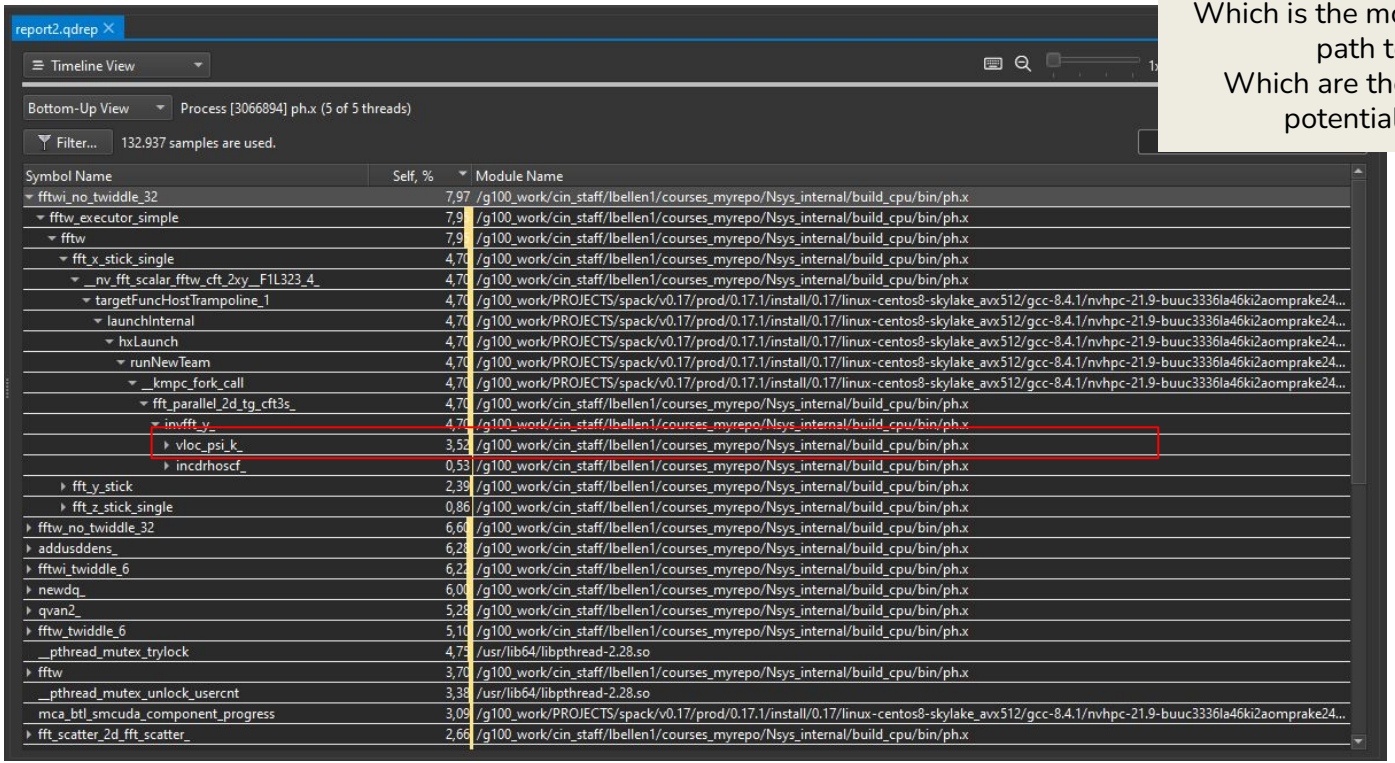






Top-down/bottom-up profiles

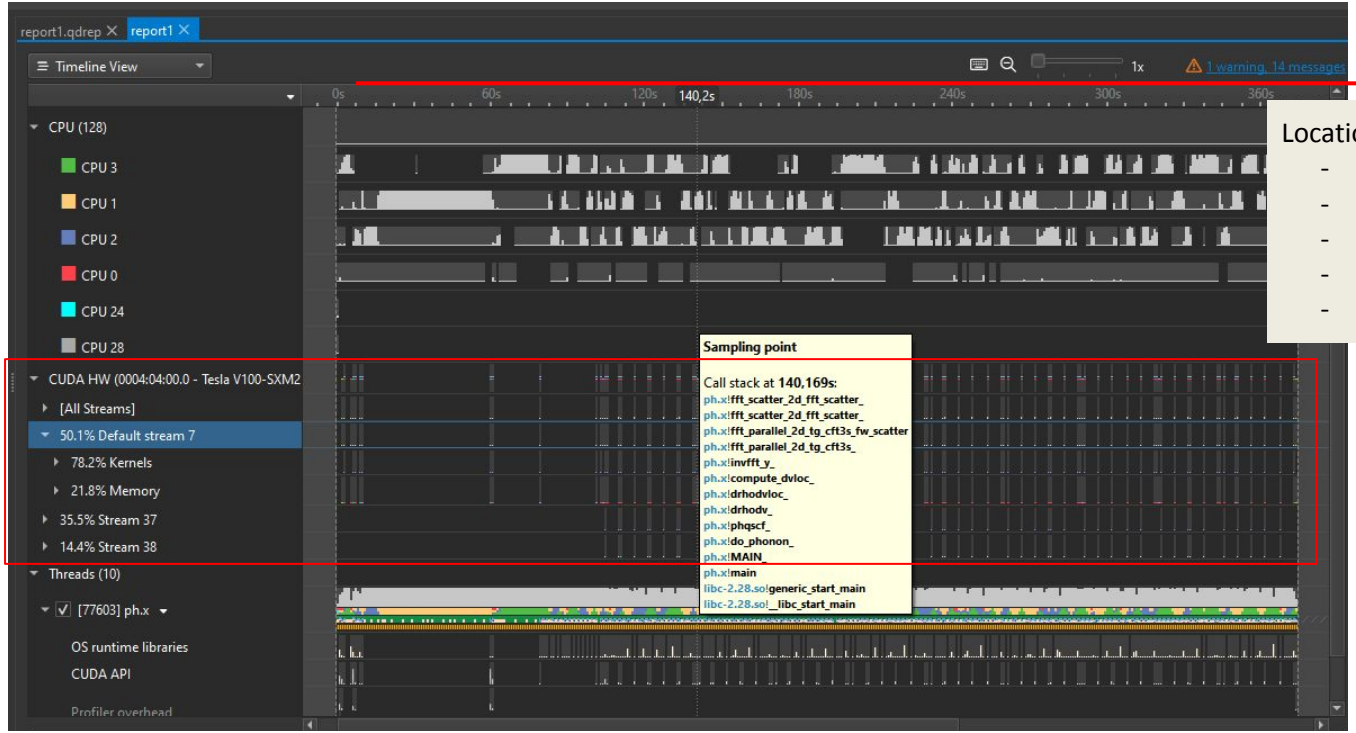
Identify the most time consuming calling path



Which is the most time consuming path to offload?
Which are the parent routines potentially involved?

Timeline view

```
nsys profile  --trace=osrt,cuda --sample=cpu --backtrace=fp
```

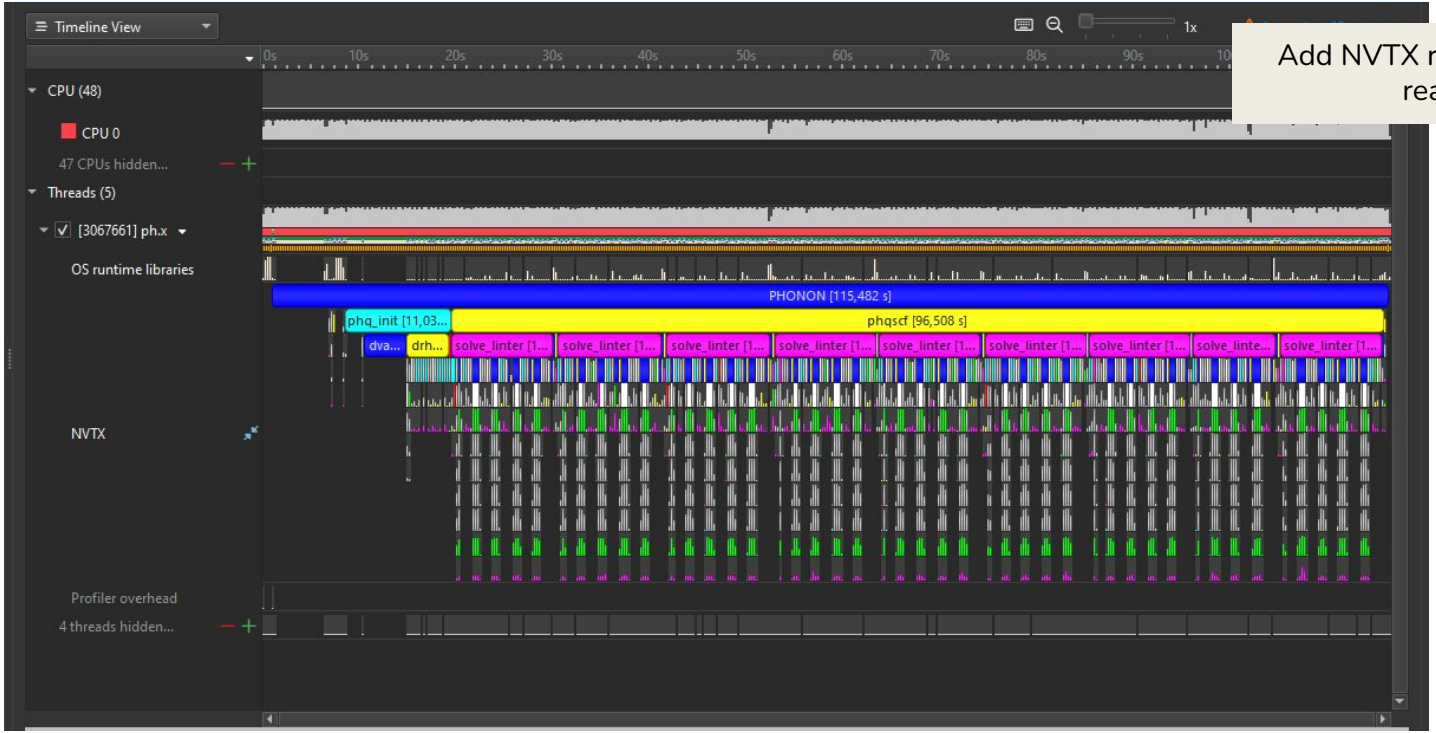


Location can be

- architecture (CPU/PU)
- stream
- thread
- library (CUDA, CUPTI)
- runtime (OpenACC)

NVTX ranges

```
nsys profile  --trace=osrt,nvtx --sample=cpu --backtrace=fp
```



Add NVTX ranges to increase readability

NVTX ranges

Source code is not automatically instrumented. NVTX library (`nvToolsExt.h`) can be used to enhance trace readability:

- C-based API to annotate events and code ranges, to be visualized in the Nsight System timeline
- Limited overhead when the tool is not attached to the application

Functionalities

- **NVTX Markers** → annotate events occurring at a specific time
- **NVTX Ranges** → annotate timespan of code regions

Events

- associated to message (ASCII, Unicode) → **A, W** variant of function calls
- associated to structure with attributes → **Ex** variant of function calls

```
//Set to default

nvtxEventAttributes_t eventAttrib = {0};

//Declare version and size

eventAttrib.version = NVTX_VERSION;
eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;

// Message type and message
// // ASCII

eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;
eventAttrib.message.ascii = __FUNCTION__ ":ascii";
// //UNICODE

eventAttrib2.messageType = NVTX_MESSAGE_TYPE_UNICODE;
eventAttrib2.message.unicode = __FUNCTIONW__ L
":unicode \u2603 snowman";

// Color type and color

eventAttrib.colorType = NVTX_COLOR_ARGB;
eventAttrib.color = COLOR_YELLOW;
```

NVTX ranges

Source code is not automatically instrumented. NVTX library (`nvToolsExt.h`) can be used to enhance trace readability:

- C-based API to annotate events and code ranges, to be visualized in the Nsight System timeline
- Limited overhead when the tool is not attached to the application

Functionalities

- **NVTX Markers** → annotate events occurring at a specific time
- **NVTX ranges** → annotate timespan of code regions

Events

- associated to message (ASCII, Unicode) → **A, W** variant of function calls
- associated to structure with attributes → **Ex** variant of function calls

```
//Set to default

nvtxEventAttributes_t eventAttrib = {0};

//Declare version and size

eventAttrib.version = NVTX_VERSION;
eventAttrib.size = NVTX_EVENT_ATTRIB_STRUCT_SIZE;

// Message type and message
// // ASCII

eventAttrib.messageType = NVTX_MESSAGE_TYPE_ASCII;
eventAttrib.message.ascii = __FUNCTION__ ":ascii";
// //UNICODE

eventAttrib2.messageType = NVTX_MESSAGE_TYPE_UNICODE;
eventAttrib2.message.unicode = __FUNCTIONW__ L
":unicode \u2603 snowman";

// Color type and color

eventAttrib.colorType = NVTX_COLOR_ARGB;
eventAttrib.color = COLOR_YELLOW;
```

NVTX ranges

Markers (events at a specific time)

```
nvtxMarkA(__FUNCTION__ ":nvtxMarkA");
nvtxMarkW(__FUNCTIONW__ L":nvtxMarkW");
nvtxMarkEx(&eventAttrib);
```

Ranges (nested time ranges occurring on a CPU thread)

```
start:  nvtxRangePushEx
        nvtxRangePushA
        nvtxRangePushW
end :  nvtxRangePop
```

```
// for message-only events

nvtxRangePushA(__FUNCTION__ ":nvtxRangePushA");
[... code here ...]
nvtxRangePop();

nvtxRangePushW(__FUNCTIONW__
L":nvtxRangePushW");
[... code here ...]
nvtxRangePop();

// for structured events

nvtxRangePushEx(&eventAttrib);
[... code here ...]
nvtxRangePop();
```

NVTX ranges

Markers (events at a specific time)

```
nvtxMarkA(__FUNCTION__ ":nvtxMarkA");
nvtxMarkW(__FUNCTIONW__ L":nvtxMarkW");
nvtxMarkEx(&eventAttrib);
```

Ranges (nested time ranges occurring on a CPU thread)

start: `nvtxRangePushEx`
`nvtxRangePushA`
`nvtxRangePushW`

end : `nvtxRangePop`

```
// for message-only events

nvtxRangePushA(__FUNCTION__ ":nvtxRangePushA");
[... code here ...]
nvtxRangePop();

nvtxRangePushW(__FUNCTIONW__
L":nvtxRangePushW");
[... code here ...]
nvtxRangePop();

// for structured events

nvtxRangePushEx(&eventAttrib);
[... code here ...]
nvtxRangePop();
```

NVTX ranges

Fortran module

Wrapper to NVTX APIs available at https://github.com/maxcuda/NVTX_example

```
subroutine nvtxStartRange(name,id)
  character(kind=c_char,len=*) :: name
  integer, optional:: id
  type(nvtxEventAttributes):: event
  character(kind=c_char,len=256) :: trimmed_name
  integer:: i

  trimmed_name=trim(name)//c_null_char

  ! move scalar trimmed_name into character array tempName
  do i=1,LEN(trim(name)) + 1
     tempName(i) = trimmed_name(i:i)
  enddo

  if ( .not. present(id)) then
    call nvtxRangePush(tempName)
  else
    event%color=col(mod(id,7)+1)
    event%message=c_loc(tempName)
    call nvtxRangePushEx(event)
  end if
end subroutine

subroutine nvtxEndRange
  call nvtxRangePop
end subroutine

end module nvtx
```

```
type, bind(C):: nvtxEventAttributes
  integer(C_INT16_T):: version=1
  integer(C_INT16_T):: size=48 !
  integer(C_INT):: category=0
  integer(C_INT):: colorType=1 ! NVTX_COLOR_ARGB = 1
  integer(C_INT):: color
  integer(C_INT):: payloadType=0 ! NVTX_PAYLOAD_UNKNOWN = 0
  integer(C_INT):: reserved0
  integer(C_INT64_T):: payload   ! union uint,int,double
  integer(C_INT):: messageType=1  ! NVTX_MESSAGE_TYPE_ASCII     = 1
  type(C_PTR):: message  ! ascii char
end type
```

```
interface nvtxRangePush
  ! push range with custom label and standard color
  subroutine nvtxRangePushA(name) bind(C, name='nvtxRangePushA')
  use iso_c_binding
  character(kind=C_CHAR) :: name(256)
  end subroutine

  ! push range with custom label and custom color
  subroutine nvtxRangePushEx(event) bind(C, name='nvtxRangePushEx')
  use iso_c_binding
  import:: nvtxEventAttributes
  type(nvtxEventAttributes):: event
  end subroutine
end interface

interface nvtxRangePop
  subroutine nvtxRangePop() bind(C, name='nvtxRangePop')
  end subroutine
end interface
```

Defines a derived type for the event

Fills color and message attribute according to the presence or not of the optional ID parameter

Resolves the calling routine in *Ex or *A API according to input

Summaries

nsys stats report.nsys-rep

Different kind of metrics can be summarized: nvtxsum, cudaapisum, gpukernsum, gpumemtimesum, etc.

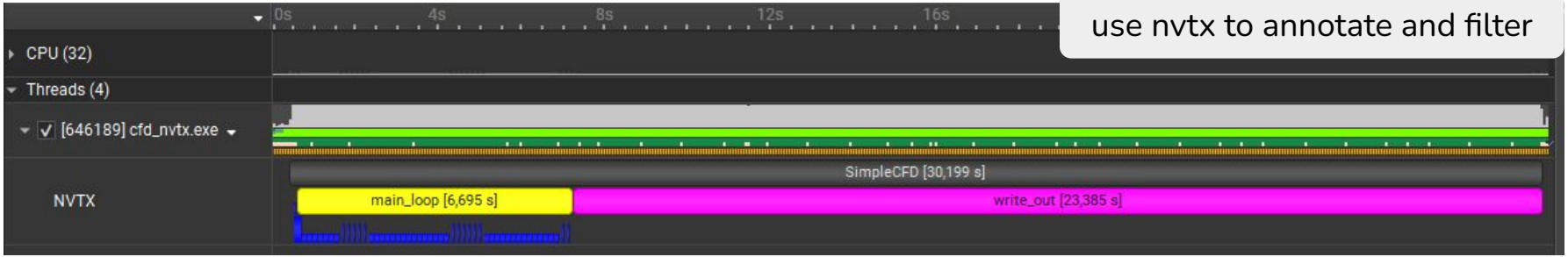
```
nsys stats -r openaccsum report_acc_data1.nsys-rep
Generating SQLite file report_acc_data1.sqlite from report_acc_data1.nsys-rep
Exporting 63118 events: [==================================================100%]
Processing [report_acc_data1.sqlite] with
[/leonardo/prod/spack/03/install/0.19/linux-rhel8-icelake/gcc-8.5.0/nvhpc-23.1-x5lw6edfmfuot2ipna3wseallzl4oolm/Linux_x86_64/23.1/profilers/Nsight_Systems/host-linux-x64/reports/openaccsum.py]...

** OpenACC Summary (openaccsum):

Time(%)  Total Time (ns)  Num Calls    Avg (ns)      Med (ns)     Min (ns)    Max (ns)    StdDev (ns)               Name
-------  ---------------  ---------  ------------  ------------  ----------  -----------  ------------  ------------------------------
   27.9    2,422,224,252         50  48,444,485.0  45,729,730.5  45,576,626  176,454,147  18,476,396.7  Exit Data@jacobi.f90:14
   26.4    2,292,084,948         50  45,841,699.0  45,729,495.0  45,514,774   47,135,248     366,299.5  Exit Data@cfd.f90:144
   20.5    1,779,791,765         50  35,595,835.3  35,474,408.0  35,253,494   36,959,563     390,115.3  Enter Data@cfd.f90:144
   19.8    1,719,850,039         50  34,397,000.8  33,854,316.0  33,666,671   55,305,149   3,046,996.4  Enter Data@jacobi.f90:14
    1.4      123,216,371         50   2,464,327.4   2,328,897.0   2,309,384    8,864,364     923,773.8  Wait@jacobi.f90:21
    1.3      115,610,293         50   2,312,205.9   2,307,225.0   2,283,833    2,386,689      21,258.0  Wait@cfd.f90:150
    0.7       58,974,491        100     589,744.9     584,813.0     497,833      684,812      88,260.9  Wait@cfd.f90:144
    0.7       58,777,669        100     587,776.7     513,375.0      48,488      683,116     100,939.1  Wait@jacobi.f90:14
    0.3       27,516,377          1  27,516,377.0  27,516,377.0  27,516,377   27,516,377           0.0  Device Init@jacobi.f90:14
    0.3       26,569,915         50     531,398.3     530,716.0     523,898      548,907       3,876.9  Compute Construct@jacobi.f90:14
    0.3       26,165,316         50     523,306.3     522,872.0     519,415      530,353       2,344.2  Compute Construct@cfd.f90:144
    0.1        8,457,818      1,000       8,457.8       7,139.5       5,890       72,950       3,862.6  Enqueue Upload@cfd.f90:144
```

CFD code

```
do iter = 1, numiter

   call jacobistep_acc(psitmp, psi, m, n)

!  Compute current error value if required

   if (checkerr .or. iter == numiter) then
      error = deltasq(psitmp, psi, m, n)
      error = sqrt(error)
      error = error / bnorm
   end if

!  Quit early if we have reached required tolerance

   if (checkerr) then
      if (error .lt. tolerance) then
         write(*,*) 'CONVERGED iteration ', iter, ': terminating'
         exit
      end if
   end if

!  Copy back
   do j=1,m
      do i=1,n
        psi(i,j) = psitmp(i,j)
      end do
   end do

!  End iterative Jacobi loop

   if (mod(iter,printfreq) == 0) then

      if (.not. checkerr) then
         write(*,*) 'completed iteration ', iter
      else
         write(*,*) 'completed iteration ', iter, ', error = ', error
      end if

   end if

end do
```

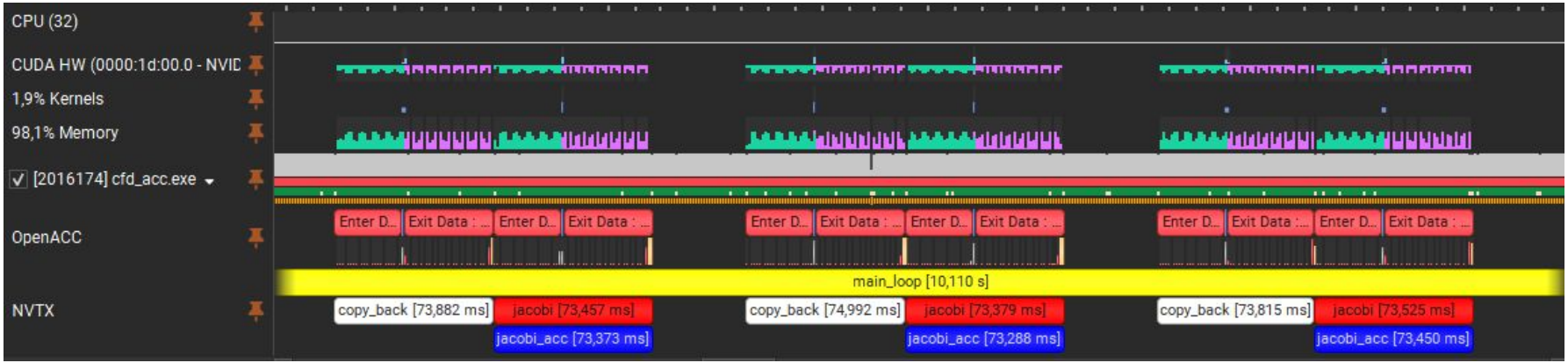
```
subroutine jacobistep_acc(psinew, psi, m, n)
  integer :: m, n
  integer :: i, j

  double precision, dimension(0:m+1, 0:n+1) :: psinew, psi

   do j = 1, n
      do i = 1, m
          psinew(i, j) = 0.25d0*(psi(i+1, j) + psi(i-1, j) + &
                                 psi(i, j+1) + psi(i, j-1)     )
      end do
  end do

end subroutine jacobistep_acc

double precision function deltasq(new, old, m, n)

  integer :: m, n, i, j
  double precision, dimension(0:m+1, 0:n+1) :: new, old
  double precision :: deltasq

  integer :: ierr

   deltasq = 0.d0
     do j = 1, n
      do i = 1, m
      deltasq = deltasq + (new(i,j)-old(i,j))**2
     end do
    end do

end function deltasq
```

CFD code



use nvtx to annotate and filter

```
--trace=nvtx  --capture-range=nvtx --nvtx-capture='main_loop' --env-var=NSYS_NVTX_PROFILER_REGISTER_ONLY=0
```

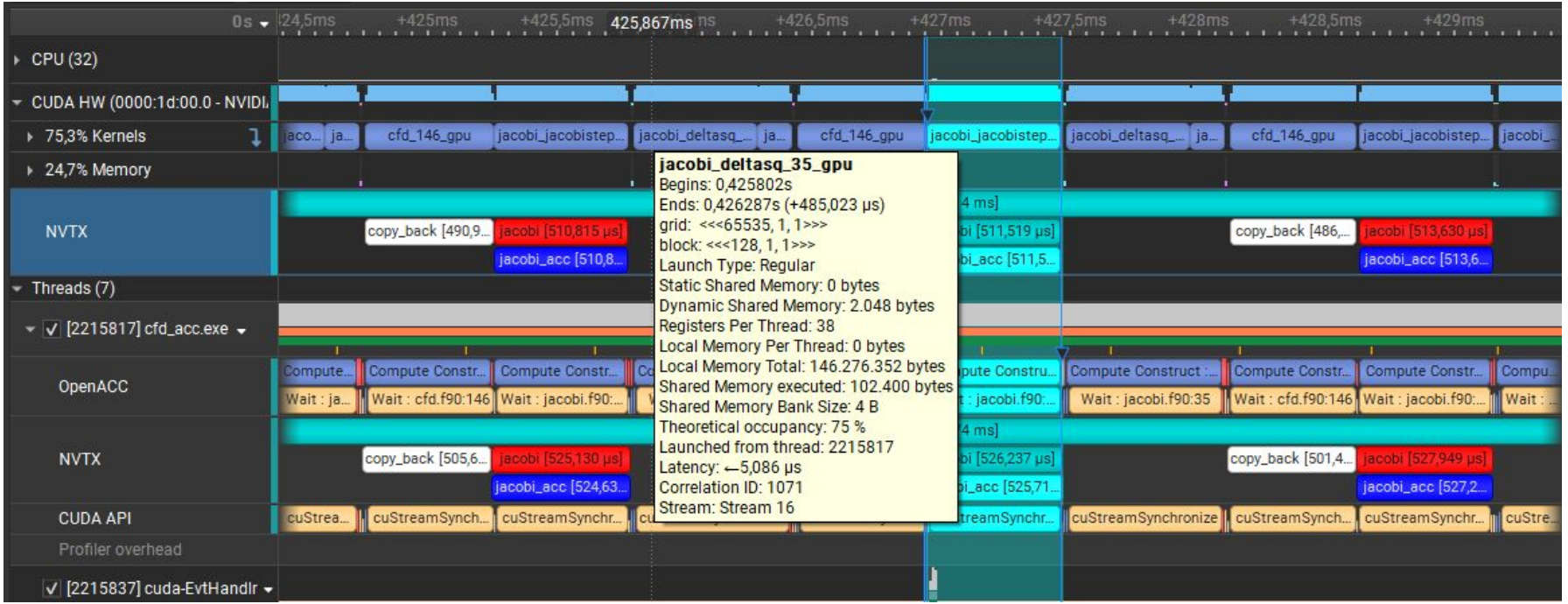


CFD code

check for expensive data movements

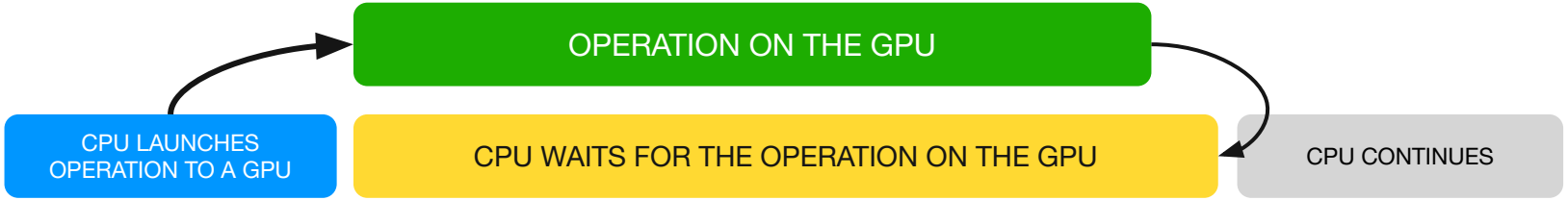
CFD code

analyze how kernels are mapped to GPU hardware

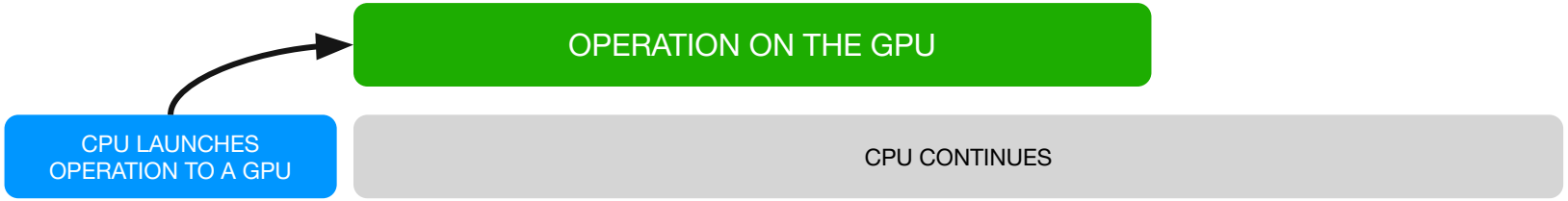


Asynchronous operations

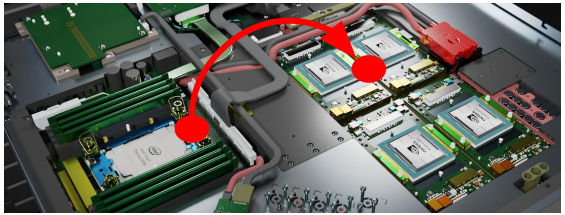
SYNC : CPU waits for the operation on the GPU to be over



ASYNC : CPU does not wait for the operation on the GPU to be over

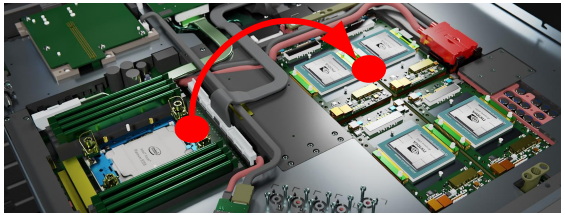


Asynchronous operations

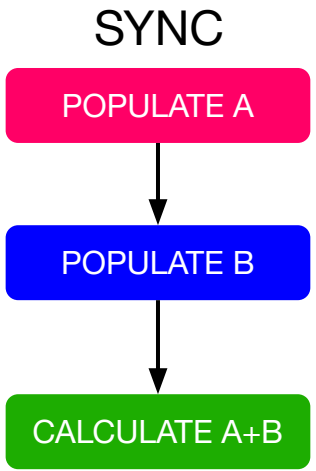


```
for (i = 0; i<n: i++)
 a[i] = 1

for (i = 0; i<n: i++)
 b[i] = 1

for (i = 0; i<n: i++)
c[i] = a[i] + b[i]
```

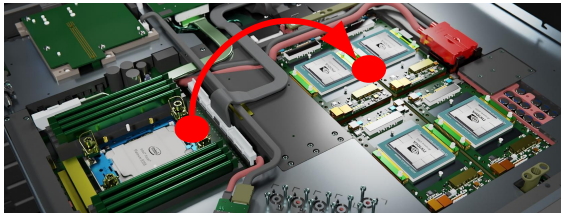
Asynchronous operations



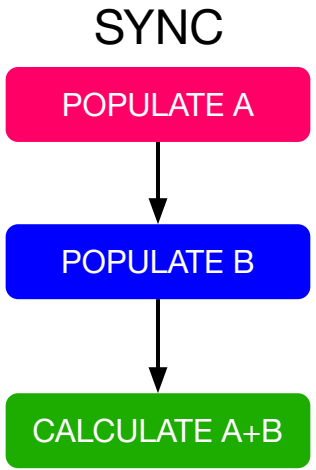
```
for (i = 0; i<n: i++)
 a[i] = 1

for (i = 0; i<n: i++)
 b[i] = 1

for (i = 0; i<n: i++)
c[i] = a[i] + b[i]
```

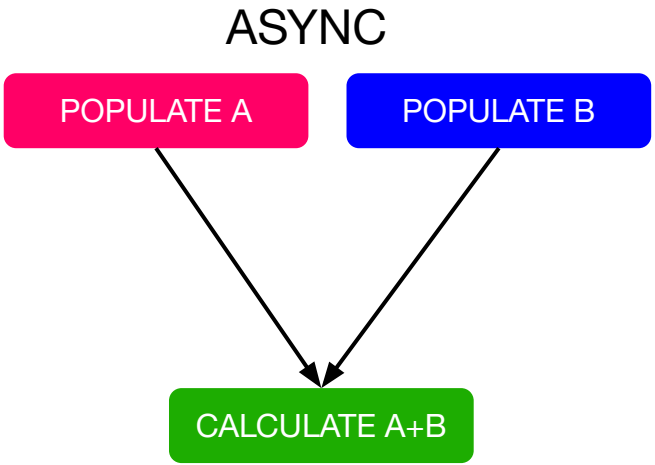


Asynchronous operations

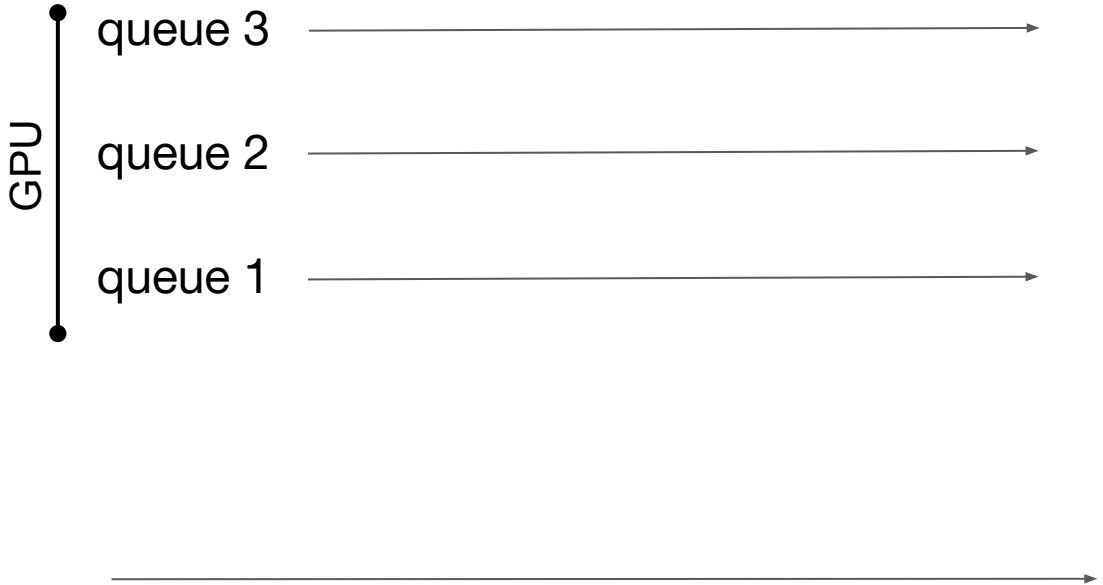


```
for (i = 0; i<n: i++)
 a[i] = 1

for (i = 0; i<n: i++)
 b[i] = 1

for (i = 0; i<n: i++)
c[i] = a[i] + b[i]
```





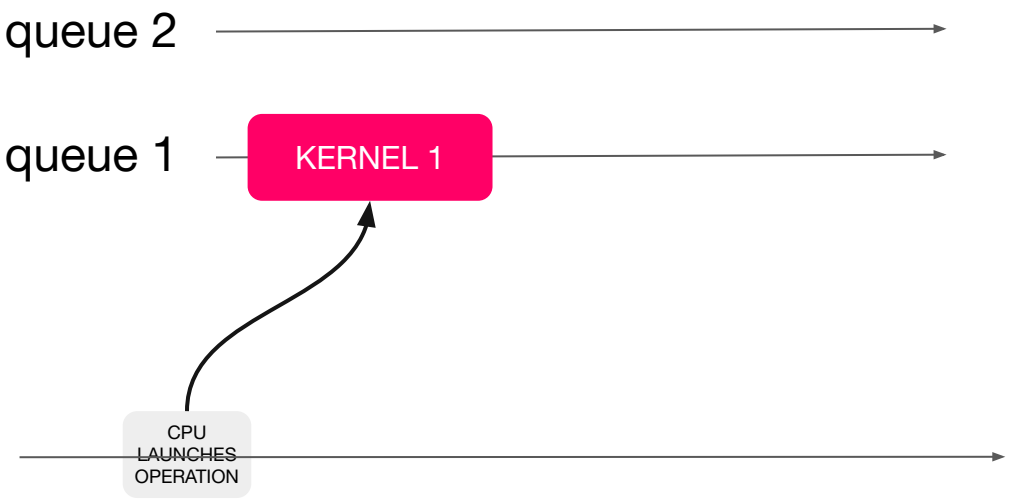
Asynchronous operations & multistream



Asynchronous operations & multistream

```
#pragma acc parallel loop async(1)
for (i = 0; i<n: i++)
 a[i] = 1
```

THIS KERNEL IS COMPUTED ON “STREAM” 1

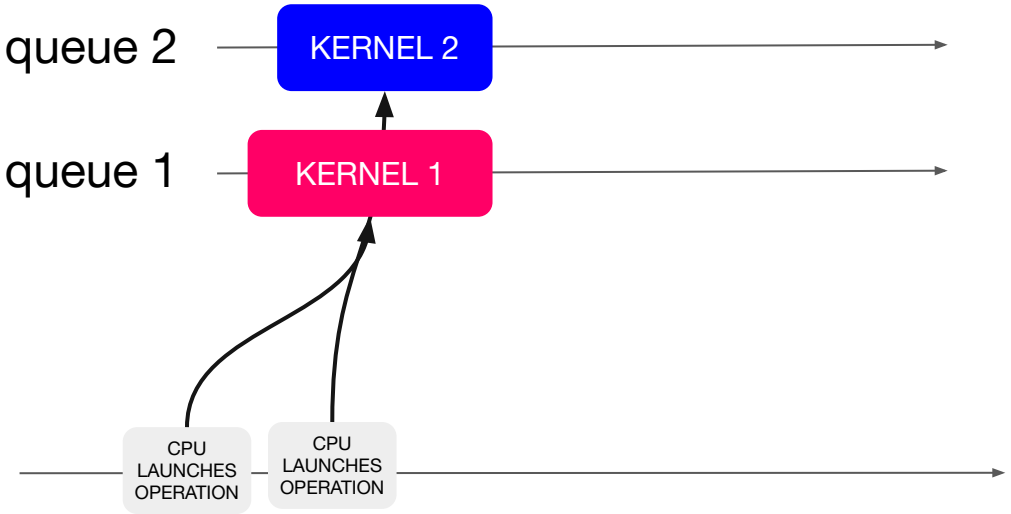


Asynchronous operations & multistream

```
#pragma acc parallel loop async(1)
for (i = 0; i<n: i++)
 a[i] = 1
#pragma acc parallel loop async(2)
for (i = 0; i<n: i++)
 b[i] = 1
```

THIS KERNEL IS COMPUTED ON “STREAM” 1

THIS KERNEL IS COMPUTED ON “STREAM” 2



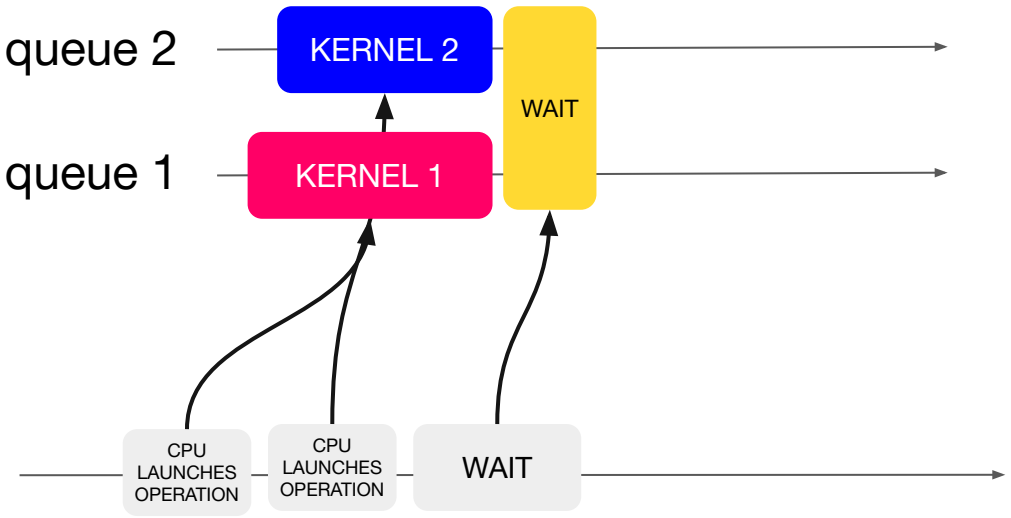
Asynchronous operations & multistream

```
#pragma acc parallel loop async(1)
for (i = 0; i<n: i++)
 a[i] = 1
#pragma acc parallel loop async(2)
for (i = 0; i<n: i++)
 b[i] = 1
#pragma acc wait(1) async(2)
```

THIS KERNEL IS COMPUTED ON “STREAM” 1

THIS KERNEL IS COMPUTED ON “STREAM” 2

THIS KERNEL WAITS FOR OPERATION IN 1 TO BE COMPLETED AND THEN IS PUT IN THE QUEUE OF STREAM 2



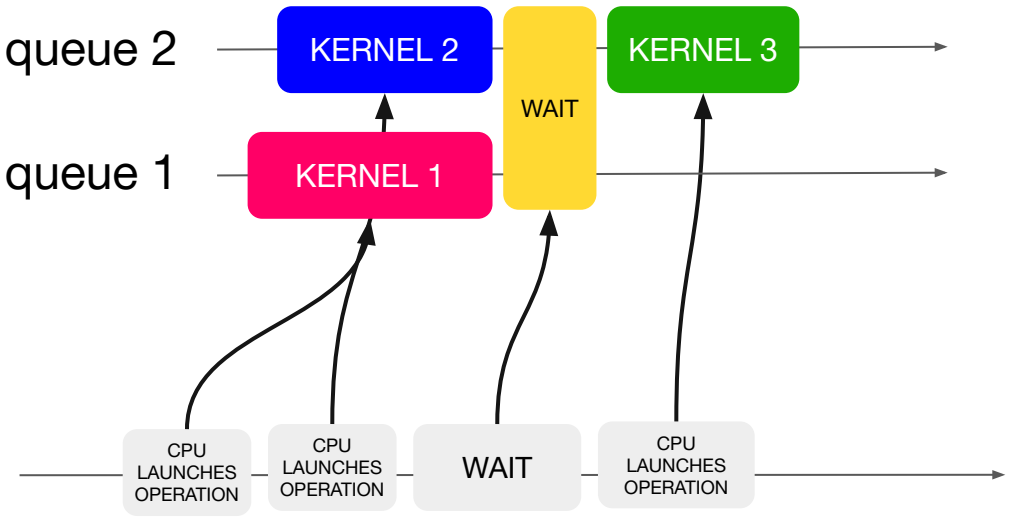
Asynchronous operations & multistream

```
#pragma acc parallel loop async(1)
for (i = 0; i<n: i++)
 a[i] = 1
#pragma acc parallel loop async(2)
for (i = 0; i<n: i++)
 b[i] = 1
#pragma acc wait(1) async(2)
#pragma acc loop async(2)
for (i = 0; i<n: i++)
c[i] = a[i] + b[i]
```

THIS KERNEL IS COMPUTED ON “STREAM” 1

THIS KERNEL IS COMPUTED ON “STREAM” 2

THIS KERNEL WAITS FOR OPERATION IN 1 TO BE COMPLETED AND THEN IS PUT IN THE QUEUE OF STREAM 2



Asynchronous operations & multistream

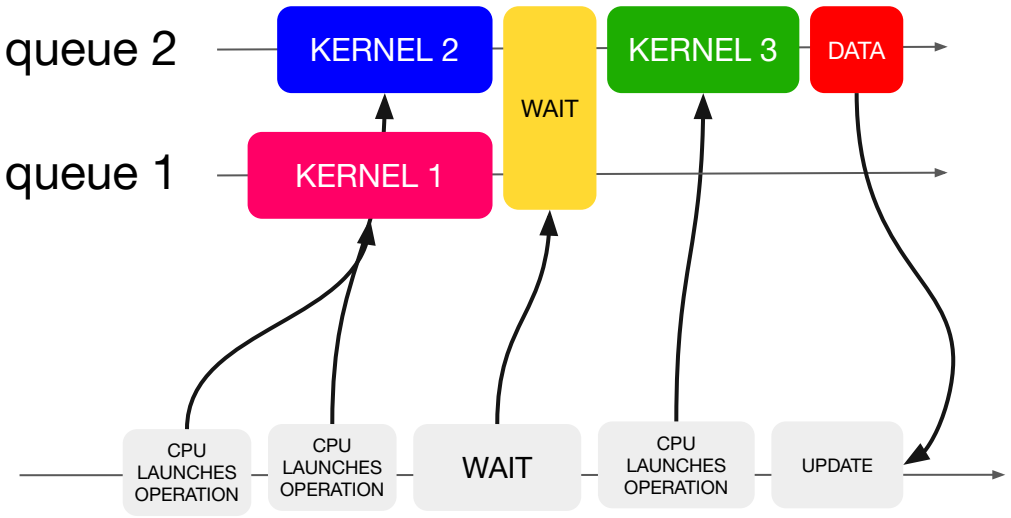
```
#pragma acc parallel loop async(1)
for (i = 0; i<n: i++)
 a[i] = 1
#pragma acc parallel loop async(2)
for (i = 0; i<n: i++)
 b[i] = 1
#pragma acc wait(1) async(2)
#pragma acc loop async(2)
for (i = 0; i<n: i++)
c[i] = a[i] + b[i]
#pragma acc update self[c[0:N]] async(2)
#pragma acc wait
```

THIS KERNEL IS COMPUTED ON “STREAM” 1

THIS KERNEL IS COMPUTED ON “STREAM” 2

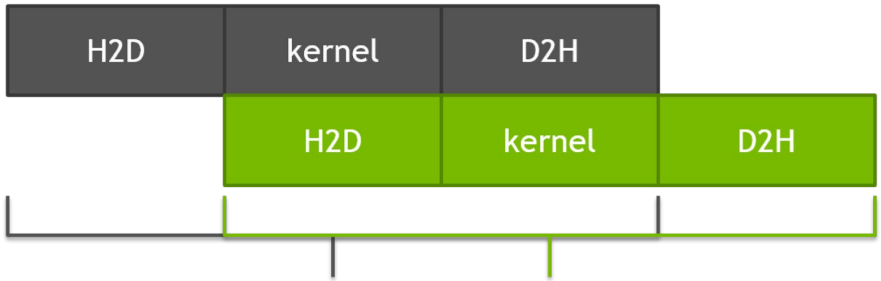
THIS KERNEL WAITS FOR OPERATION IN 1 TO BE COMPLETED AND THEN IS PUT IN THE QUEUE OF STREAM 2

WHEN STREAM2 HAS COMPLETED THE OPERATION, THE HOST IS UPDATED



Asynchronous operations & multistream

Two Independent Operations Serialized



Overlapping Copying and Computation

NOTE: In real applications, your boxes will not be so evenly sized.

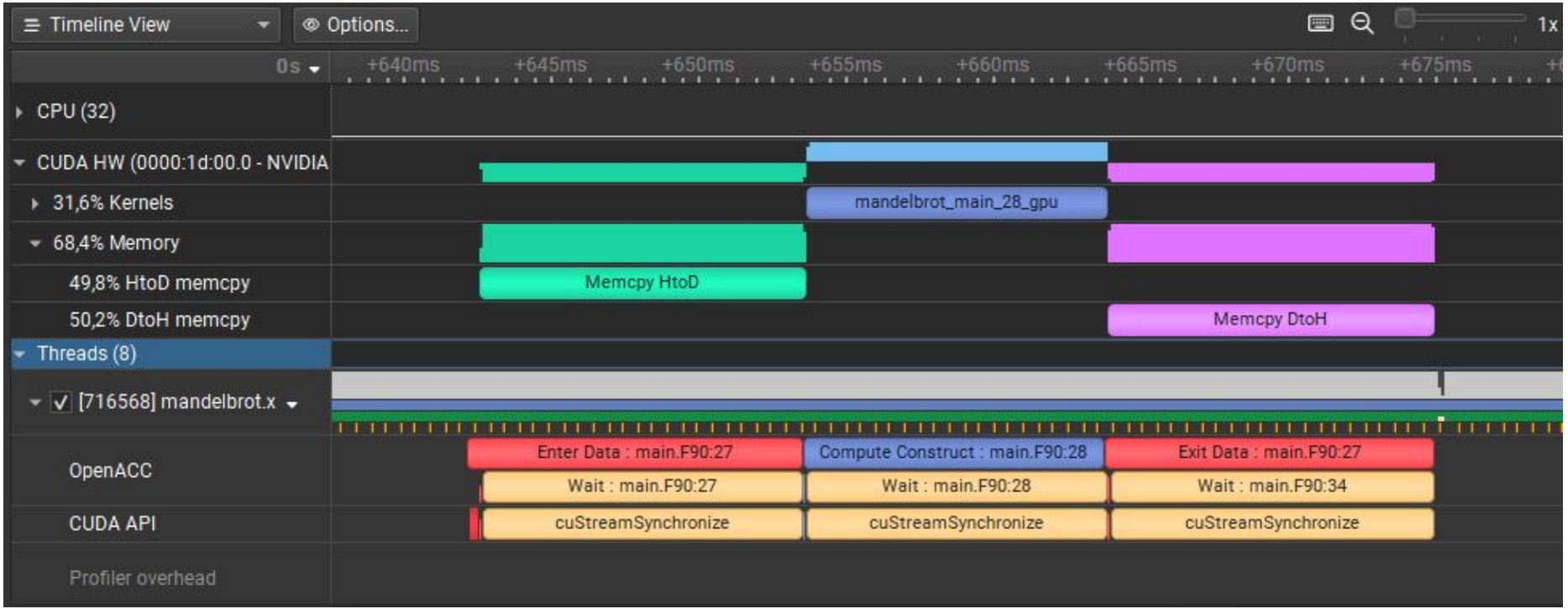
Mandelbrot example

```
!$acc data copy(image(1:HEIGHT, 1:WIDTH))
do block=0,(num_blocks-1)
  starty = block  * (WIDTH/NUM_BLOCKS) + 1
  endy   = min(starty + (WIDTH/NUM_BLOCKS), WIDTH)
  !$acc parallel loop
  do iy=starty,endy
    do ix=1,HEIGHT
      image(ix,iy) = min(max(int(mandelbrot(ix-1,iy-1)),0),MAXCOLORS)
    enddo
  enddo
enddo
!$acc end data
```

```
do while(((x*x+y*y).lt.4.0d0).and.(i.lt.MAX_ITERS))
   xtemp=x*x - y*y + x0
   y=2*x*y + y0
   x=xtemp
   i = i+1
enddo
mandelbrot =  dble(MAXCOLORS)*i/MAX_ITERS
```

- each block is a kernel launched on the GPU
- each unit of work executes and independent “mandelbrot”
- data is already optimized

Mandelbrot example



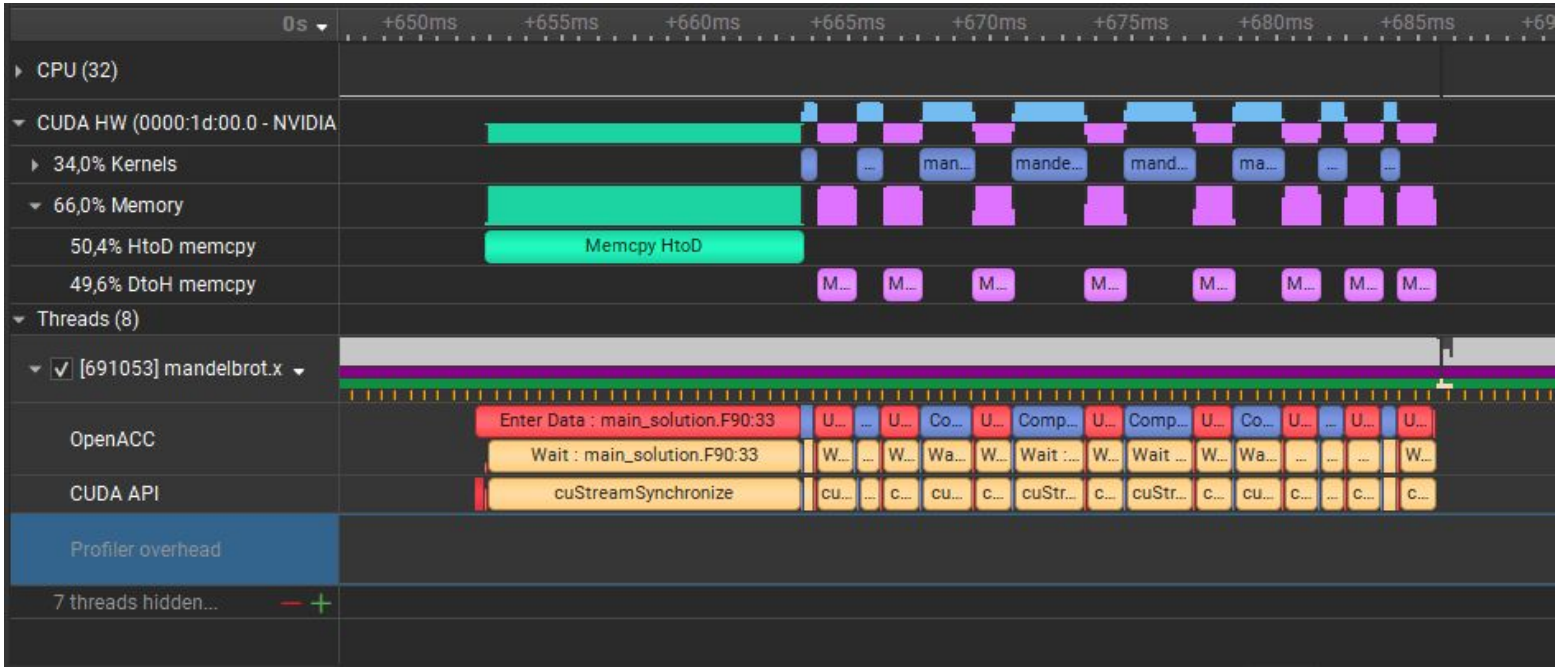
Mandelbrot example

```
!$acc data create(image(1:HEIGHT, 1:WIDTH))
do block=0,(num_blocks-1)
  starty = block  * (WIDTH/NUM_BLOCKS) + 1
  endy   = min(starty + (WIDTH/NUM_BLOCKS), WIDTH)
  !$acc parallel loop
  do iy=starty,endy
    do ix=1,HEIGHT
      image(ix,iy) = min(max(int(mandelbrot(ix-1,iy-1)),0),MAXCOLORS)
    enddo
  enddo
  !$acc update self(image(:,starty:endy))
enddo
!$acc end data
```

```
do while(((x*x+y*y).lt.4.0d0).and.(i.lt.MAX_ITERS))
   xtemp=x*x - y*y + x0
   y=2*x*y + y0
   x=xtemp
   i = i+1
enddo
mandelbrot =  dble(MAXCOLORS)*i/MAX_ITERS
```

- each block is a kernel launched on the GPU
- each unit of work executes and independent “mandelbrot”
- use update instead

Mandelbrot example



about 22 ms for 8 blocks

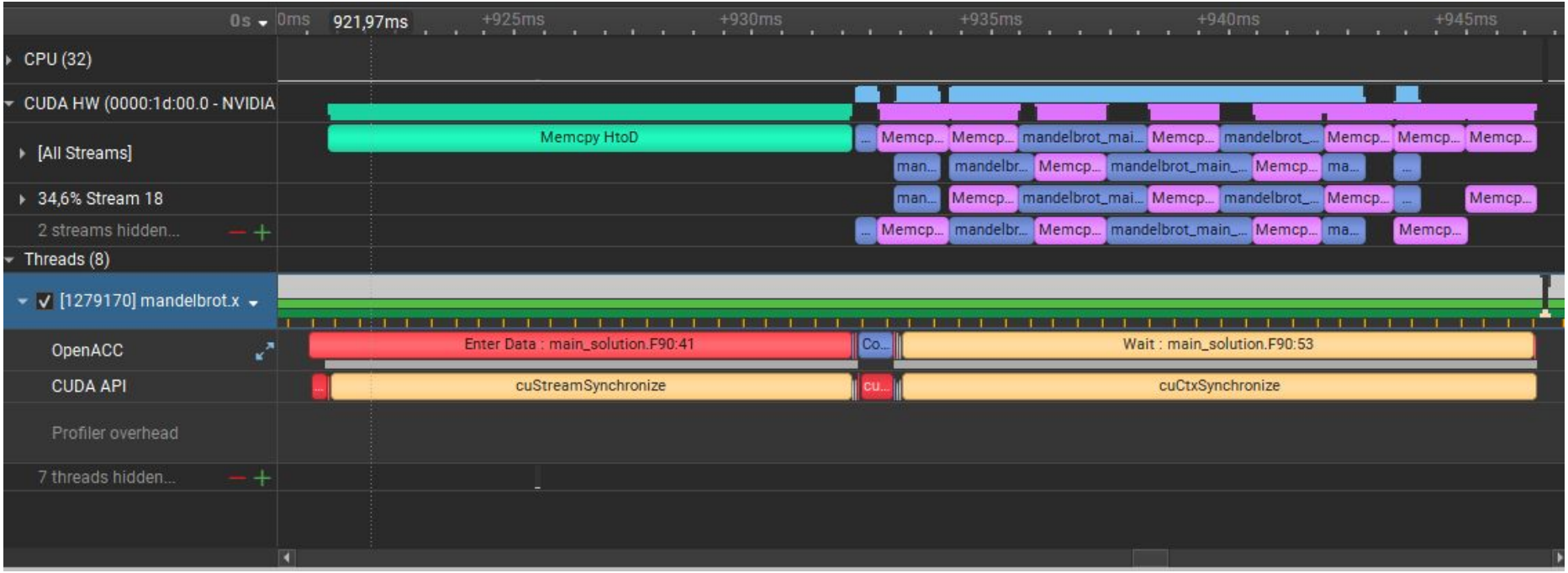
Mandelbrot example

```
!$acc data create(image(1:HEIGHT, 1:WIDTH))
do block=0,(num_blocks-1)
  starty = block  * (WIDTH/num_blocks) + 1
  endy   = min(starty + (WIDTH/num_blocks), WIDTH)
  !$acc parallel loop async(mod(block,2))
  do iy=starty,endy
    do ix=1,HEIGHT
      image(ix,iy) = min(max(int(mandelbrot(ix-1,iy-1)),0),MAXCOLORS)
    enddo
  enddo
  !$acc update self(image(:,starty:endy)) async(mod(block,2))
enddo
!$acc wait
!$acc end data
```

```
do while(((x*x+y*y).lt.4.0d0).and.(i.lt.MAX_ITERS))
   xtemp=x*x - y*y + x0
   y=2*x*y + y0
   x=xtemp
   i = i+1
enddo
mandelbrot =  dble(MAXCOLORS)*i/MAX_ITERS
```

- each block is a kernel launched on the GPU
- it is sent asynchronously to a stream
- each unit of work executes and independent “mandelbrot”
- data update can overlap with computation of the next block on a different stream

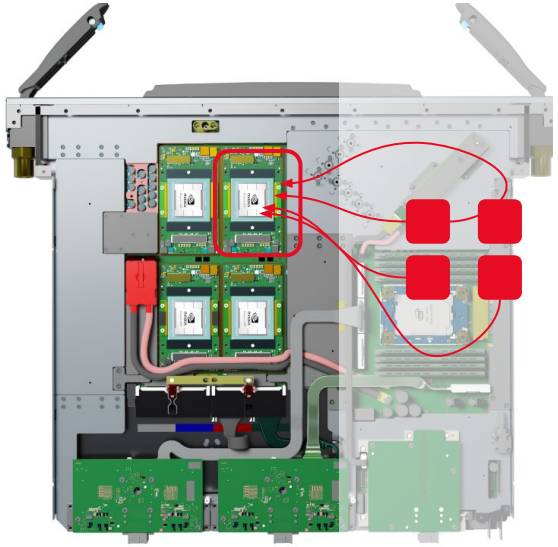
Mandelbrot example



about 14 ms for 8 blocks on 2 streams

Multi process service

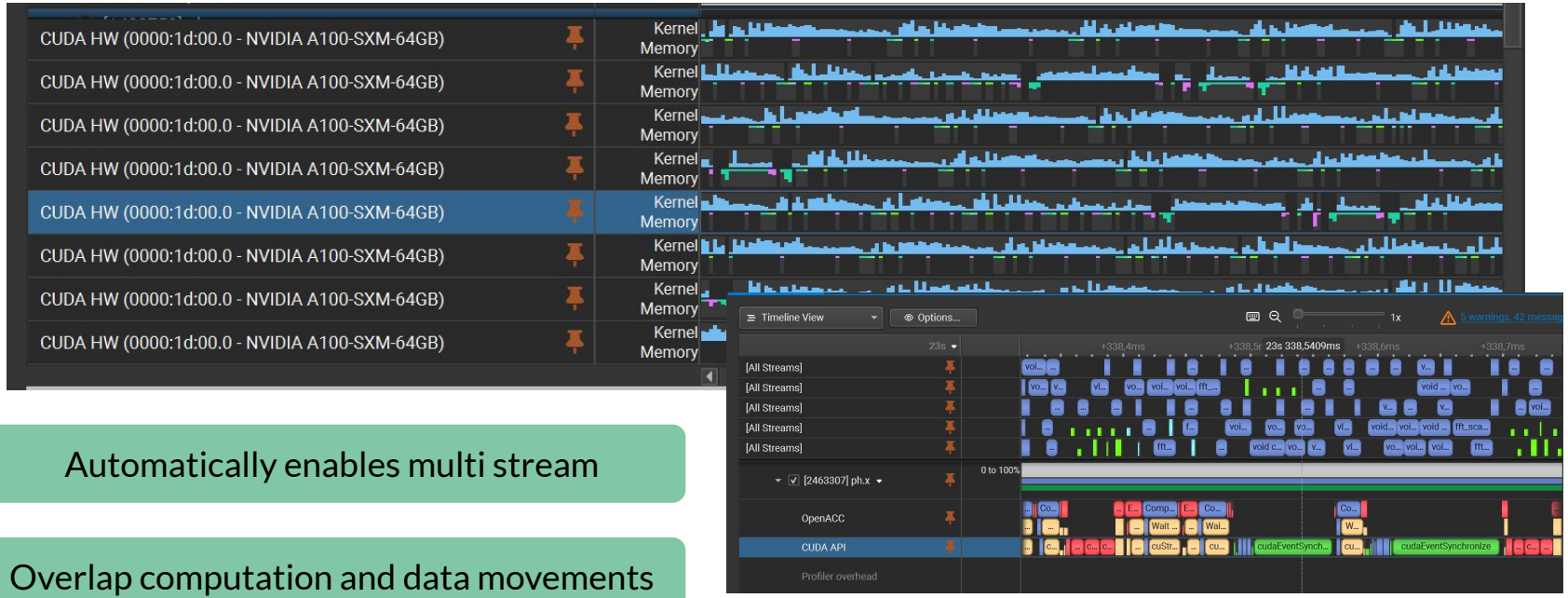
Multi process service to enable cooperative multi-process CUDA applications, typically MPI jobs
Uses Hyper-Q to process concurrently CUDA kernels on the same GPU



```
#SLURM -N 1
#SLURM cpus-per-task=1
#SLURM tasks-per-node=8
#SLURM gres=gpu:1

#This must be different for each node
export CUDA_MPS_LOG_DIRECTORY=./pipe0
export CUDA_MPS_PIPE_DIRECTORY=./pipe0

nvidia-cuda-mps-control -d
mpirun -np 8 --map-by node:PE=1 -rank-by core pw.x

echo quit | nvidia-cuda-mps-control
```

Multi process service

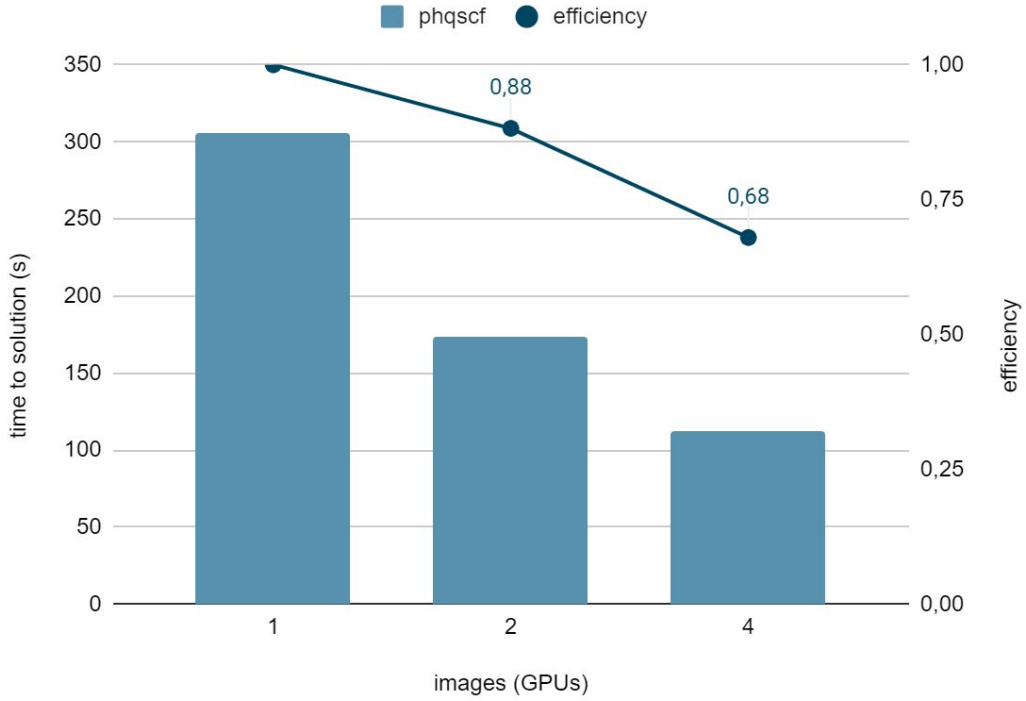
Quartz nat = 9



Automatically enables multi stream

Overlap computation and data movements

Score-P for heterogenous programs



PHonon simulation from QuantumESPRESSO distributed by “images”

- MPI, OpenACC and CUDA
- 1 MPI : 1 GPU
- 1 MPI task per image
- images implement fairly independent computations

Why efficiency is low at 4 GPUs?

Score-P for heterogenous programs

- Code instrumented with Score-P (source-code instrumentation)

```
SCOREP_WRAPPER=off cmake <cmake-options> <source-path>
make SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--mpp=mpi --openacc --cuda" <executable>
```

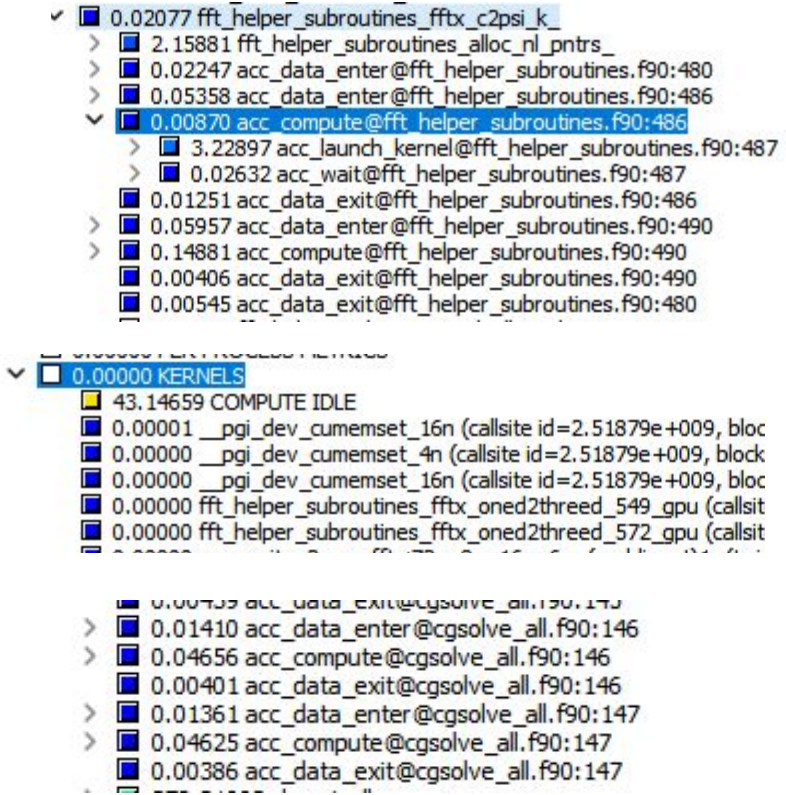
- Measurement at runtime; environment variables to configure it

```
export SCOREP_CUDA_ENABLE=yes
export SCOREP_OPENACC_ENABLE=yes
[. . .]
srun -n <task-number> ./myapp.exe # generates scorep_ folder with profiling data
```

- More at

  https://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-6.0/html/scorepmeasurementconfig.html

Score-P workflows



Time spent in OpenACC region

- launching kernels
- waiting for GPU compute
- implicit and explicit data movement

Identify expensive kernel launches

Identify expensive data movements

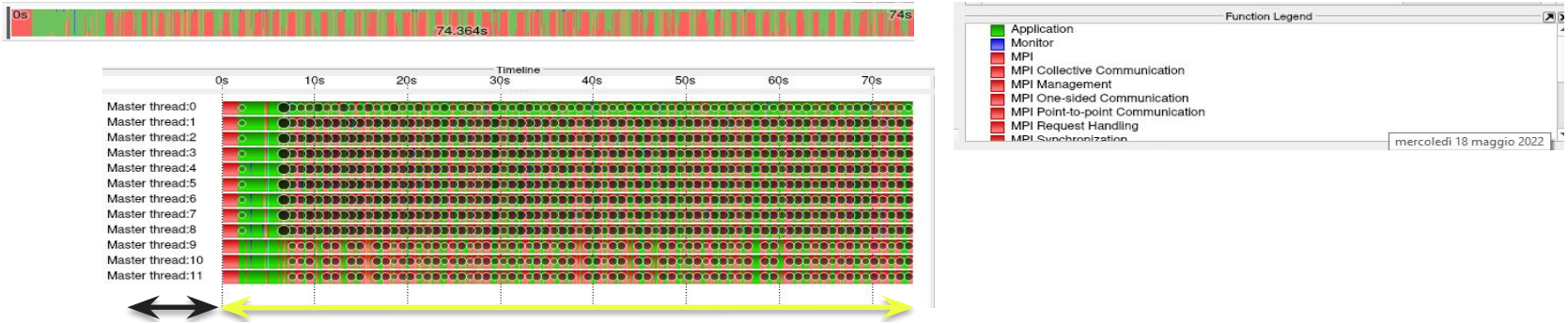
From the GPU perspective

- time spent in GPU kernels
- time spent IDLE state

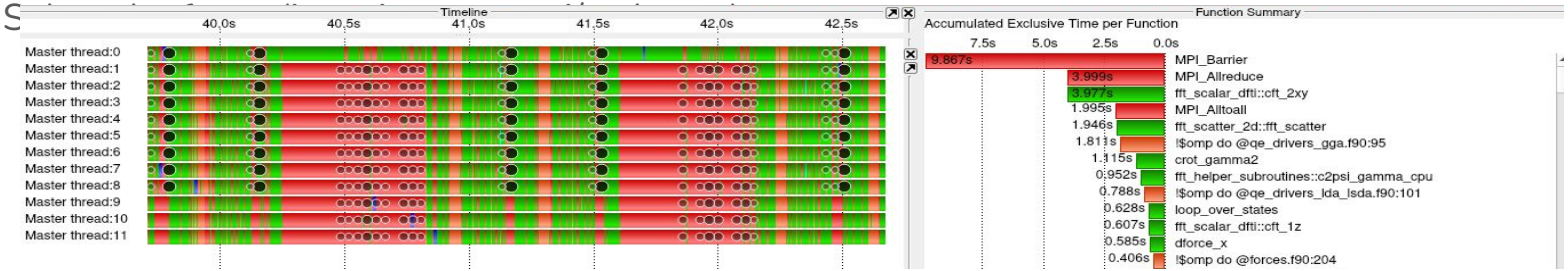
Vampir

```
vampir scorep_12x1_trace/traces.otf2
```

(1) Identify the pattern: initialization, iteration, finalization



(2) S



Vampir

```
vampir scorep_12x1_trace/traces.otf2
```

(3) Visualize communication patterns, durations, data movement (CPU-GPU), CUDA kernels on event timeline