GPU-Based Real-Time SAS Processing On-Board Autonomous Underwater Vehicles

Jesús Ortiz
Advanced Robotics Department (ADVR)
Istituto Italiano di tecnologia (IIT)

Francesco Baralli
Center for Maritime Research and Experimentation (CMRE)
NATO Science and Technology Organization (STO)

CMRE@NATO STO Overview

NATO Science and Technology Organization

Center for Maritime Research and Experimentation

- Within the framework of the STO, CMRE organizes and conducts scientific research and technology development, and delivers innovative and field-tested S&T solutions to address the defense and security needs of the Alliance.
- The CMRE is customer funded. Customers will be comprised of NATO bodies, NATO Nations and other parties, consistent with NATO policy.
- The mission of the CMRE is centered on the maritime domain, but may extrapolate into other domains to meet customers' demands.
- CMRE maintains a set of core technical capabilities, including: underwater acoustics, sensors and signal processing, autonomy, ocean engineering, seagoing capability, and AUVs.

ADVR@IIT Overview

Istituto Italiano di Tecnologia

Advanced Robotics Department

- Innovative and multidisciplinary approach to humanoid design and control, and the development of novel robotic components and technologies:
  - Hardware: mechanical/ electrical design and fabrication, sensor systems, actuation development, etc.
  - Software: control, computer software, human factors, etc.
- Transition from traditional (hard-bodied) robots towards a new generation of hybrid systems (biologically inspired concepts: muscle, bone, tendon, skin…)
- Research activities:
  - core scientific/technological research aimed at providing fundamental competencies needed to develop robotic and humanoid technology
  - advanced research demonstrators that provide large focused research projects integrating the core sciences

Autonomous Underwater Vehicle (AUV)

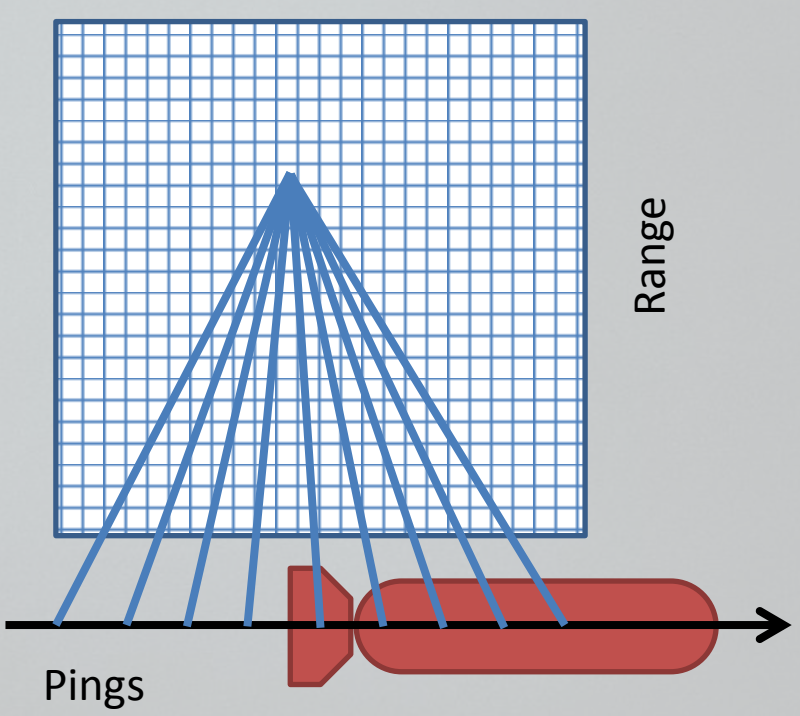
- Basic:
  - Perform underwater missions following pre-programmed path
  - No human operator in the control loop
  - Carry different sets of environmental and/or imaging sensors
  - Communicate with surface station and with other agents

- Advanced
  - Real-time data processing and analysis
  - High level of autonomy: adaptive behavior
  - Cooperative behavior among different AUVs

Sonar Imaging

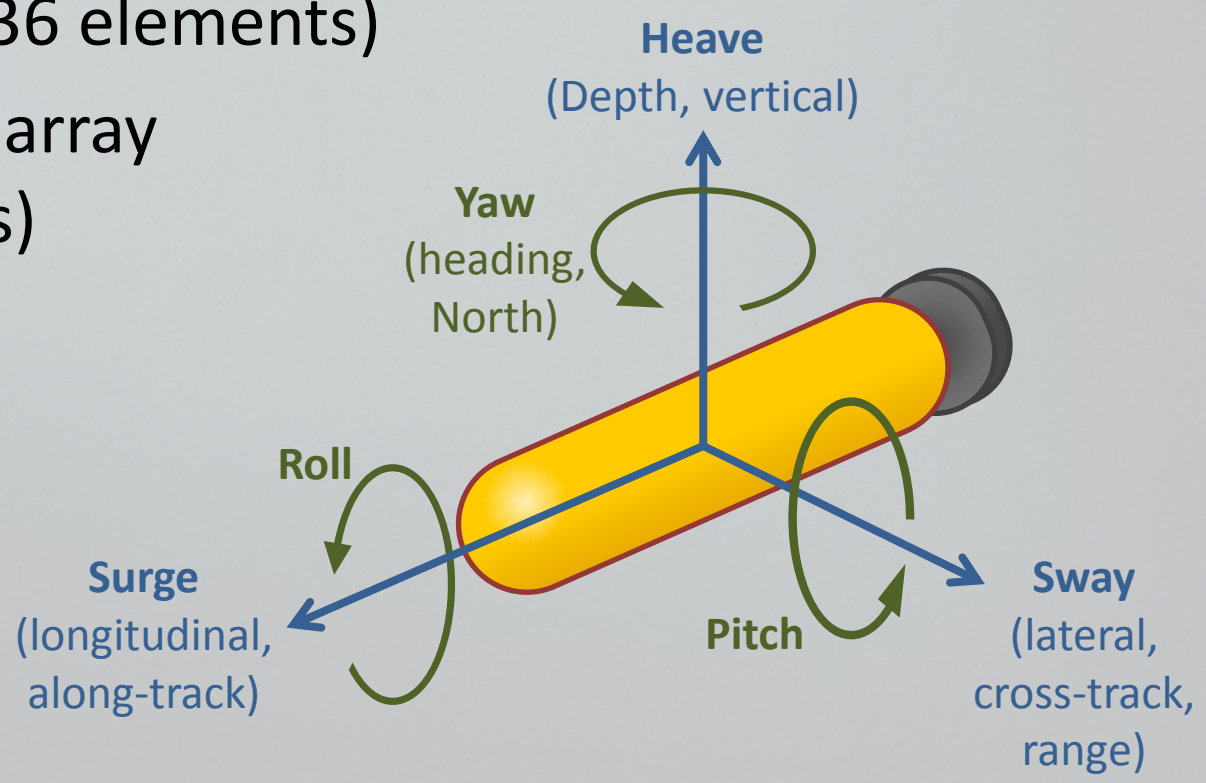
Synthetic Aperture Sonar (SAS):

- Multiple pulses to create a large synthetic array (aperture)
- One pixel of the image is produced using information from multiple pulses
- Multistatic approach (one emitter, multiple receivers)
- The emitted ping is a “chirp” (broad frequency band)

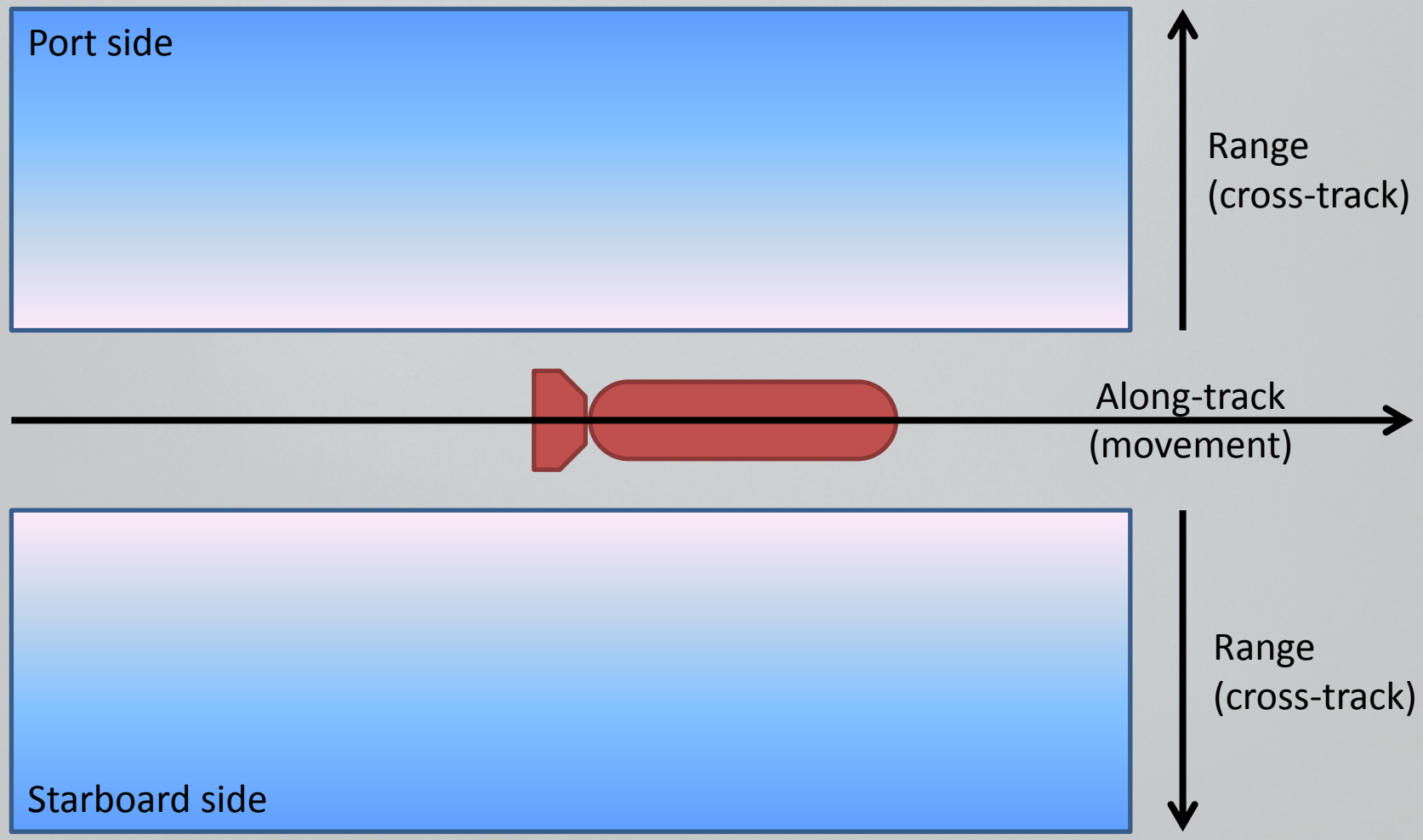


AUV

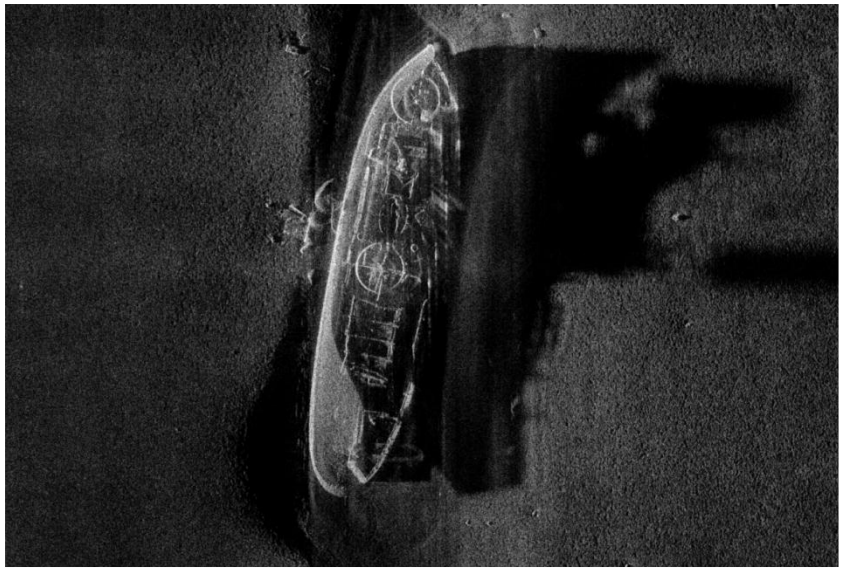
- Sonar on both sides (port & starboard)
- Two receiver arrays per side:
  - Main array (36 elements)
  - Bathymetric array (12 elements)

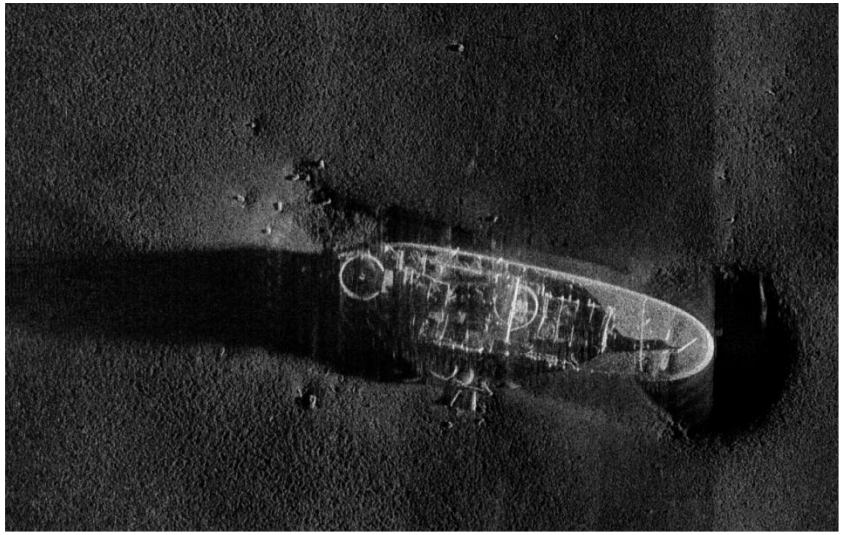


Synthetic Aperture Sonar (SAS)



SAS result





SAS processing

- Preprocessing (calibration & matching)
- Displaced Phase Center Antenna (DPCA)
- Navigation Integration
- SAS imaging
- SAS Interferometry → 3D images

Automatic Target Recognition (ATR)

Replanning

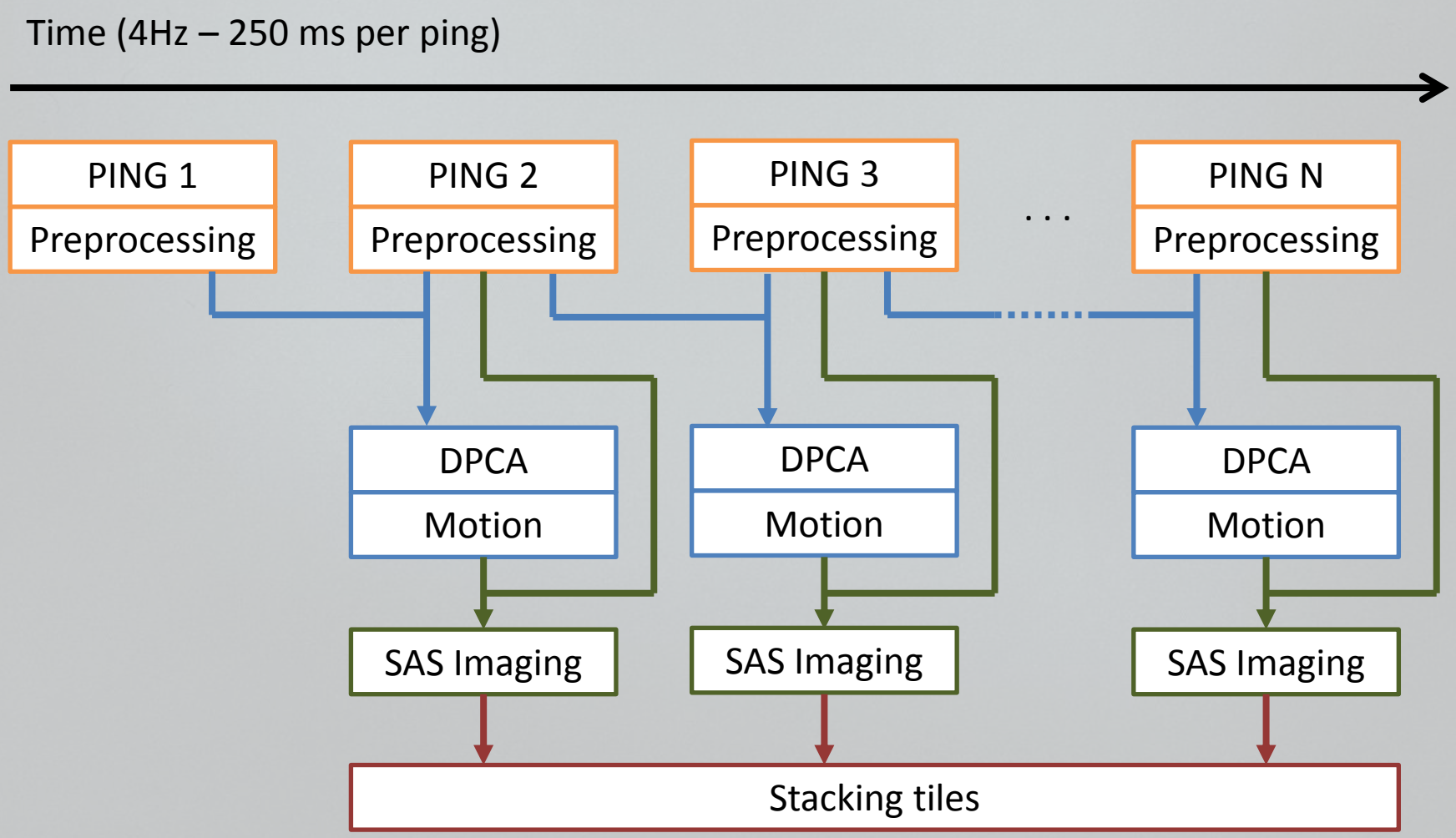
SAS processing

- Preprocessing (calibration & matching)
- Displaced Phase Center Antenna (DPCA)
- Navigation Integration
- SAS imaging
- SAS Interferometry → 3D images

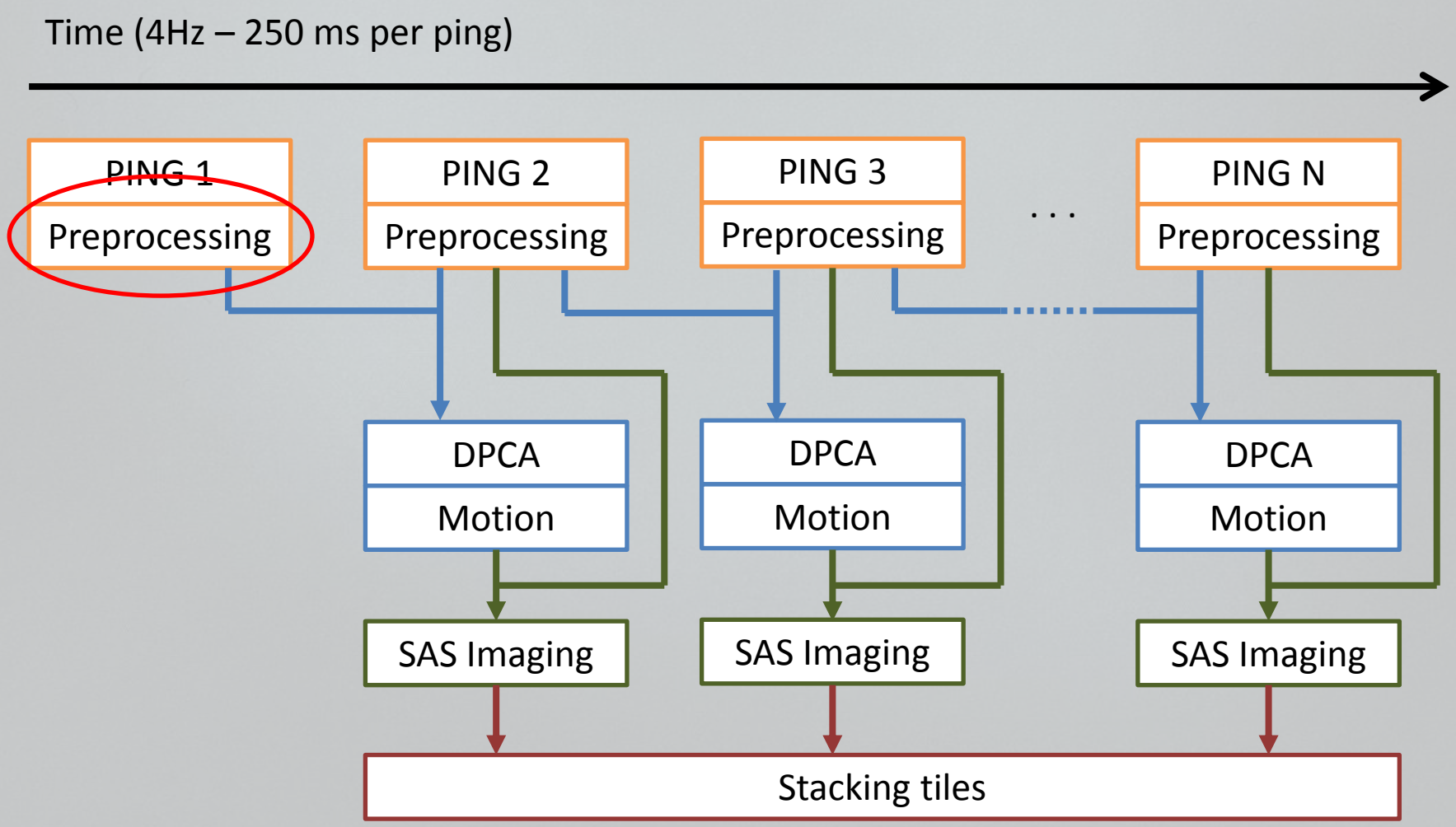
Automatic Target Recognition (ATR)

Replanning

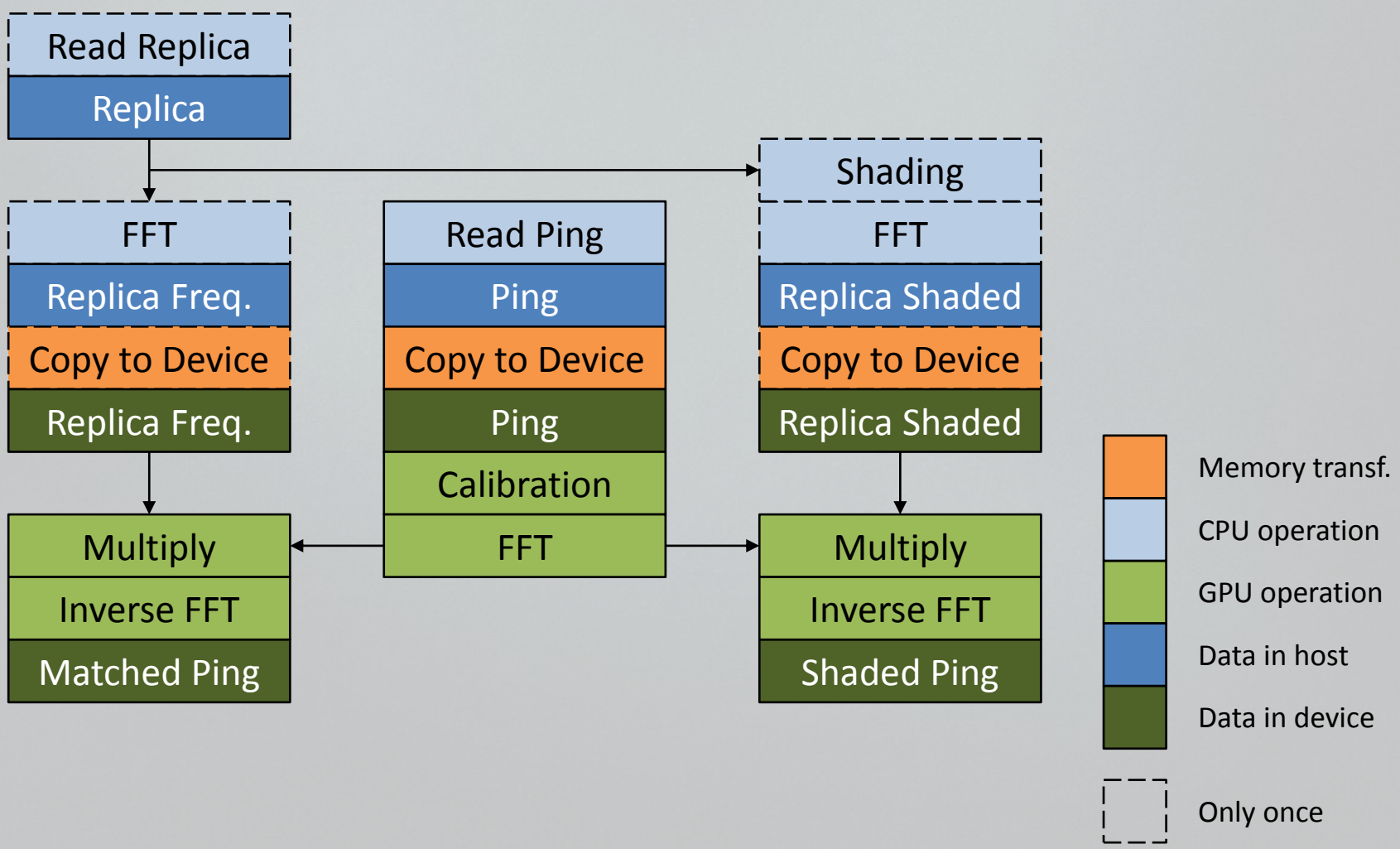
SAS processing



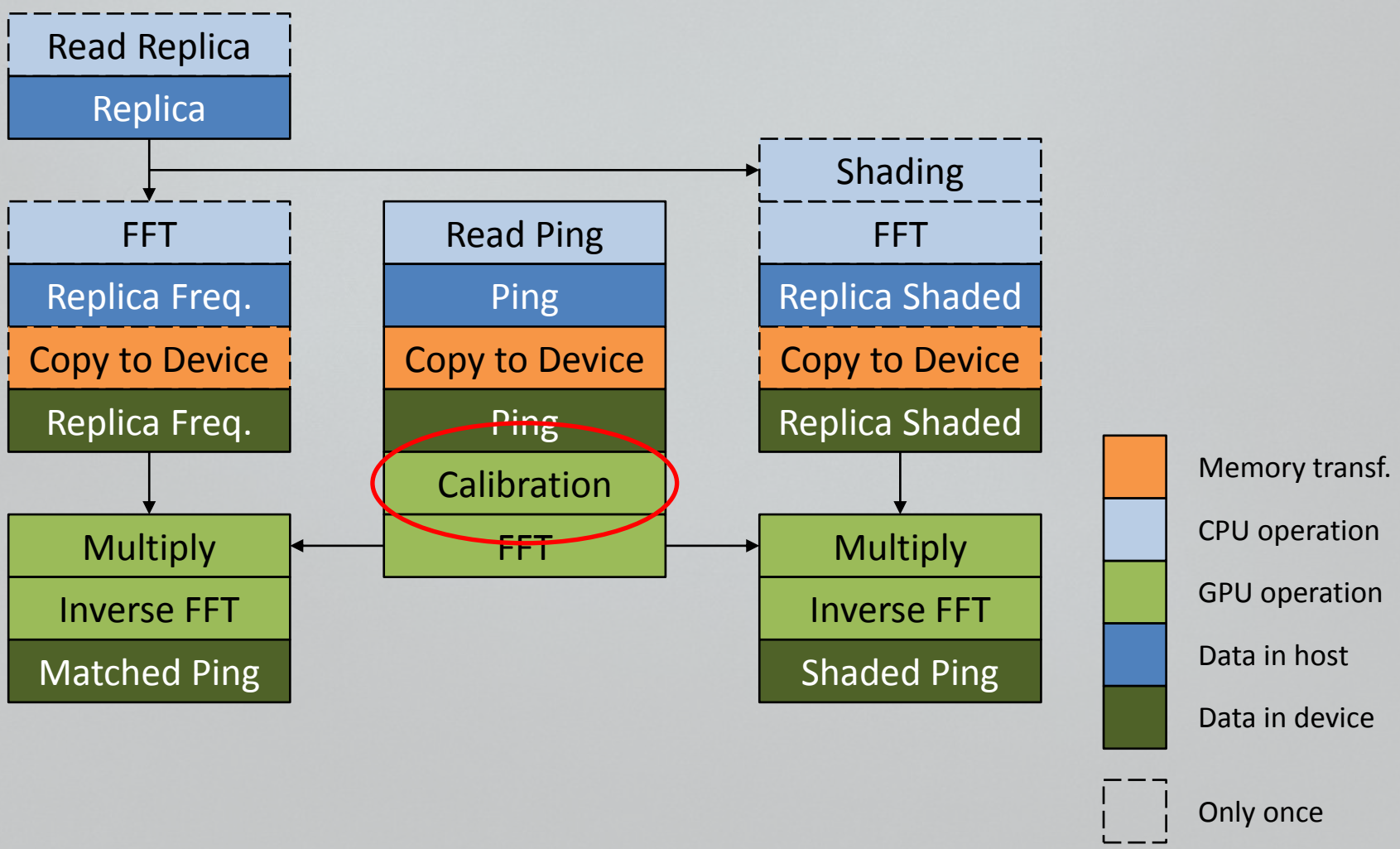
SAS processing



Preprocessing



Preprocessing



Preprocessing - Calibration

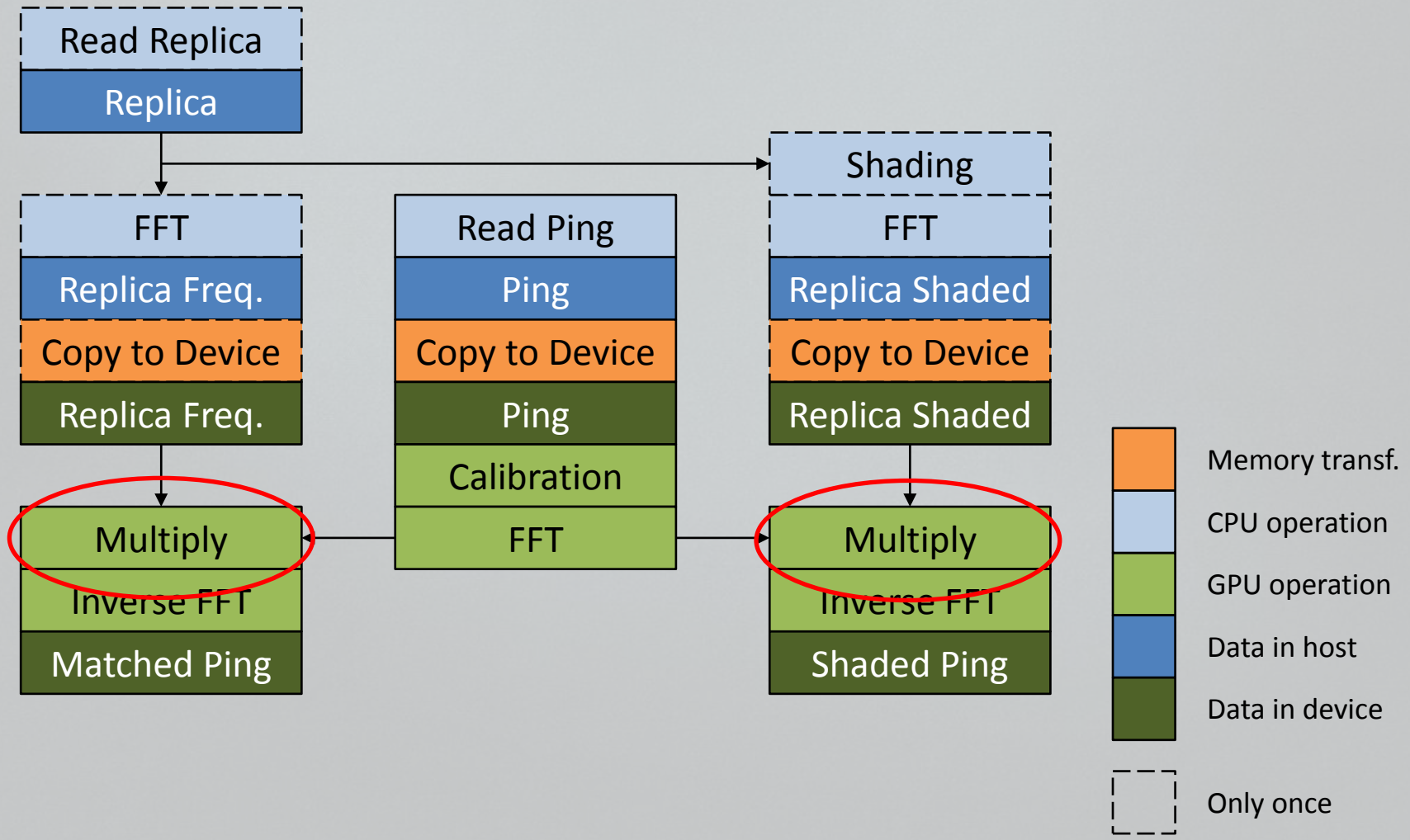
- Calibrate acoustic data
  - Adjust the phase of each receiver due to wiring and reading delays

$$A_c = A_r \cdot C \Rightarrow \begin{cases} A_c^r = A_r^r \cdot C^r - A_r^i \cdot C^i \\ A_c^i = A_r^r \cdot C^i + A_r^i \cdot C^r \end{cases}$$

where:

- A_c is the calibrated Acoustic data (complex)
- A_r is the raw Acoustic data (complex)
- C is the Calibration (complex of modulus 1)

Preprocessing



Preprocessing - Matching

- Match acoustic data with the replica
  - Unshaded replica → For DPCA
  - Shaded replica → For Imaging

$$A_{mu} = A_c \cdot R_u$$

$$A_{ms} = A_c \cdot R_s$$

where:

- A_{mu} is the Unshaded Matched Acoustic data (complex)
- A_{ms} is the Shaded Matched Acoustic data (complex)
- R_u is the Unshaded Replica (complex)
- R_s is the Shaded Replica (complex)

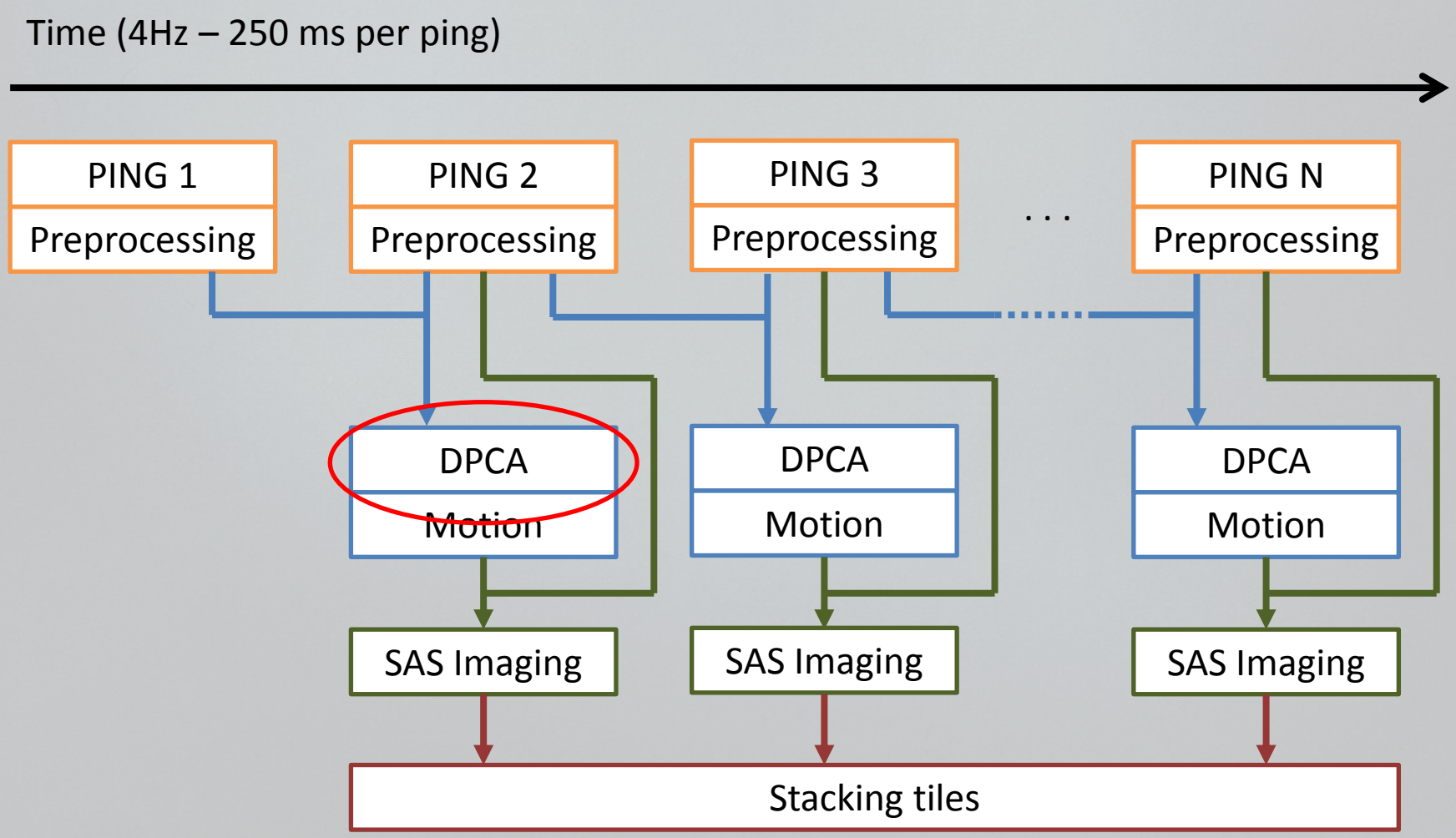
Preprocessing - Results

- Calibration done in place to reduce the memory requirements
- FFT with FFTW (CPU) and cuFFT (GPU)

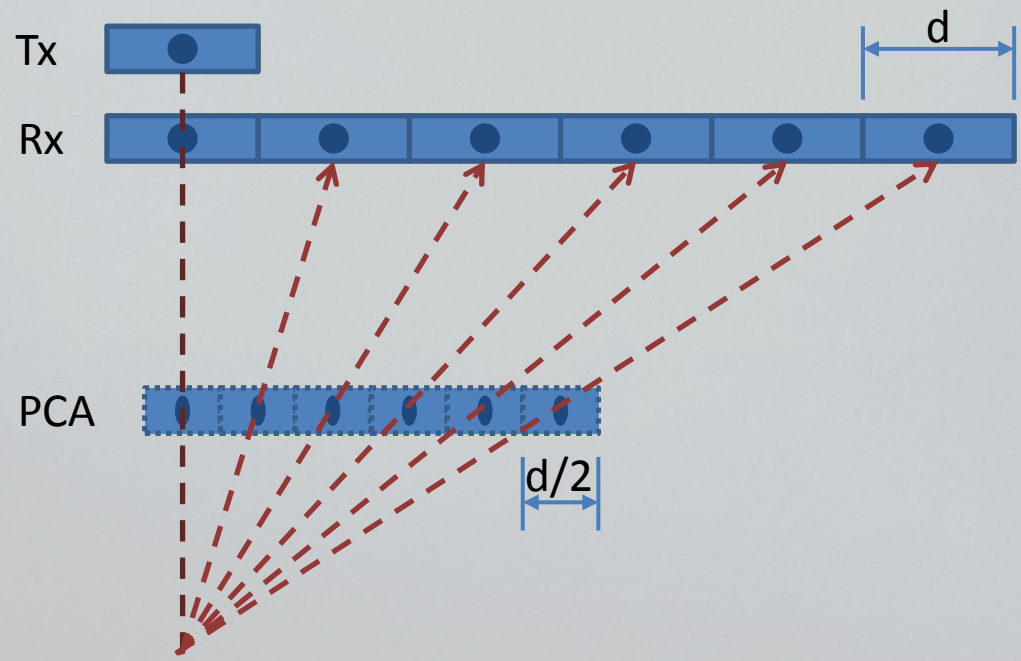
CPU[1]	GPU[2]	Speedup
84.3 ms	3.7 ms	x23

[1] Running on an Intel Core i7-920 @ 2.67 GHz
[2] Running on a NVIDIA Tesla C1060

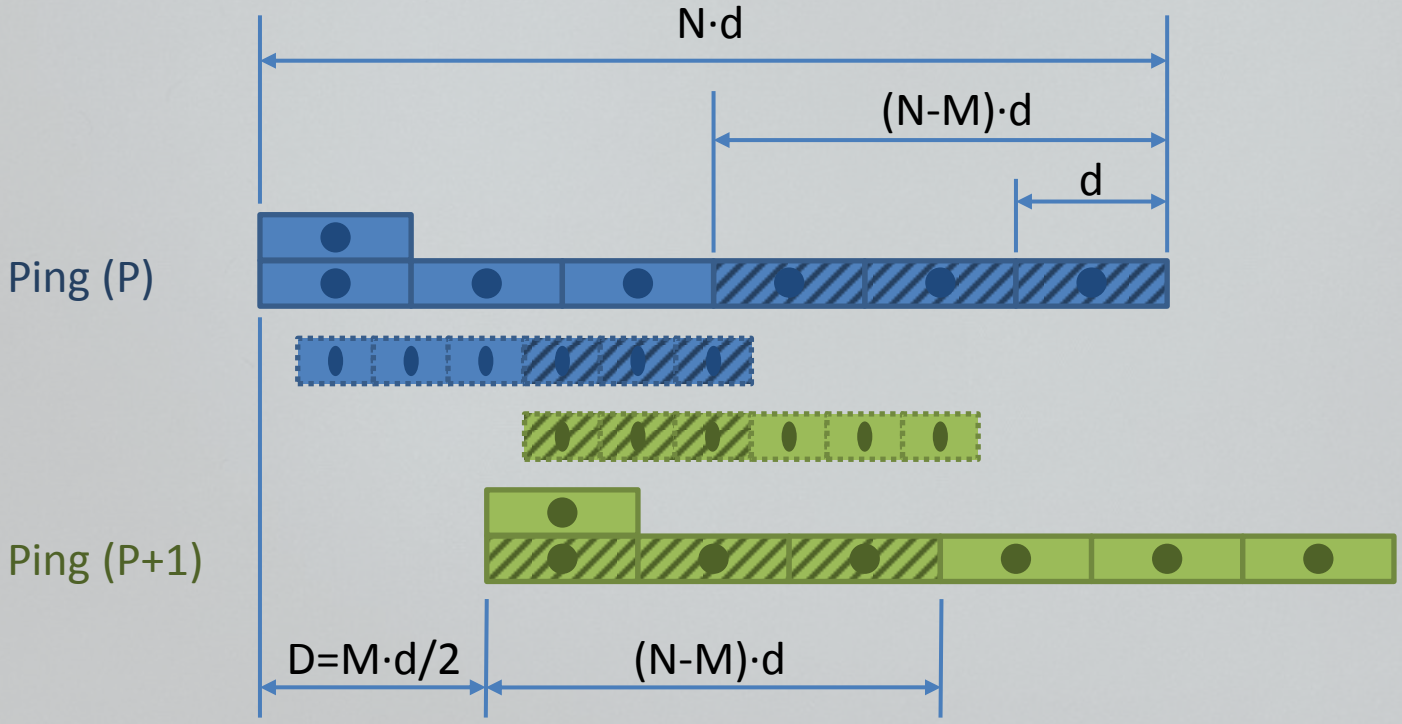
SAS processing



Phase Center Approximation



DPCA concept

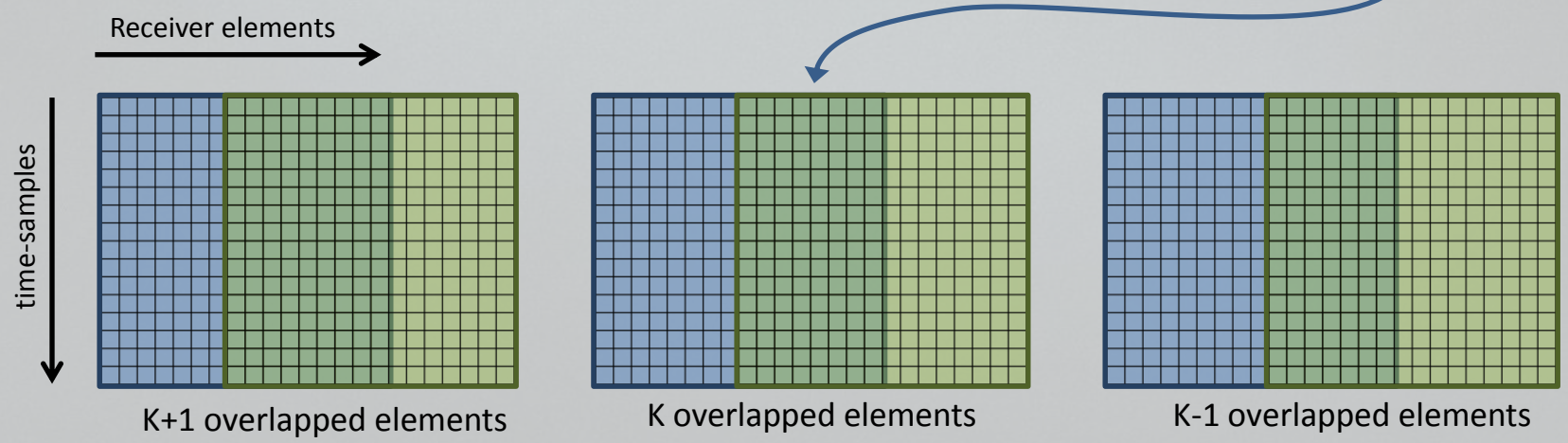


D → AUV displacement
d → Separation elements
N → Number of elements
K=(N-M) → Number of overlapped elements

R. Heremans, A. Bellettini and M. Pinto, "Milestone: Displaced Phase Center Array", September 2006, http://www.sic.rma.ac.be/~rhereman/milestones/dpca.pdf

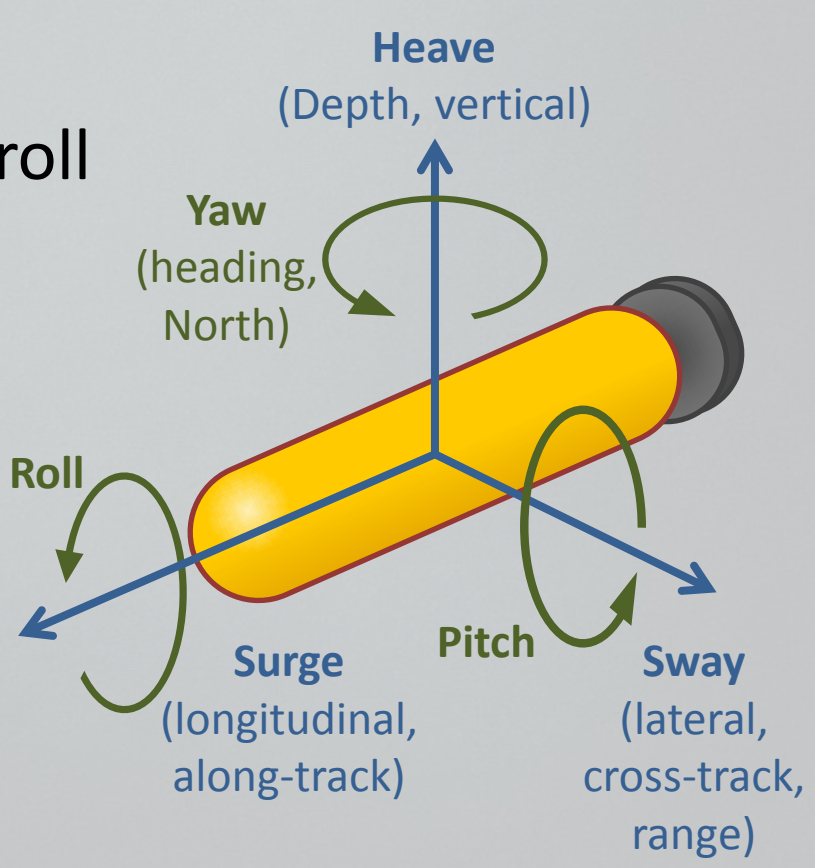
DPCA calculation

- Find best correlation between consecutive pings
  - Use INS for first approximation of surge
  - Get best correlation → surge correction
  - Get delay → sway calculation
  - Use INS for other variables (roll, pitch, yaw)



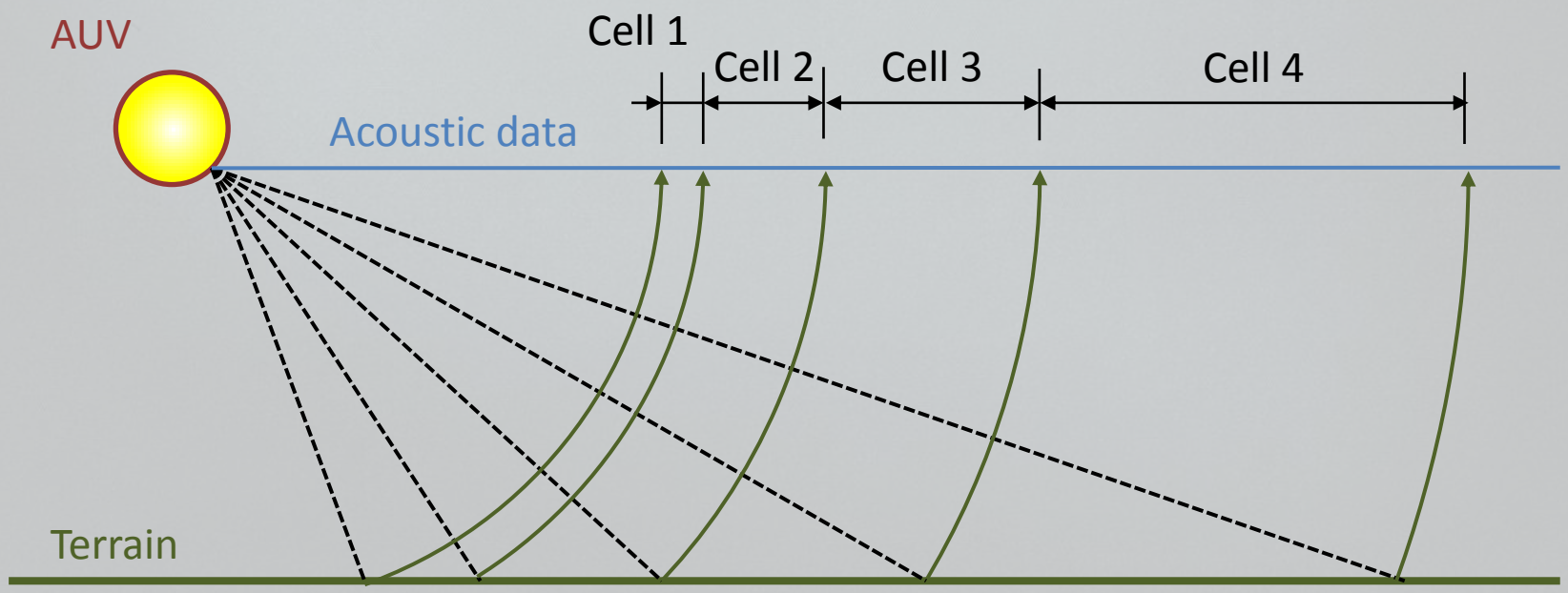
Motion Estimation

- INS information:
  - Speed → surge estimation
  - Accelerometer → pitch & roll
  - Compass → yaw
  - Depth → heave
- DPCA:
  - Surge correction
  - Sway calculation

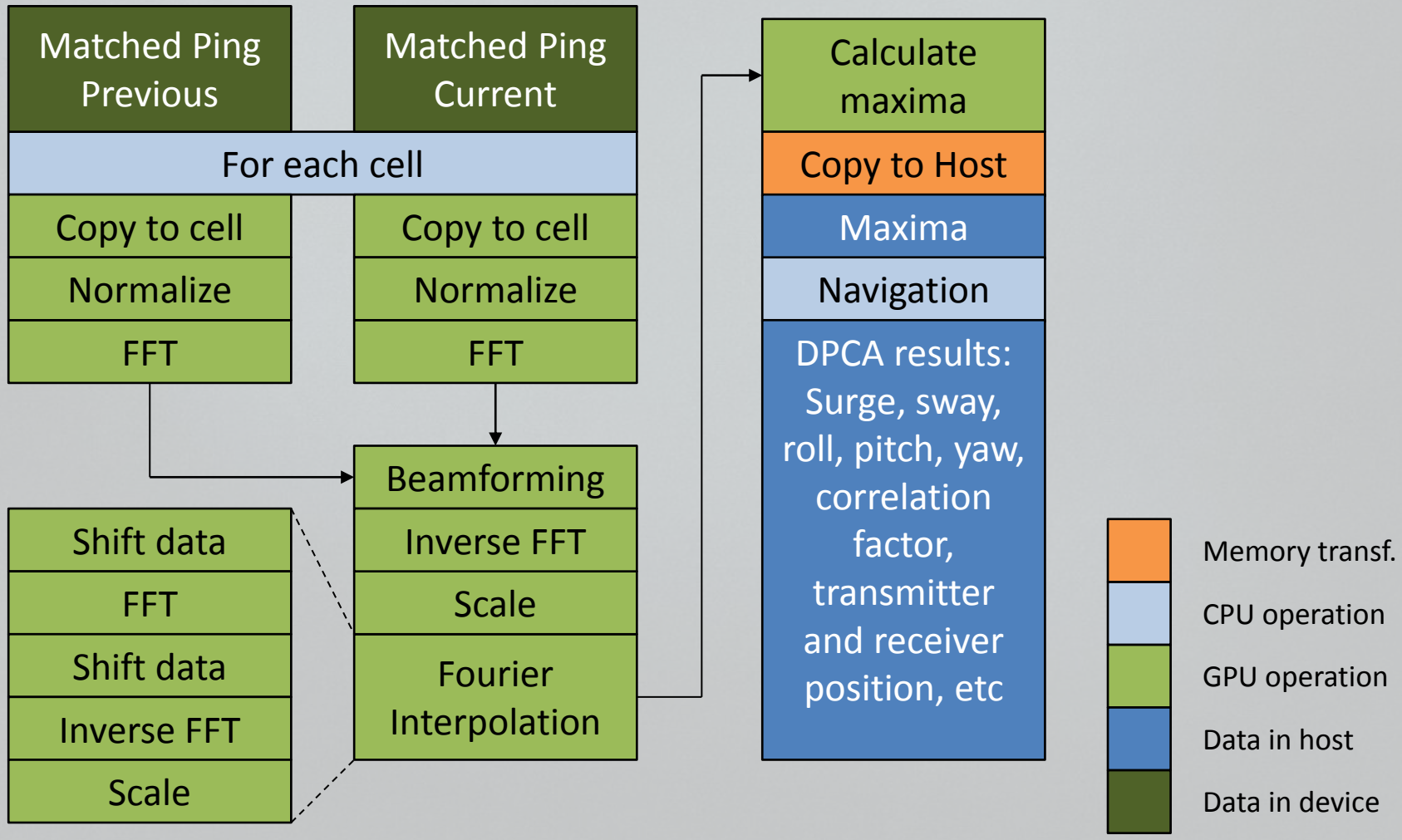


DPCA - Cells

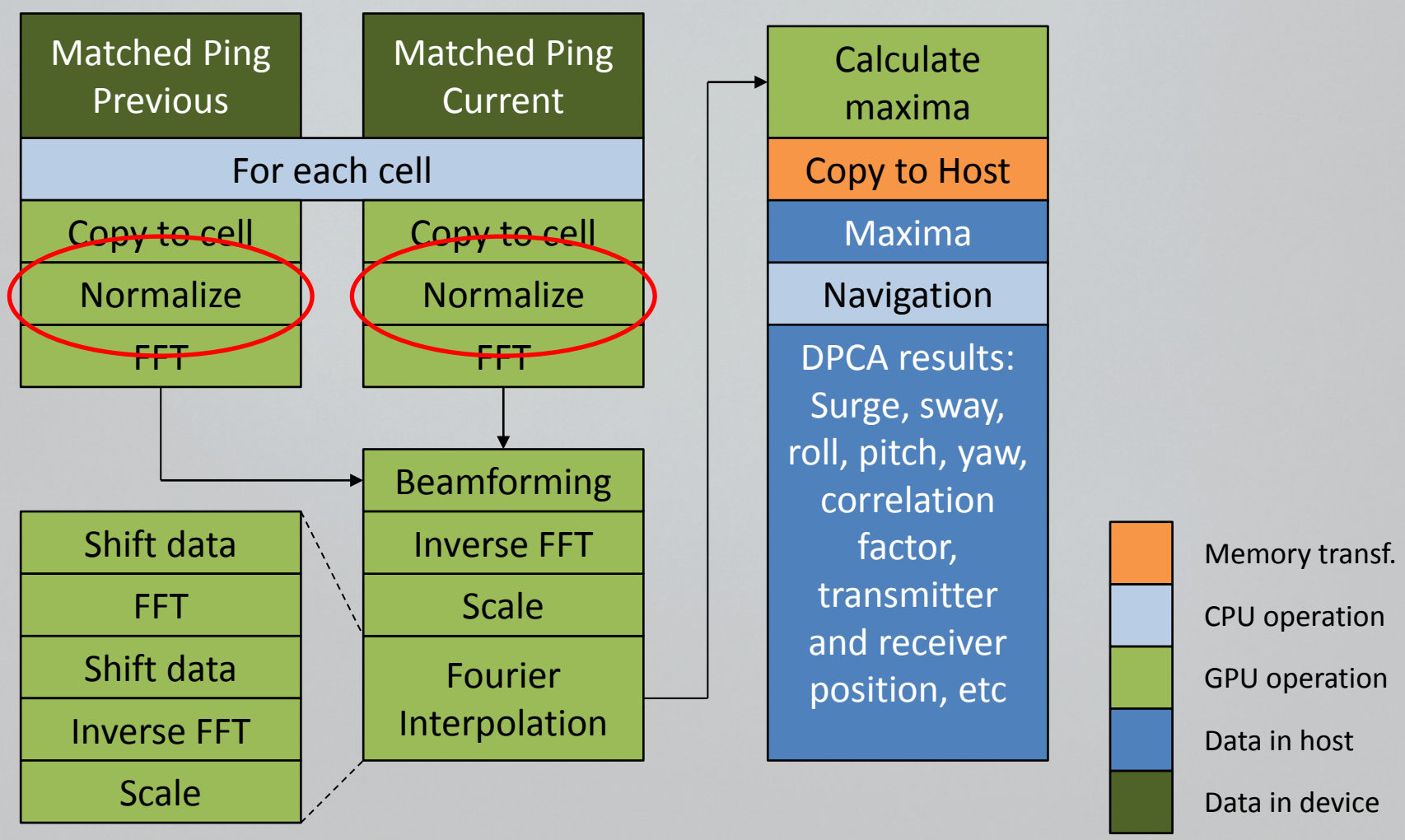
- Divide the acoustic data in cells
- Each cell covers a different range of time/distance



DPCA Processing



DPCA Processing



DPCA - Normalize

- Calculate modulus:

$$modulus = \sqrt{\sum_{i=1}^{N_s} \left(R_i^2 + I_i^2\right)}$$

Where:

- R_i is the real part of the sample i
- I_i is the imaginary part of the sample i
- N_s is the number of samples

- Divide the values by the modulus:

$$R'_i = R_i / modulus \qquad I'_i = I_i / modulus$$

DPCA - Normalize

- Calculate the modulus of each sample
- Each thread calculates more than one sample

$$modulus_t^2 = \sum_{i=t\cdot N_{st}+1}^{(t+1)\cdot N_{st}} \left(R_i^2 + I_i^2\right)$$

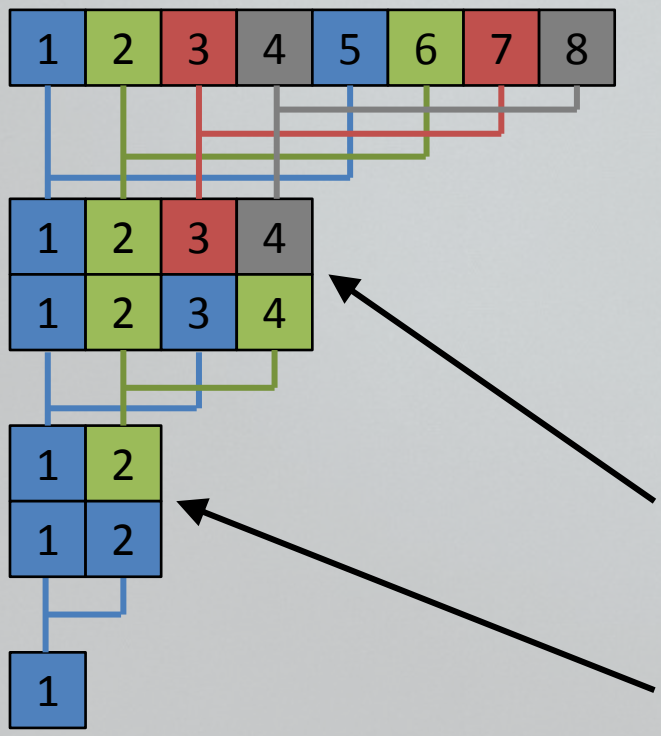
where:

- $modulus_t^2$ is the partial modulus calculated by the thread t
- N_{st} is the number of samples per thread

- Save the partial modulus in the shared memory

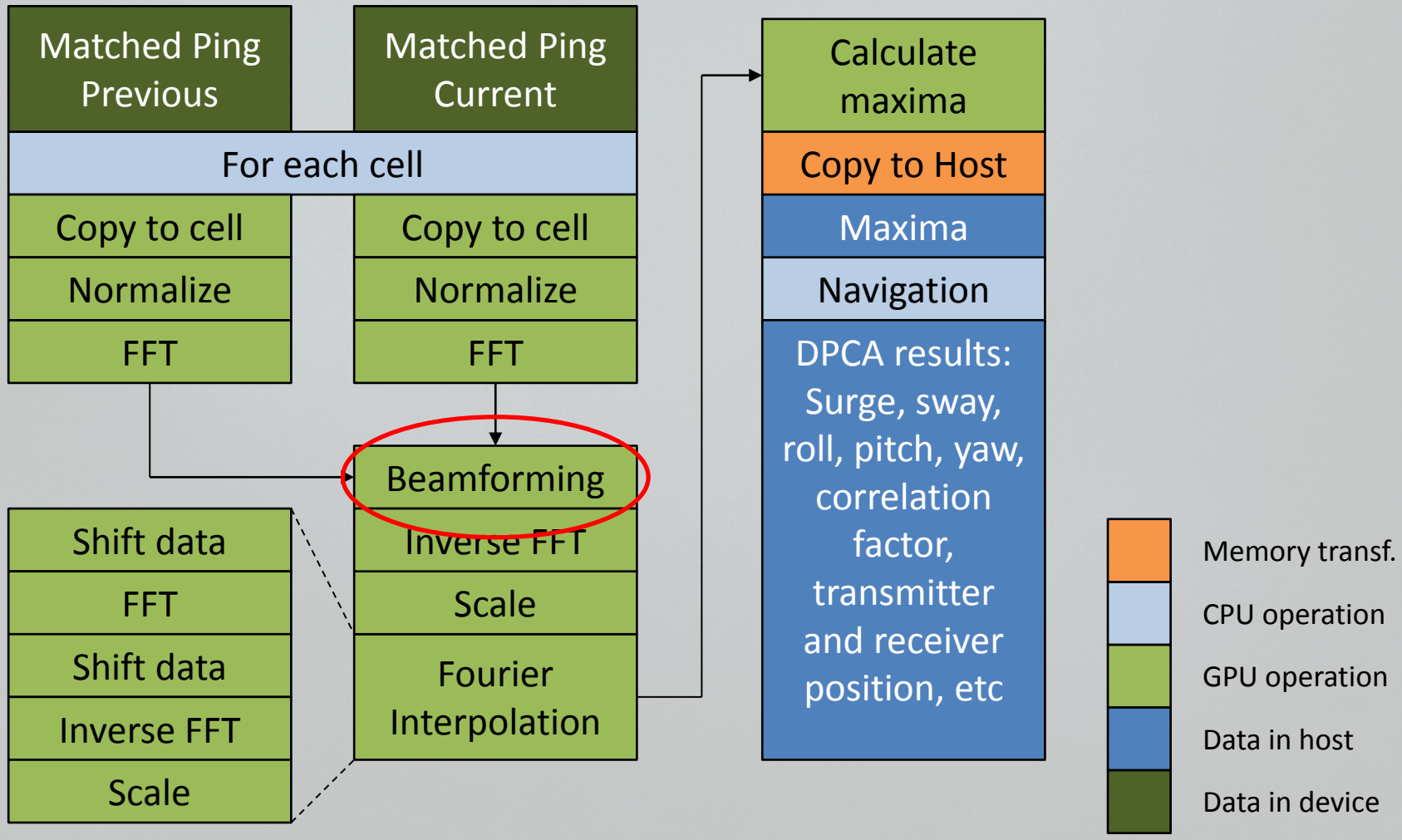
DPCA - Normalize

- Calculate the total modulus using a tree




```
__syncthreads();

if (threadIdx.x < 128)
  modulus[threadIdx.x] += modulus[threadIdx.x + 128];

__syncthreads();

if (threadIdx.x < 64)
  modulus[threadIdx.x] += modulus[threadIdx.x + 64];

__syncthreads();
...
__syncthreads();

if (threadIdx.x < 2)
  modulus[threadIdx.x] += modulus[threadIdx.x + 2];

__syncthreads();

if (threadIdx.x == 0)
  modulus[threadIdx.x] += modulus[threadIdx.x + 1];

__syncthreads();
```

DPCA Processing



DPCA - Beamforming

- Multiply current and previous cell with a beamforming coefficient:

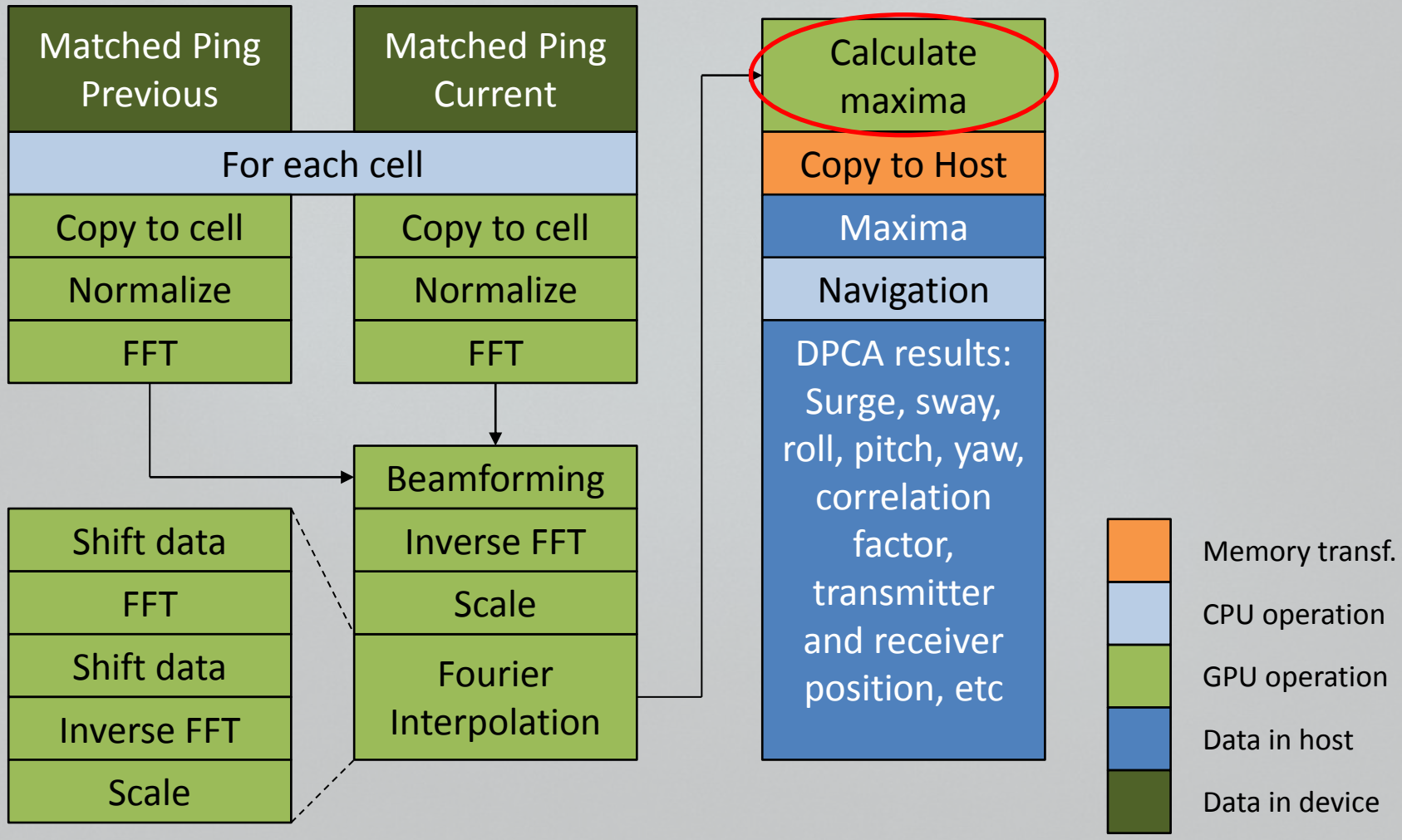
$$correlation_i = A_i^{curr} \cdot \overline{A_{N-K+i}^{prev}}\text{, with } i = 1 \dots K$$

$$bfArray_i = correlation_i \cdot bfCoefficient_i$$

where

- A_i^{curr} is the current acoustic data
- A_i^{prev} is the previous acoustic data
- $correlation_i$ is the correlation between the current and previous
- $bfCoefficient_i$ is the beamforming coefficient
- $bfArray_i$ is the beamforming array

DPCA Processing



DPCA - Maxima

- Find the maximum and use parabolic filtering → We need to store the two neighbors:
  - Three points in total (left, maximum, right)
  - Two values per point (complex values, not enough with modulus)
  - Only position of the maximum
- Each thread calculates a local maximum between a group of N_{st} (number of samples per thread) → Stores modulus of the maximum and the position in shared memory

DPCA - Maximums

- Calculate the maximum using a tree, first thread stores results in global memory

```
if (threadIdx.x < 128)
{
  if (modulus[threadIdx.x] < modulus[threadIdx.x + 128])
  {
    modulus[threadIdx.x] = modulus[threadIdx.x + 128];
    positions[threadIdx.x] = positions[threadIdx.x + 128];
  }
}

__syncthreads();
...
__syncthreads();

if (threadIdx.x < 2)
{
  if (modulus[threadIdx.x] < modulus[threadIdx.x + 2])
  {
    modulus[threadIdx.x] = modulus[threadIdx.x + 2];
    positions[threadIdx.x] = positions[threadIdx.x + 2];
  }
}

__syncthreads();
```

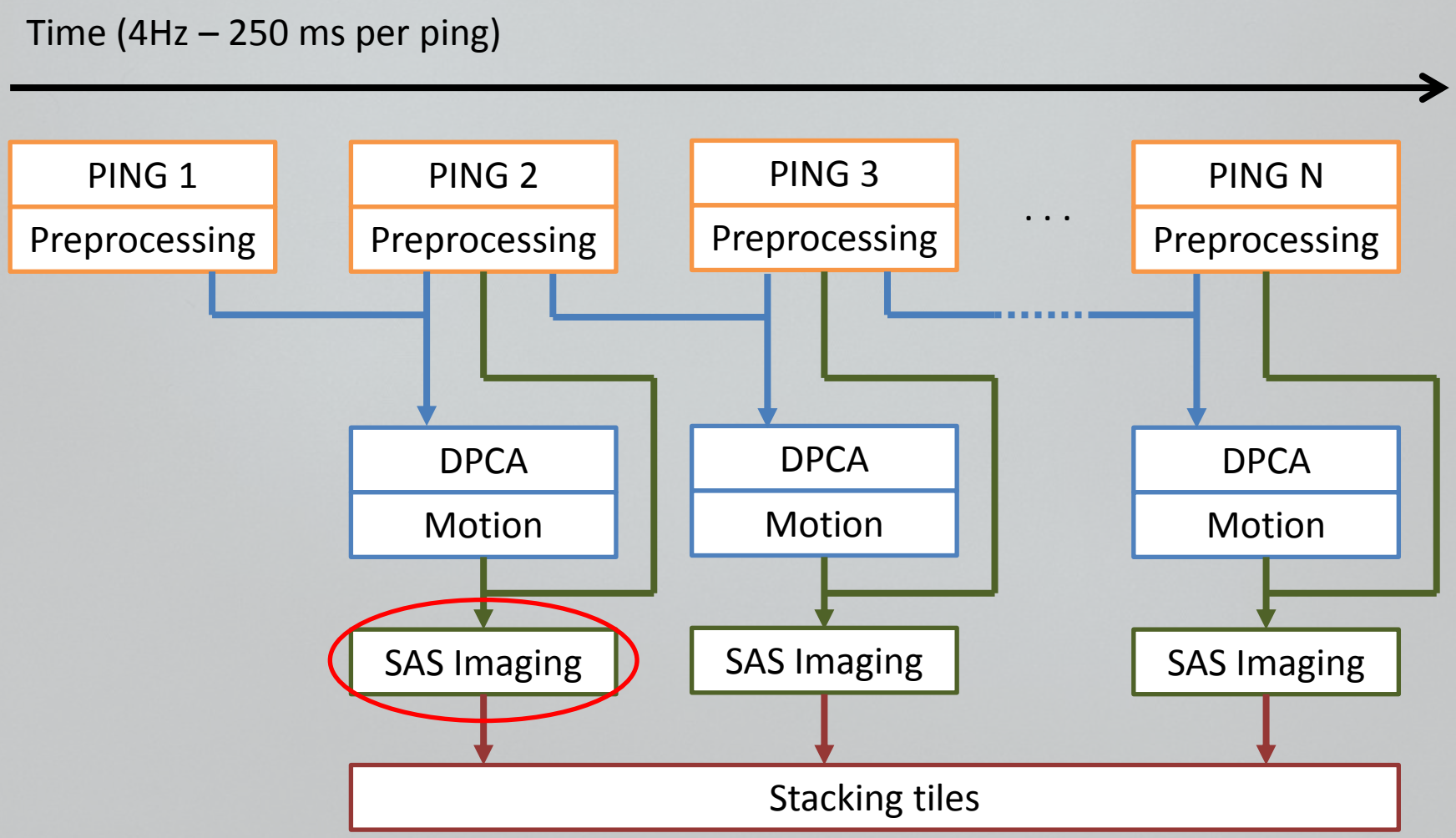
DPCA - Results

- Parabolic filtering, motion calculation, navigation integration always on the CPU
- FFT with FFTW (CPU) and cuFFT (GPU)

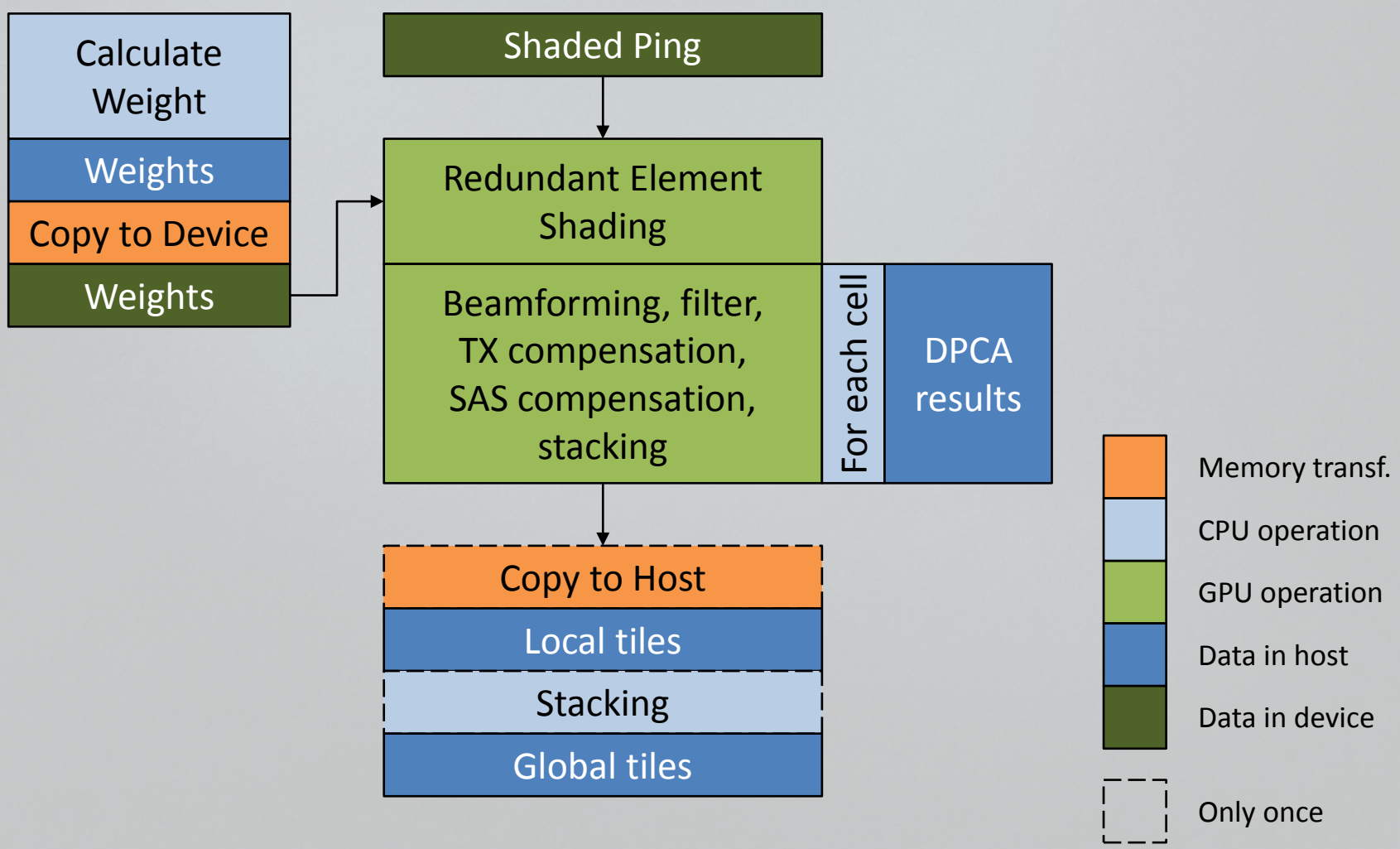
CPU[1]	GPU[2]	Speedup
93.5 ms	18.4 ms	x5

[1] Running on an Intel Core i7-920 @ 2.67 GHz
[2] Running on a NVIDIA Tesla C1060

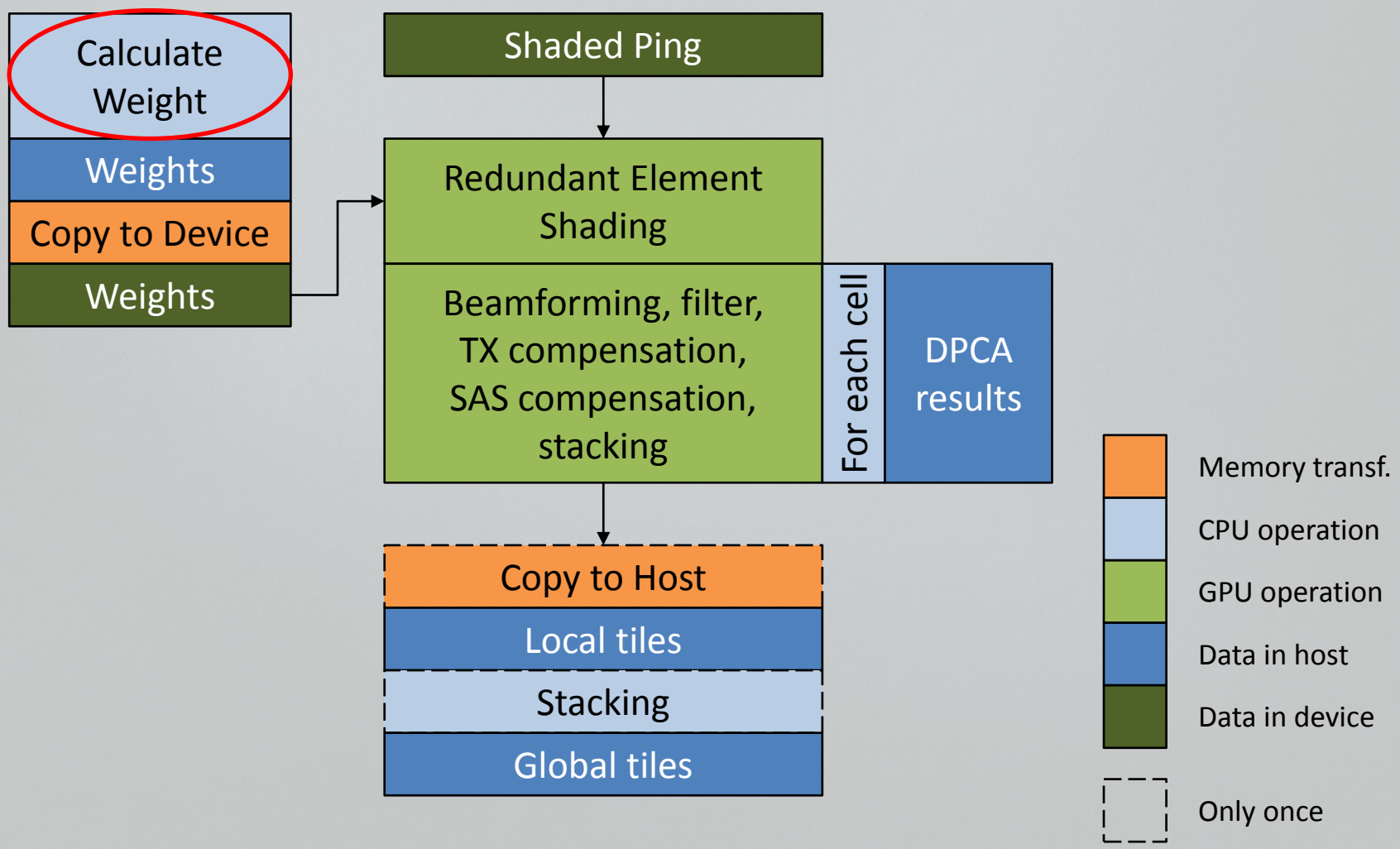
SAS processing



SAS Imaging Processing

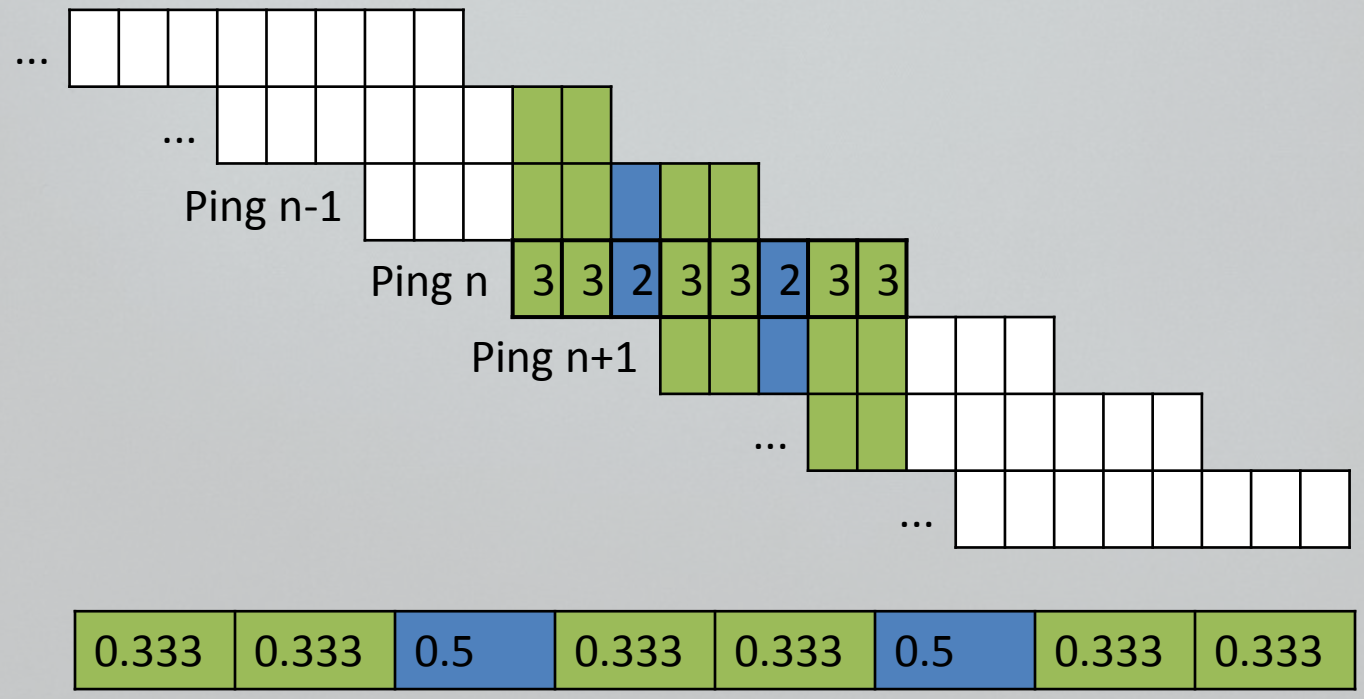


SAS Imaging Processing

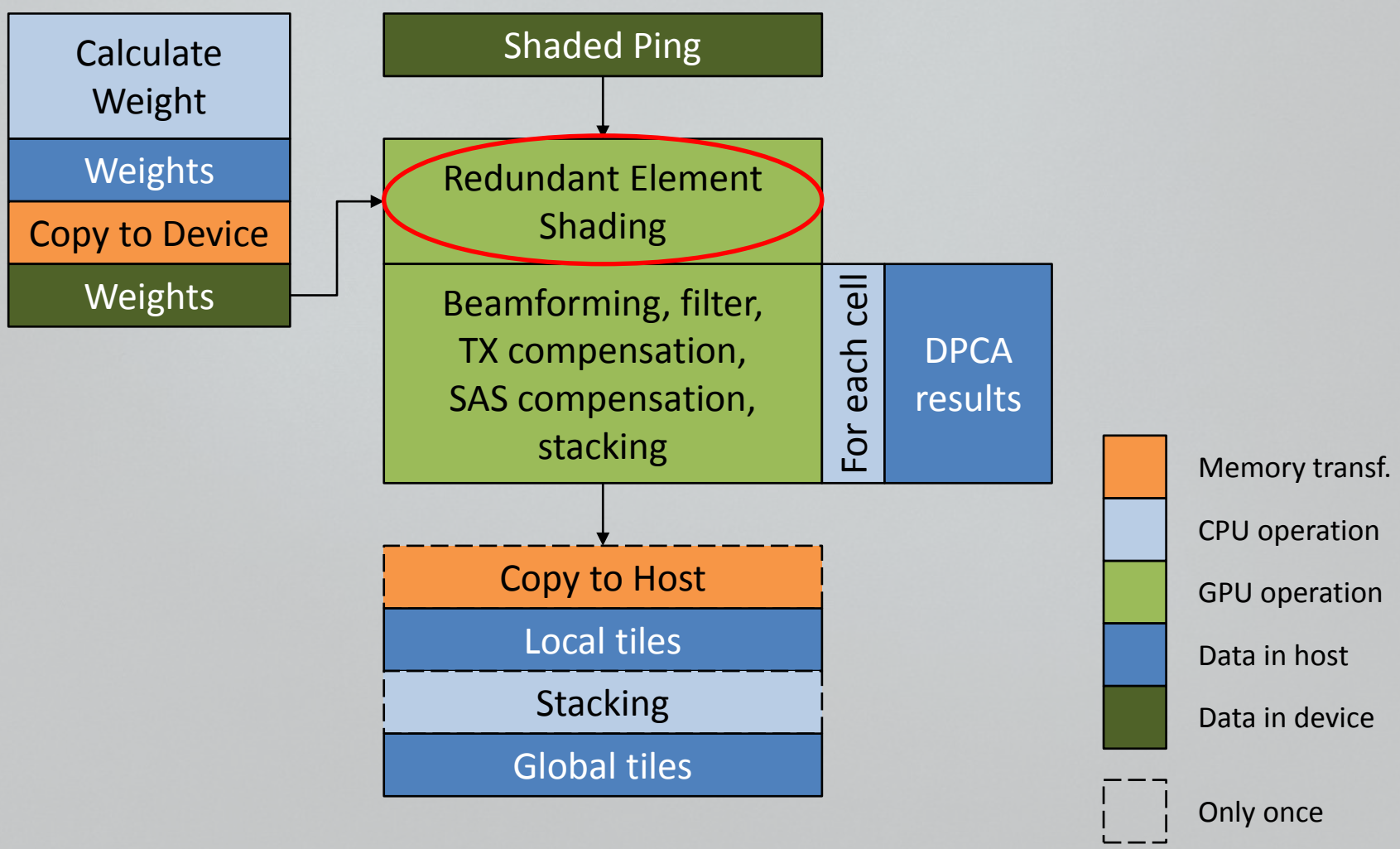


Redundant Element Shading

- Correct redundancy of the elements
- It depends on surge
- Example: 8 elements and displacement of 3



SAS Imaging Processing



Redundant Element Shading

- Multiply the acoustic data by the weight:

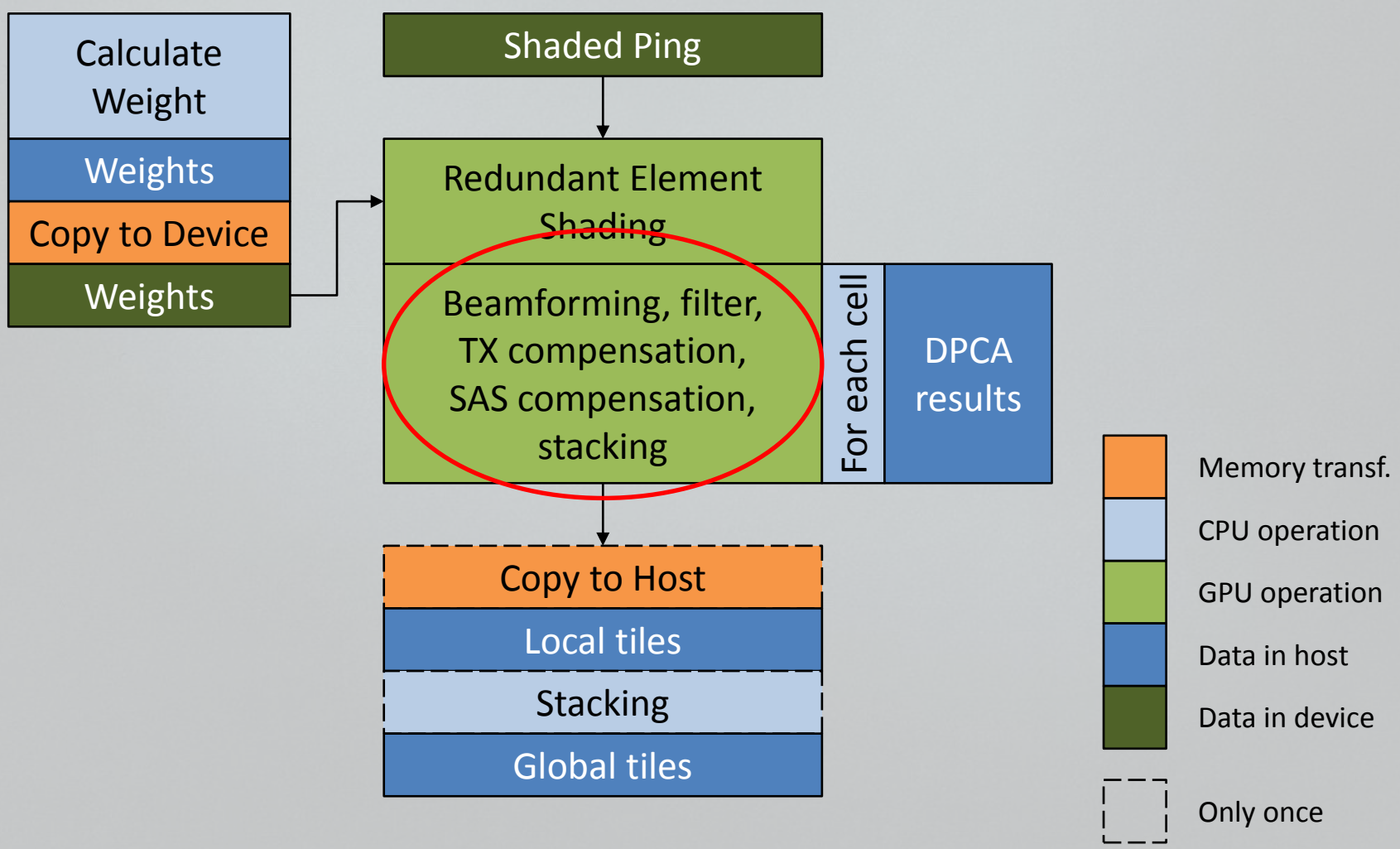
$$A_j^w = A_j \cdot w_j$$

Where:

- A_j^w is the weighted acoustic data of the element j
- A_j is the acoustic data of the element j
- w_j is the weight for the element j

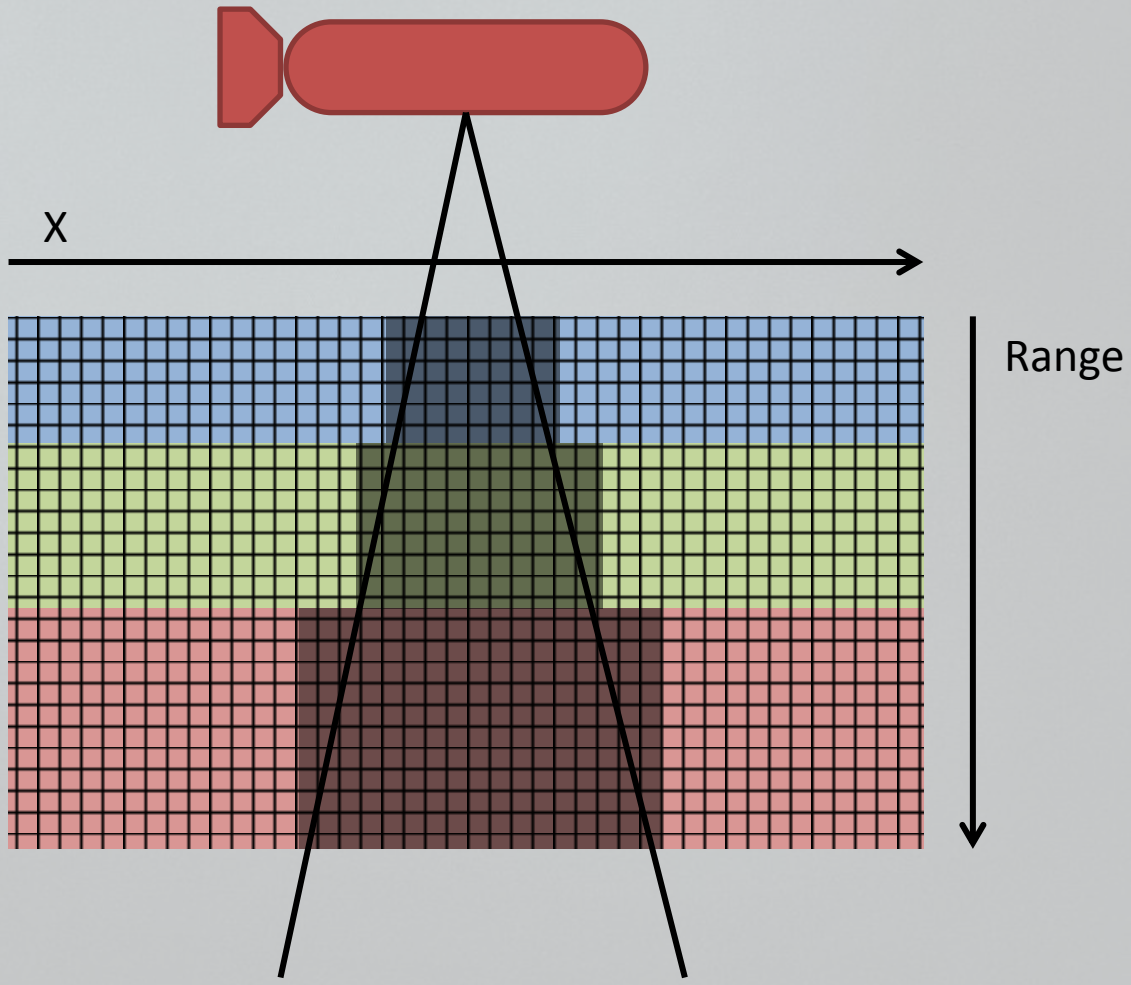
- The multiplication is done in place (acoustic data is overwritten), to save memory.

SAS Imaging Processing



SAS Imaging - Beamforming

- Each cell has:
  - Min range
  - Max range
- We calculate:
  - Min X
  - Max X
- With this values we have the bounding box (dark areas)



SAS Imaging - Beamforming

- Number of blocks = Cells in range
- Number of threads = Cells in X
- Variable number of cells in range → No big problem
- Variable number of cells in X → Optimization problem
  - Each thread calculates a fix number of points (adjusted to the maximum expected size) and we vary the number of threads → maybe odd number of threads
  - Variable number of points per thread and fixed number of threads (optimized number of threads for occupancy) → Problem with small numbers
  - Variable number of points per thread and number of threads → More adaptable to any size → Store pair of values in a table (input: number of cells in X, output: number of points per thread and number of threads)

SAS Imaging - Beamforming

- Blocks and threads distribution

Blocks (range)

			1	2	3	4			
		1	2	3	4	1	2		
		1	2	3	4	1	2		
		1	2	3	4	1	2		
	1	2	3	4	1	2	3	4	
	1	2	3	4	1	2	3	4	
	1	2	3	4	1	2	3	4	
1	2	3	4	1	2	3	4	1	2
1	2	3	4	1	2	3	4	1	2
1	2	3	4	1	2	3	4	1	2

Threads (X)

SAS Imaging - Beamforming

- Range index (r) and position (R) of the block:

$$r = r_{min} + j, \text{where } j = 0..N_b - 1 \text{ and } N_b = r_{max}\text{-}r_{min}$$

$$R = r \cdot R_{res} + R_{min}$$

where:

- r_{min} and r_{max} are the minimum and maximum range index
- R_{min} and R_{max} are the minimum and maximum range distance
- N_b is the number of blocks
- r is the range index
- R is the range distance
- R_{res} is the resolution

SAS Imaging - Beamforming

- Calculate x_{min} and x_{max} from range and beam angle
- X index of the thread:

$$x = x_{min} + i + k \cdot N_t \text{ , where } i = 0 \ldots N_t - 1 \text{ and } k = 0 \ldots N_{pt} - 1$$

Alternative calculation (slower): $x = x_{min} + i \cdot N_{pt} + k$

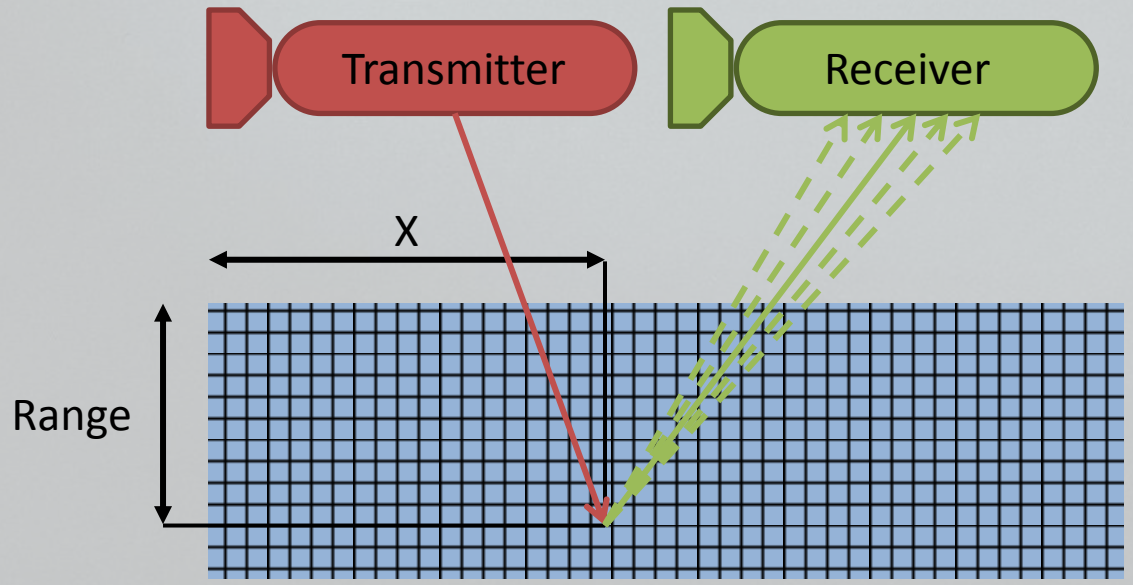
where:

- x_{min} and x_{max} are the minimum and maximum X index
- N_t is the number of threads
- N_{pt} is the number of points per thread
- x is the X index

- Conversion between index and position analogue to range, with X, X_{min}, X_{max} and X_{res}

SAS Imaging - Beamforming

- With position in range and X we can calculate the distance to the transmitter and to each receiver using the AUV position estimation from the DPCA
- The total distance travelled by the sound (transmitter → floor → receiver) gives us the sample index in the acoustic data



SAS Imaging - Filter

- Instead of taking one single sample, we use a filter similar to *sinc*
- Store acoustic data and filter in 1D textures (float2 format)

```
distanceTransmitter = calculate_distance_to_transmitter(posTransmitter, posTile)
tileAcc = 0.0

for each receiver

  distanceReceiver = calculate_distance_to_receiver(posReceiver, posTile)
  pathLength = distanceTransmitter + distanceReceiver
  indexData = calculate_index_data(pathLength)
  indexFilter = calculate_index_filter(...)
  tileSum = 0.0

  for filter size
    tileSum += filter(...) * acousticData(...)
  next

  tileAcc += tileSum * distanceReceiver

next

tileAcc *= distanceTransmitter / nReceivers
```

SAS Imaging - Compensation

- Multiply the accumulated value by the compensation factors:
  - TX compensation: Eliminate rippling effects (similar to inverse *sinc* filter)
  - SAS compensation: Add smoothing in the beam
- Stacking:
  - Accumulate value in the tile

SAS Imaging - Results

- Performance depends on the resolution and range
- Three configurations:

  A. Testing: Short range (110-150m), medium resolution (0.025x0.050), port side

  B. Sea trials: Full range (35-145m), medium resolution (0.050x0.050), both sides

  C. Scientific: Full range (35-145m), high resolution (0.025x0.015), port side

	CPU[1]	GPU[2]	Speedup
A	3663.8ms	47.8ms	x77
B	8614.2ms	115.0ms	x75
C	28588.2ms	352.9ms	x81

[1] Running on an Intel Core i7-920 @ 2.67 GHz
[2] Running on a NVIDIA Tesla C1060

Global performance

	CPU[1]	GPU[2]	Speedup
Pre-processing	84.3 ms	3.7 ms	x23
DPCA	93.5 ms	18.4 ms	x5
Imaging A	3663.8ms	47.8ms	x77
Imaging B	8614.2ms	115.0ms	x75
Imaging C	28588.2ms	352.9ms	x81
Pings per second A	0.26	14.3	x55
Pings per second B	0.11	6.3	x57
Pings per second C	0.035	2.7	x77

[1] Running on an Intel Core i7-920 @ 2.67 GHz
[2] Running on a NVIDIA Tesla C1060

Sea Trials

- Software tested on board an AUV
- x86 processor with a NVIDIA GT240 equivalent graphics card
- 4 pings per second (250ms between pings)
- Running in real-time in medium resolution (configuration B)

Future work

- Upgrade the on-board GPU to calculate in real-time in high resolution
- Further software improvements to increase performance
- Continue development of multi-GPU and multi-CPU versions for desktop computers (scientific research)
- Add further algorithms in the chain (interferometry → generate 3D images from bathymetric and main arrays)
- Update algorithms? → Adapt them to GPU
- Currently testing the NVIDIA Carma platform → It's already running!

Thank you very much!