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)
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.
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
• 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
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)
• Sonar on both sides (port & starboard)
• Two receiver arrays per side:
  – **Main array** (36 elements)
  – Bathymetric array (12 elements)
Synthetic Aperture Sonar (SAS)

Port side

Starboard side

Range (cross-track)

Along-track (movement)

Range (cross-track)
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 $\rightarrow$ 3D images

Automatic Target Recognition (ATR)
Replanning
SAS processing

Time (4Hz – 250 ms per ping)

PING 1
Preprocessing

PING 2
Preprocessing

PING 3
Preprocessing

PING N
Preprocessing

DPCA
Motion

DPCA
Motion

DPCA
Motion

SAS Imaging

SAS Imaging

SAS Imaging

Stacking tiles
SAS processing

Time (4Hz – 250 ms per ping)

PING 1
Preprocessing

PING 2
Preprocessing

PING 3
Preprocessing

PING N
Preprocessing

DPCA
Motion

DPCA
Motion

DPCA
Motion

SAS Imaging

SAS Imaging

SAS Imaging

Stacking tiles

Stacking tiles

Stacking tiles
Preprocessing

Read Replica
  Replica
    FFT
      Replica Freq.
        Copy to Device
          Replica Freq.
            Multiply
              Inverse FFT
                Matched Ping
      Read Ping
        Ping
          Calibration
            FFT
                Multiply
                  Inverse FFT
                    Shaded Ping
  Shading
    FFT
      Replica Shaded
        Copy to Device
          Replica Shaded

Memory transf.
  CPU operation
  GPU operation
  Data in host
  Data in device
  Only once
Preprocessing

- **Read Replica**
  - Replica
  - **FFT**
  - Replica Freq.
  - Copy to Device
  - Replica Freq.
  - **Multiply**
  - Inverse FFT
  - Matched Ping

- **FFT**
  - Replica Freq.

- **Read Ping**
  - Ping
  - Copy to Device
  - Ping
  - **Calibration**
  - FFT

- **Shading**
  - Replica Shaded
  - Copy to Device
  - Replica Shaded
  - **Multiply**
  - Inverse FFT
  - Shaded Ping

**Legend:**
- Memory transf.
- CPU operation
- GPU operation
- Data in host
- Data in device
- Only once
• Calibrate acoustic data

– Adjust the phase of each receiver due to wiring and reading delays

\[ A_c = A_r \cdot C \Rightarrow \begin{align*}
A_c^r &= A_r^r \cdot C^r - A_r^i \cdot C^i \\
A_c^i &= A_r^r \cdot C^i + A_r^i \cdot C^r
\end{align*} \]

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

- Read Replica
  - Replica
    - FFT
      - Replica Freq.
        - Copy to Device
          - Replica Freq.
            - Multiply
              - Inverse FFT
                - Matched Ping
                - Read Ping
                  - Ping
                    - Calibration
                      - FFT
                        - Multiply
                          - Inverse FFT
                            - Shaded Ping
              - Copy to Device
                - Replica Freq.
                  - Ping
                    - Calibration
                      - FFT
                        - Multiply
                          - Inverse FFT
                            - Shaded Ping
      - Copy to Device
        - Read Ping
          - Ping
            - Calibration
              - FFT
                - Multiply
                  - Inverse FFT
                    - Shaded Ping
  - Shading
    - FFT
      - Replica Shaded
        - Copy to Device
          - Replica Shaded
            - Multiply
              - Inverse FFT
                - Shaded Ping
  - CPU operation
    - Inverse FFT
      - Data in host
        - Data in device
          - Only once
  - GPU operation
    - Memory transf.
• 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)

<table>
<thead>
<tr>
<th></th>
<th>CPU\textsuperscript{1}</th>
<th>GPU\textsuperscript{2}</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>84.3 ms</td>
<td>3.7 ms</td>
<td>x23</td>
</tr>
</tbody>
</table>

\textsuperscript{1} Running on an Intel Core i7-920 @ 2.67 GHz
\textsuperscript{2} Running on a NVIDIA Tesla C1060
SAS processing

Time (4Hz – 250 ms per ping)

PING 1
Preprocessing

PING 2
Preprocessing

PING 3
Preprocessing

PING N
Preprocessing

DPCA
Motion

SAS Imaging

Stacking tiles
Phase Center Approximation

\[ \text{PCA} \]

\[ \text{Rx} \]

\[ \text{Tx} \]

\[ d \]

\[ d/2 \]
DPCA concept

D → AUV displacement

\( d \rightarrow \text{Separation elements} \)

\( N \rightarrow \text{Number of elements} \)

\( K=(N-M) \rightarrow \text{Number of overlapped elements} \)

• 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 $\rightarrow$ surge estimation
  – Accelerometer $\rightarrow$ pitch & roll
  – Compass $\rightarrow$ yaw
  – Depth $\rightarrow$ heave

• DPCA:
  – Surge correction
  – Sway calculation
• Divide the acoustic data in cells
• Each cell covers a different range of time/distance
DPCA Processing

For each cell:

- Matched Ping Previous
- Copy to cell
- Normalize
- FFT

- Matched Ping Current
- Copy to cell
- Normalize
- FFT

- Beamforming
- Inverse FFT
- Scale
- Fourier Interpolation

Calculate maxima:

- Copy to Host
- Maxima
- Navigation

DPCA results:
Surge, sway, roll, pitch, yaw, correlation factor, transmitter and receiver position, etc

Memory transf.
CPU operation
GPU operation
Data in host
Data in device
DPCA Processing

For each cell:
- Matched Ping
- Copy to cell
- Normalize
- FFT

Beamforming:
- Shift data
- FFT
- Inverse FFT
- Scale
- Fourier Interpolation

Calculate maxima:
- Copy to Host
- Maxima
- Navigation

DPCA results:
Surge, sway, roll, pitch, yaw, correlation factor, transmitter and receiver position, etc

Memory transf.
CPU operation
GPU operation
Data in host
Data in device
DPCA - Normalize

• Calculate modulus:

\[
\text{modulus} = \sqrt{\sum_{i=1}^{N_S} (R_i^2 + I_i^2)}
\]

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:

\[
R'_i = \frac{R_i}{\text{modulus}} \quad I'_i = \frac{I_i}{\text{modulus}}
\]
DPCA - Normalize

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

\[
\text{modulus}_t^2 = \sum_{i=t\cdot N_{st}+1}^{(t+1)\cdot N_{st}} (R_i^2 + I_i^2)
\]

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
• 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

For each cell:
- Matched Ping
- Copy to cell
- Normalize
- FFT

Beamforming:
- Shift data
- FFT
- Inverse FFT
- Scale
- Fourier Interpolation

Calculate maxima:
- Copy to Host
- Maxima
- Navigation
- DPCA results: Surge, sway, roll, pitch, yaw, correlation factor, transmitter and receiver position, etc

Memory transf.
CPU operation
GPU operation
Data in host
Data in device

Matched Ping
Previous

Matched Ping
Current
DPCA - Beamforming

• Multiply current and previous cell with a beamforming coefficient:

\[
\text{correlation}_i = A_i^{\text{curr}} \cdot \overline{A_i^{\text{prev}}}_{N-K+i}, \text{ with } i = 1 \ldots K
\]

\[
\text{bfArray}_i = \text{correlation}_i \cdot \text{bfCoefficient}_i
\]

where

• \(A_i^{\text{curr}}\) is the current acoustic data
• \(A_i^{\text{prev}}\) is the previous acoustic data
• \(\text{correlation}_i\) is the correlation between the current and previous
• \(\text{bfCoefficient}_i\) is the beamforming coefficient
• \(\text{bfArray}_i\) is the beamforming array
DPCA Processing

For each cell:
- Matched Ping
  - Previous
  - Current

- Copy to cell
- Normalize
- FFT

Beamforming:
- Shift data
- FFT
- Inverse FFT
- Scale
- Fourier Interpolation

Calculate maxima:
- Copy to Host
  - Maxima
  - Navigation

DPCA results:
- Surge, sway, roll, pitch, yaw,
- correlation factor,
- transmitter and receiver position, etc

CPU operation
GPU operation
Data in host
Data in device
Memory transf.
• 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
Calculate the maximum using a tree, first thread stores results in global memory

```c
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)

<table>
<thead>
<tr>
<th>CPU¹</th>
<th>GPU²</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>93.5 ms</td>
<td>18.4 ms</td>
<td>x5</td>
</tr>
</tbody>
</table>

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

Time (4Hz – 250 ms per ping)

PING 1
Preprocessing

PING 2
Preprocessing

PING 3
Preprocessing

PING N
Preprocessing

DPCA
Motion

SAS Imaging

Stacking tiles
SAS Imaging Processing

Calculate Weight

Weights

Copy to Device

Weights

Shaded Ping

Redundant Element Shading

Beamforming, filter, TX compensation, SAS compensation, stacking

For each cell

DPCA results

Copy to Host

Local tiles

Stacking

Global tiles

Memory transf.

CPU operation

GPU operation

Data in host

Data in device

Only once
SAS Imaging Processing

- Calculate Weight
- Weights
- Copy to Device
- Weights
- Shaded Ping
- Redundant Element Shading
  - Beamforming, filter, TX compensation, SAS compensation, stacking
- Copy to Host
  - Local tiles
  - Stacking
  - Global tiles
- For each cell
  - DPCA results
- Memory transf.
- CPU operation
- GPU operation
- Data in host
- Data in device
- Only once
• Correct redundancy of the elements
• It depends on surge
• Example: 8 elements and displacement of 3

Redundant Element Shading

- Ping n-1
- Ping n
- Ping n+1

0.333 0.333 0.5 0.333 0.333 0.5 0.333 0.333
SAS Imaging Processing

Calculate Weight
Weights
Copy to Device
Weights

Shaded Ping

Redundant Element Shading

Beamforming, filter, TX compensation, SAS compensation, stacking

Copy to Host
Local tiles
Stacking
Global tiles

For each cell
DPCA results

Memory transf.
CPU operation
GPU operation
Data in host
Data in device
Only once
• 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

Calculate Weight
Weights
Copy to Device
Weights

Shaded Ping

Redundant Element
Shading

Beamforming, filter, TX compensation, SAS compensation, stacking

For each cell
DPCA results

Copy to Host
Local tiles
Stacking
Global tiles

Memory transf.
CPU operation
GPU operation
Data in host
Data in device
Only once
• Each cell has:
  – Min range
  – Max range

• We calculate:
  – Min X
  – Max X

• With this values we have the bounding box (dark areas)
• 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)
- **Blocks and threads distribution**

![Blocks and threads distribution diagram](image-url)
• Range index (r) and position (R) of the block:

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

\[ R = r \cdot R_{\text{res}} + R_{\text{min}} \]

where:

– \( r_{\text{min}} \) and \( r_{\text{max}} \) are the minimum and maximum range index
– \( R_{\text{min}} \) and \( R_{\text{max}} \) are the minimum and maximum range distance
– \( N_b \) is the number of blocks
– \( r \) is the range index
– \( R \) is the range distance
– \( R_{\text{res}} \) is the resolution
SAS Imaging - Beamforming

• Calculate $x_{min}$ and $x_{max}$ from range and beam angle

• X index of the thread:

\[ x = x_{min} + i + k \cdot N_t \] , where \( i = 0 \ldots N_t - 1 \) 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 $X_{res}$
• 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
• Instead of taking one single sample, we use a filter similar to $sinc$
• Store acoustic data and filter in 1D textures (float2 format)

```c
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
```
• 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

<table>
<thead>
<tr>
<th></th>
<th>CPU¹</th>
<th>GPU²</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3663.8ms</td>
<td>47.8ms</td>
<td>x77</td>
</tr>
<tr>
<td>B</td>
<td>8614.2ms</td>
<td>115.0ms</td>
<td>x75</td>
</tr>
<tr>
<td>C</td>
<td>28588.2ms</td>
<td>352.9ms</td>
<td>x81</td>
</tr>
</tbody>
</table>

¹ Running on an Intel Core i7-920 @ 2.67 GHz
² Running on a NVIDIA Tesla C1060
# Global performance

<table>
<thead>
<tr>
<th></th>
<th>CPU(^1)</th>
<th>GPU(^2)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-processing</td>
<td>84.3 ms</td>
<td>3.7 ms</td>
<td>x23</td>
</tr>
<tr>
<td>DPCA</td>
<td>93.5 ms</td>
<td>18.4 ms</td>
<td>x5</td>
</tr>
<tr>
<td>Imaging A</td>
<td>3663.8 ms</td>
<td>47.8 ms</td>
<td>x77</td>
</tr>
<tr>
<td>Imaging B</td>
<td>8614.2 ms</td>
<td>115.0 ms</td>
<td>x75</td>
</tr>
<tr>
<td>Imaging C</td>
<td>28588.2 ms</td>
<td>352.9 ms</td>
<td>x81</td>
</tr>
<tr>
<td>Pings per second A</td>
<td>0.26</td>
<td>14.3</td>
<td>x55</td>
</tr>
<tr>
<td>Pings per second B</td>
<td>0.11</td>
<td>6.3</td>
<td>x57</td>
</tr>
<tr>
<td>Pings per second C</td>
<td>0.035</td>
<td>2.7</td>
<td>x77</td>
</tr>
</tbody>
</table>

\(^1\) 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!