EECS 570 Programming Assignment 1

University of Michigan

January 17th, 2025

A Little About Me

- Education
  - M.S. in ECE, IC VLSI Track
  - B.Tech in EE + CS, IIT Bombay
- Research
  - CGRA Design, Prof. Blaauw's Lab
- Hobbies
  - Climbing

Announcements

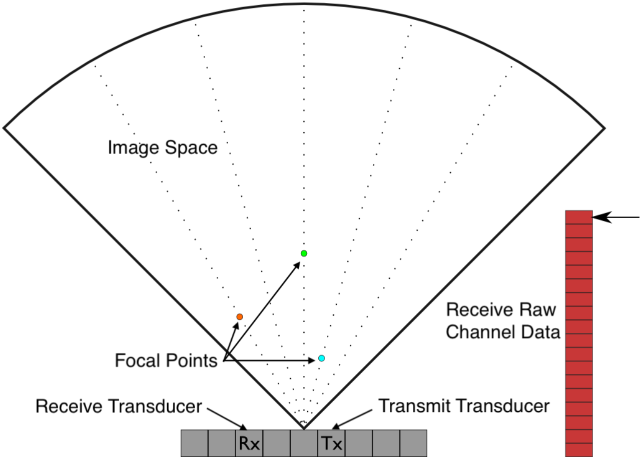
- Final Project
  - Piazza - search for teammates
  - Next friday: Project introduction/handout
- PA1
  - Due 2/10 on canvas
- Office Hours
  - Tue & Thu: 12-1:30 PM, BBB Atrium
  - Fri (when no discussion): 3:30-4:30 PM, Zoom

Overview

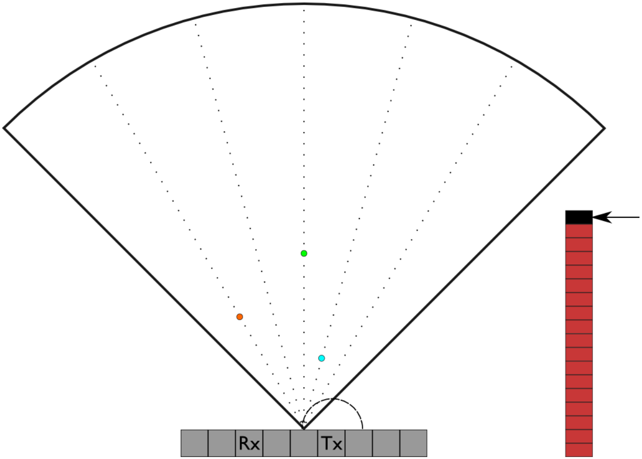
Portable Medical Imaging Devices

- Medical imaging moving towards portability
  - MEDICS (X-Ray CT) [Dasika '10]
  - Handheld 2D Ultrasound [Fuller '09]
- Not just a matter of convenience
  - Improved patient health [Gunnarsson '00, Weinreb '08]
  - Access in developing countries
- Why ultrasound?
  - Low transmit power [Nelson '10]
  - No danger or side-effects

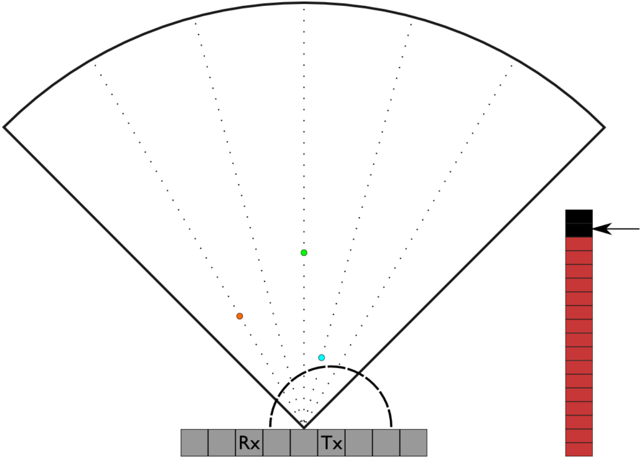
Ultrasound: Transmission and Reception



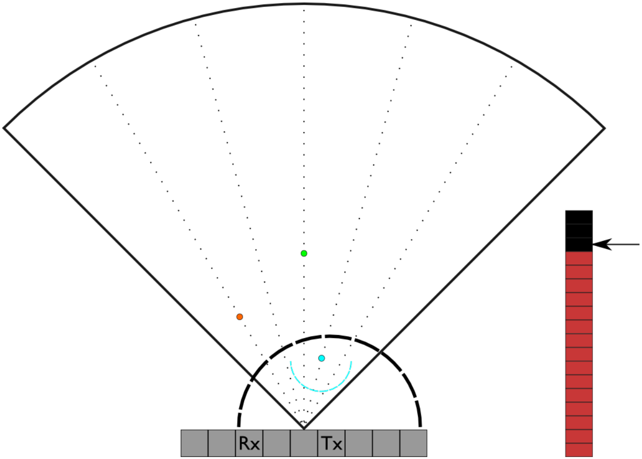
Ultrasound: Transmission and Reception



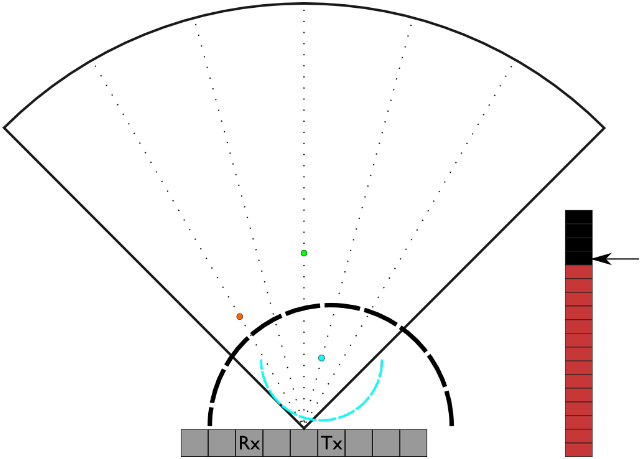
Ultrasound: Transmission and Reception



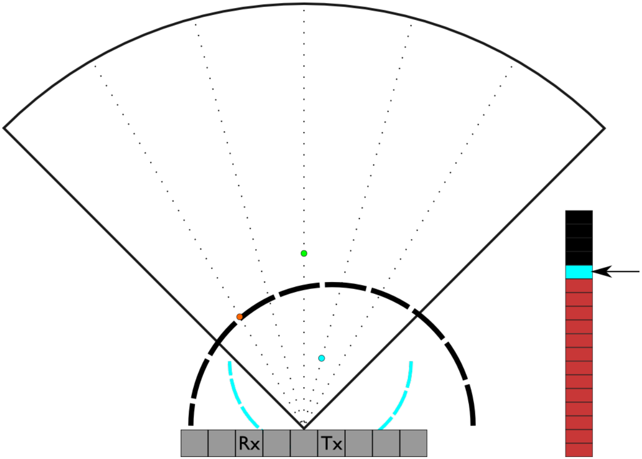
Ultrasound: Transmission and Reception



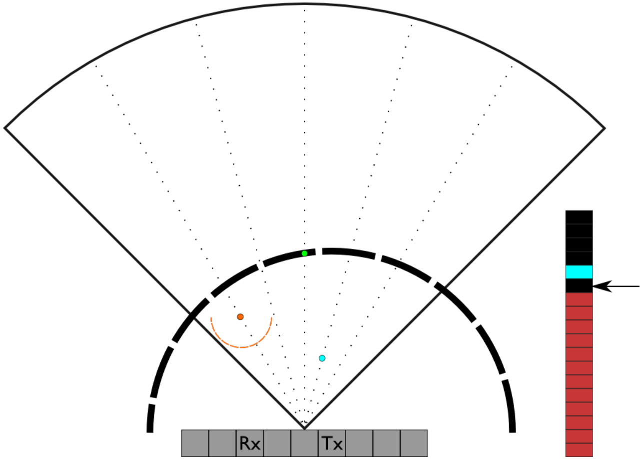
Ultrasound: Transmission and Reception



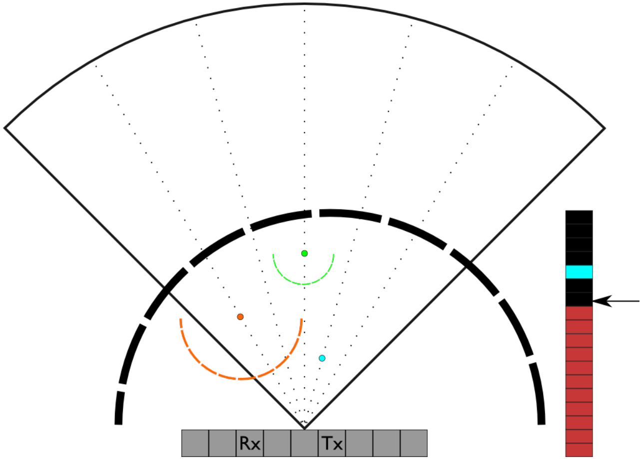
Ultrasound: Transmission and Reception



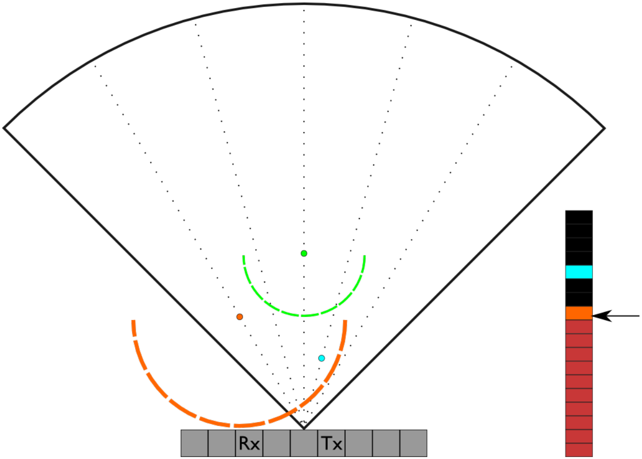
Ultrasound: Transmission and Reception



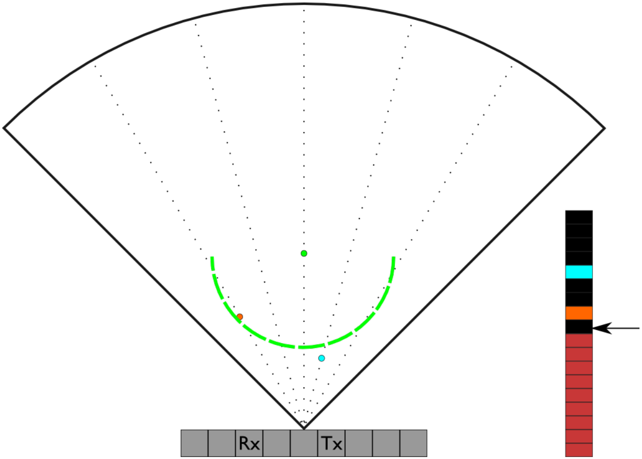
Ultrasound: Transmission and Reception



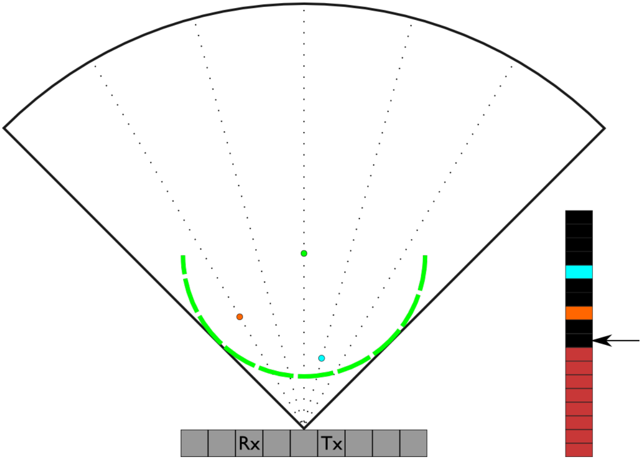
Ultrasound: Transmission and Reception



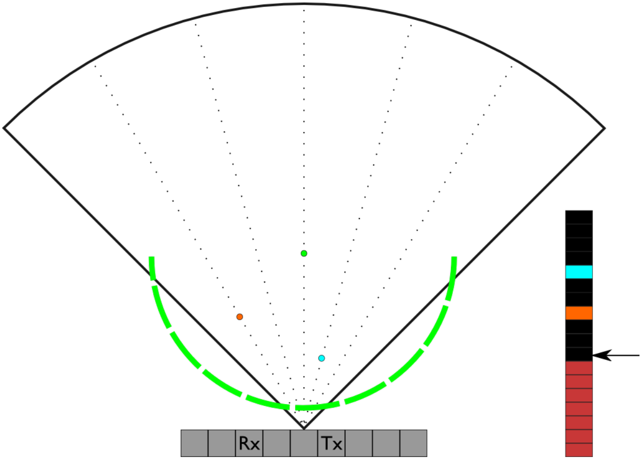
Ultrasound: Transmission and Reception



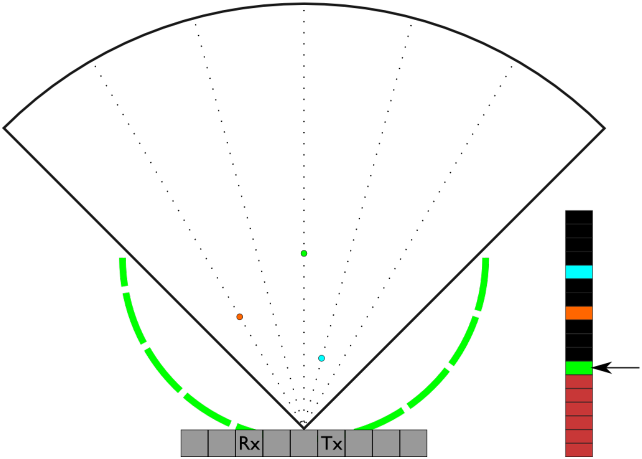
Ultrasound: Transmission and Reception



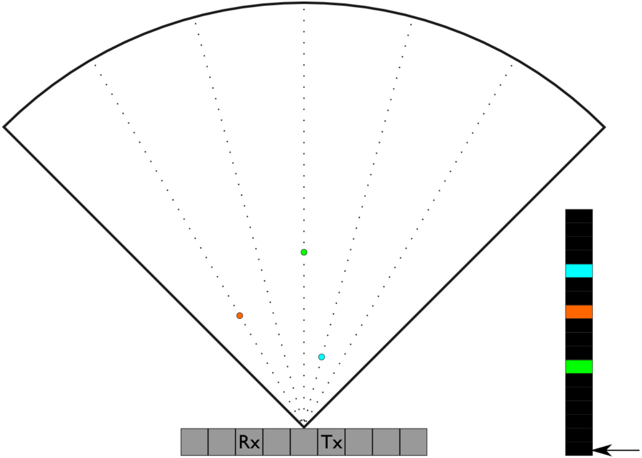
Ultrasound: Transmission and Reception



Ultrasound: Transmission and Reception



Ultrasound: Transmission and Reception



Each transducer stores an array of raw received data

Ultrasound: Transmission and Reception

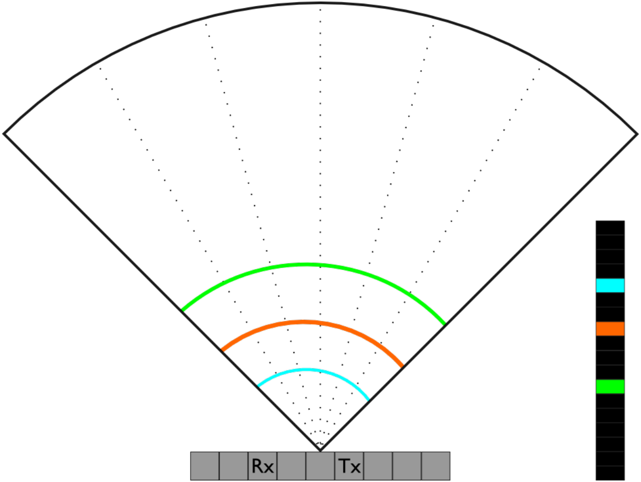
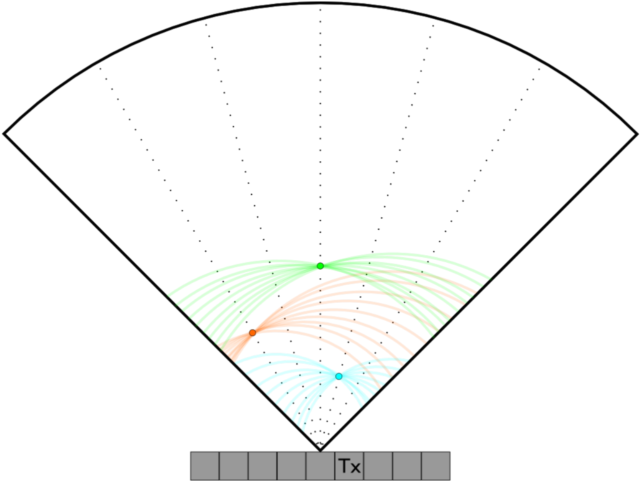


Image reconstructed from data based on round-trip delay

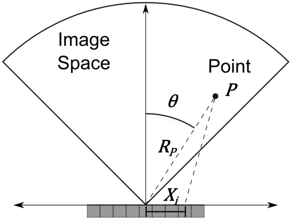
Ultrasound: Transmission and Reception



Images from each transducer combined to produce the full frame

Delay Index Calculation

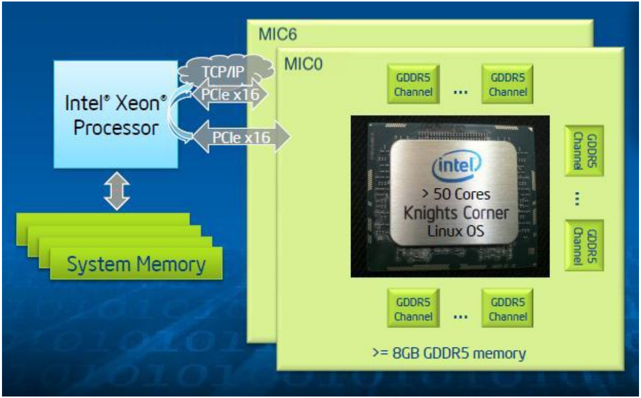
- Iterate through all image points for each transducer and calculate delay index τ_P



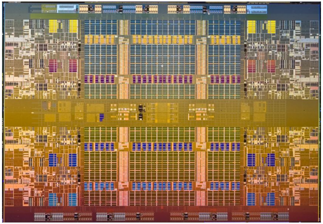
$$\tau_P = \frac{f_s}{c}\left(R_p + \sqrt{R_P^2 + X_i^2 - 2R_P X_i \sin\theta}\right)$$

- Often done with lookup tables (LUTs) instead
- 50 GB LUT required for target 3D system

Intel Xeon Phi Coprocessors and the MIC Architecture

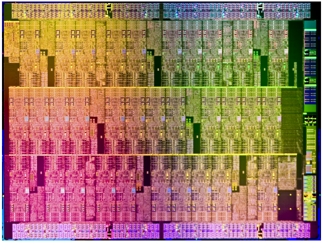


Intex Xeon Processors and the MIC Architecture



Multi-core Intel Xeon processor

- C/C++/Fortran; OpenMP/MPI
- Standard Linux OS
- Up to 768 GB of DDR3 RAM
- $\geq$ 12 cores/socket $\approx$ 3 GHz
- 2-way hyper-threading
- 256-bit AVX vectors



Many-core Intel Xeon Phi coprocessor

- C/C++/Fortran; OpenMP/MPI
- Special Linux μOS distribution
- 6-16 GB cached GDDR5 RAM
- 57-61 cores at $\approx$ 1 GHz
- 4-way hyper-threading
- 512-bit IMCI vectors

Xeon Phi Programming Models

- Native coprocessor applications
  - Compile with `-mmic`
  - Run with `micnativeloadex` or `scp+ssh`
  - The way to go for MPI applications without offload
- Explicit offload
  - Functions, global variables require `__attribute__((target(mic)))`
  - Initiate offload, data marshalling with `#pragma offload`
  - Only bitwise-copyable data can be shared
- Clusters and multiple coprocessors
  - `#pragma offload target(mic:i)`
  - Use threads to offload to multiple coprocessors
  - Run native MPI applications

Xeon Phi Programming Models

- Native coprocessor applications
  - Compile with `-mmic`
  - Run with `micnativeloadex` or `scp+ssh`
  - The way to go for MPI applications without offload
- Explicit offload
  - Functions, global variables require `__attribute__((target(mic)))`
  - Initiate offload, data marshalling with `#pragma offload`
  - Only bitwise-copyable data can be shared
- Clusters and multiple coprocessors
  - `#pragma offload target(mic:i)`
  - Use threads to offload to multiple coprocessors
  - Run native MPI applications

Native Execution

Example ("Hello World" application)

```
#include <stdio.h>
#include <unistd.h>
int main() {
    printf("Hello world! I have %ld logical cores.\n",
    sysconf(_SC_NPROCESSORS_ONLN ));
}
```

Example (compile and run on host)

```
user@host% icc -o hello hello.c
user@host% ./hello
Hello world! I have 32 logical cores.
user@host% _
```

Native Execution

Compile and run the same code on the coprocessor in native mode:

Example (compile and run on coprocessor)

```
user@host% icc -o hello.mic hello.c -mmic
user@host% micnativeloadex hello.mic -t 300 -d 0
Hello world! I have 240 logical cores.
user@host% _
```

- Use -mmic to produce executable for MIC architecture
- Use micnativeloadex to run the executable on the coprocessor
- Native MPI applications work the same way (need Intel MPI library)

Introduction to POSIX Threads

- What is a thread?

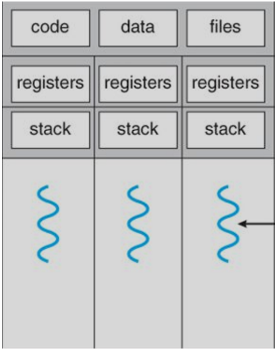
Introduction to POSIX Threads

- What is a thread?
  - Independently executing stream of instructions
  - Schedulable unit of execution for the operating system

Introduction to POSIX Threads

- What is a thread?
  - Independently executing stream of instructions
  - Schedulable unit of execution for the operating system
- Pthreads - the POSIX threading interface
  - Provides system calls to *create* and *synchronize* threads
  - Communication happens strictly through shared memory
    - Specifically, using *pointers* to shared data

Pthreads Tutorial



Posix Threads Tutorial

Questions?