EECS 570 **Programming Assignment 1**

Discussion

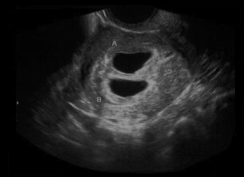
January 19 2024

Announcements

Project

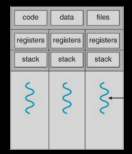
- Search for Teammates!
  - Piazza
- Fri 1/26: Discussion, Project Handout

PA1 Due Fri 2/9 11:59pon Canvas



1. 3D Ultrasound Beamforming

2. Intel Xeon Phi



3. POSIX Threads (Tutorial)

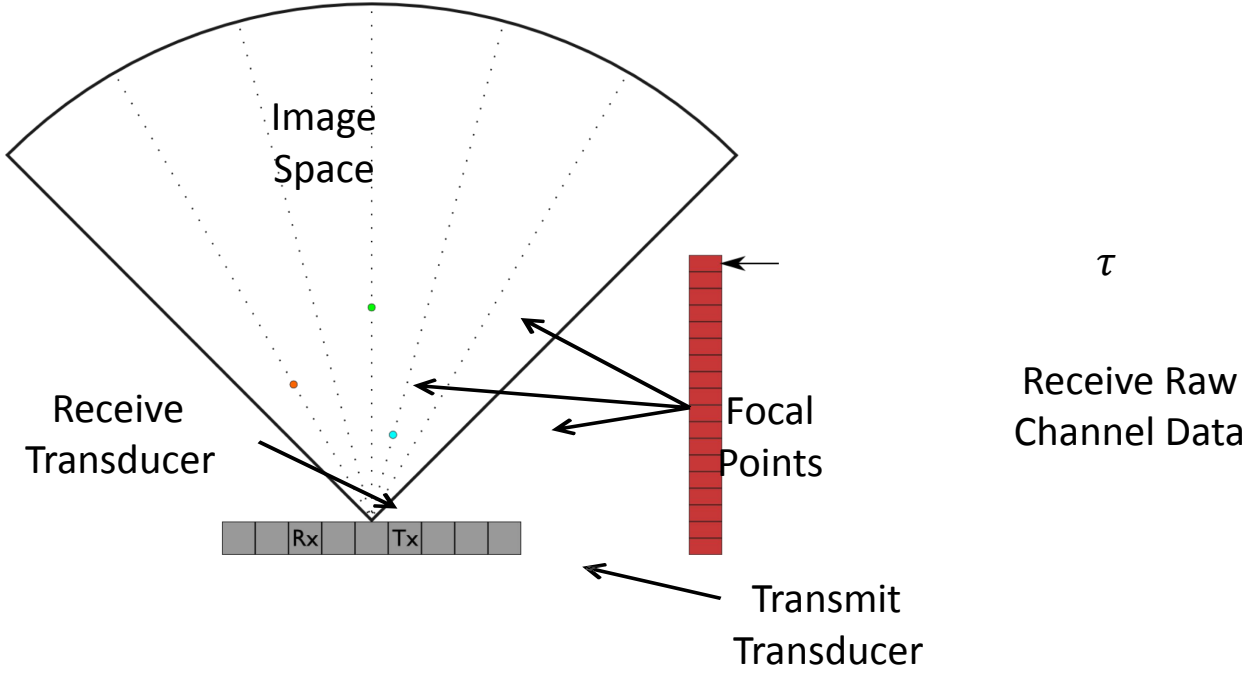
PA 1 Logistics

3D Ultrasound Beamforming

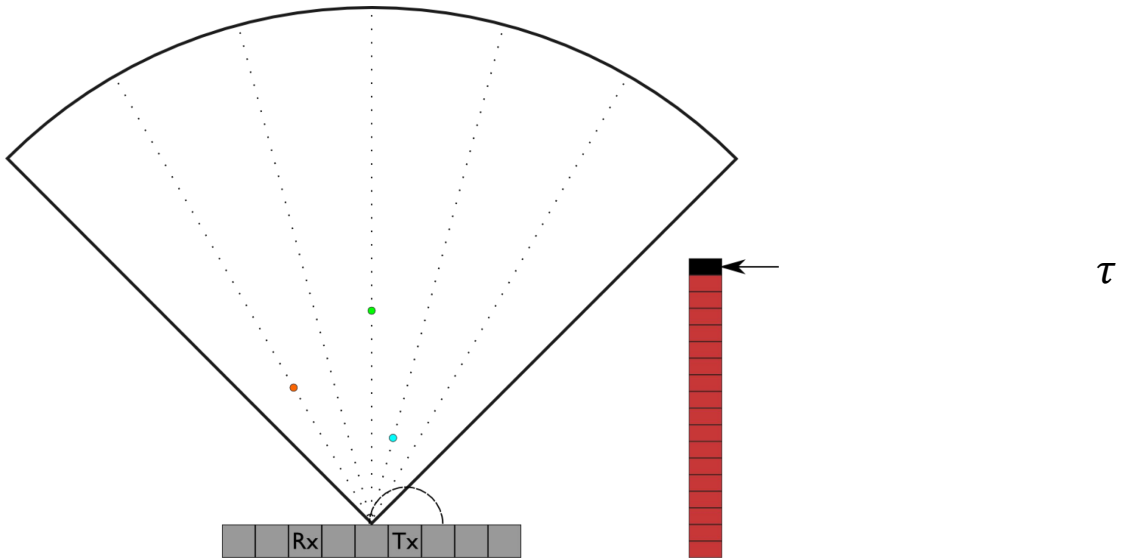
Portable Medical Imaging Devices

- 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 dangers or side-effects

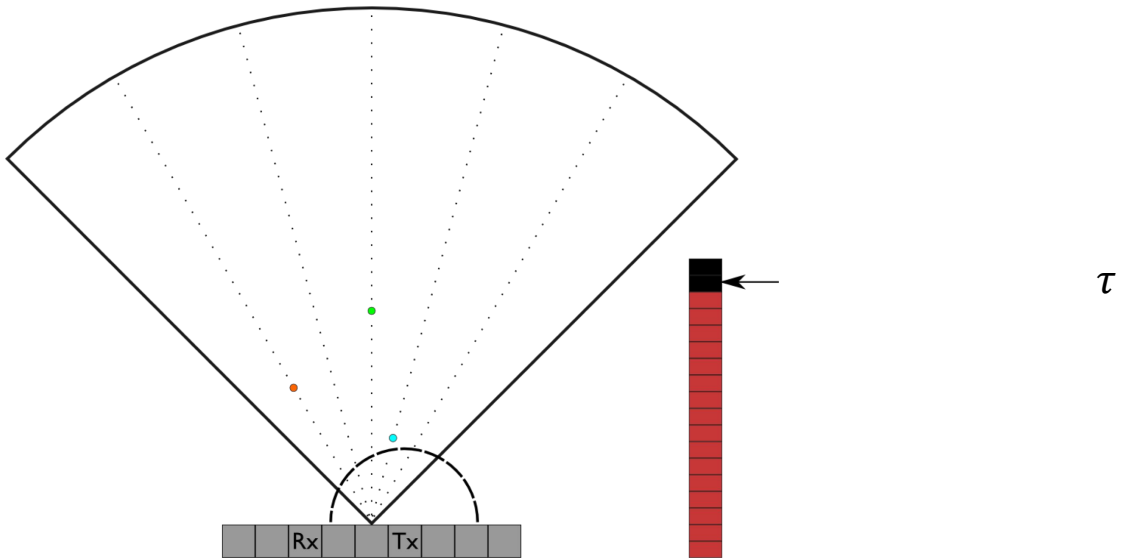
Ultrasound: Transmit and Receive



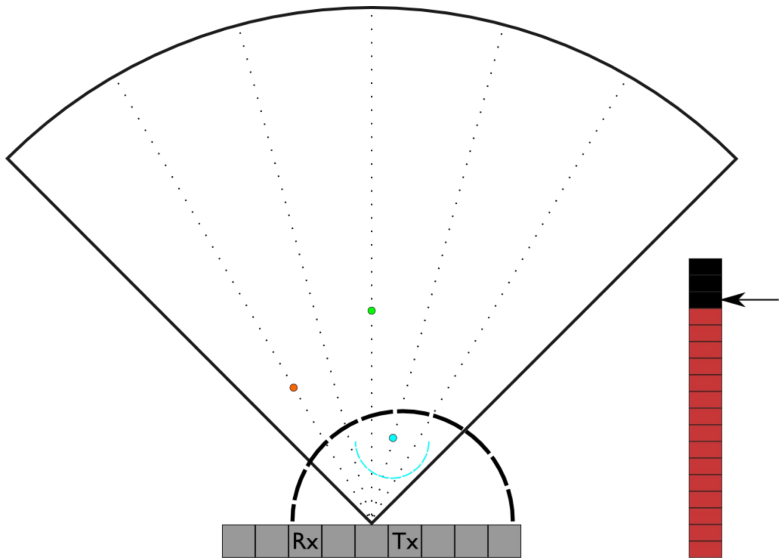
Ultrasound: Transmit and Receive



Ultrasound: Transmit and Receive

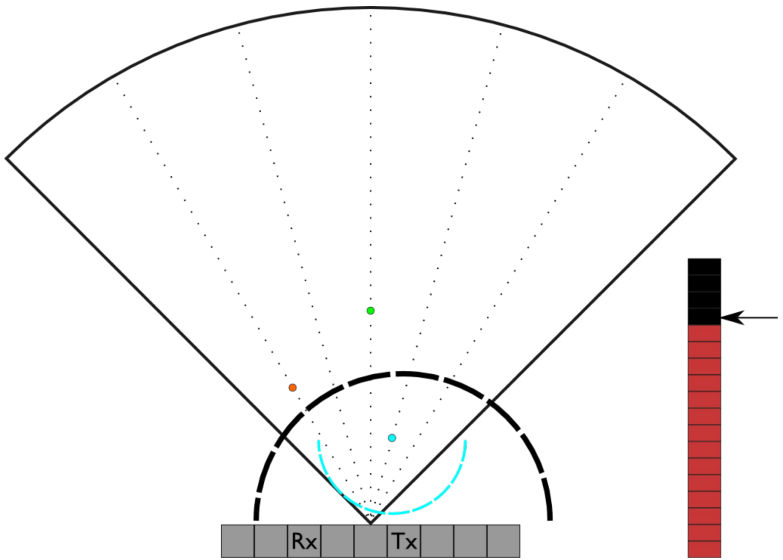


Ultrasound: Transmit and Receive



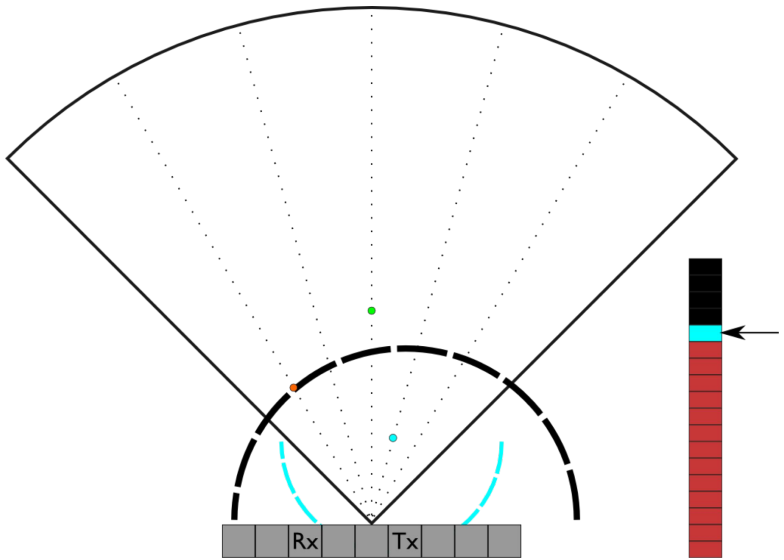
τ

Ultrasound: Transmit and Receive



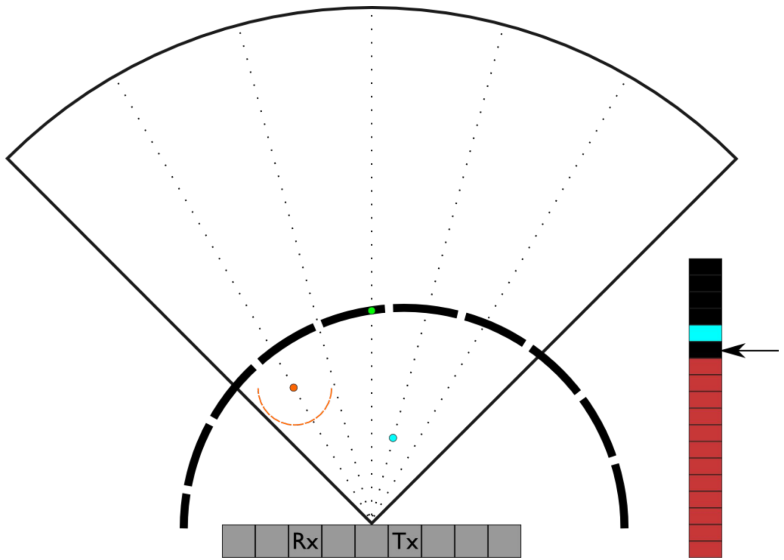
τ

Ultrasound: Transmit and Receive



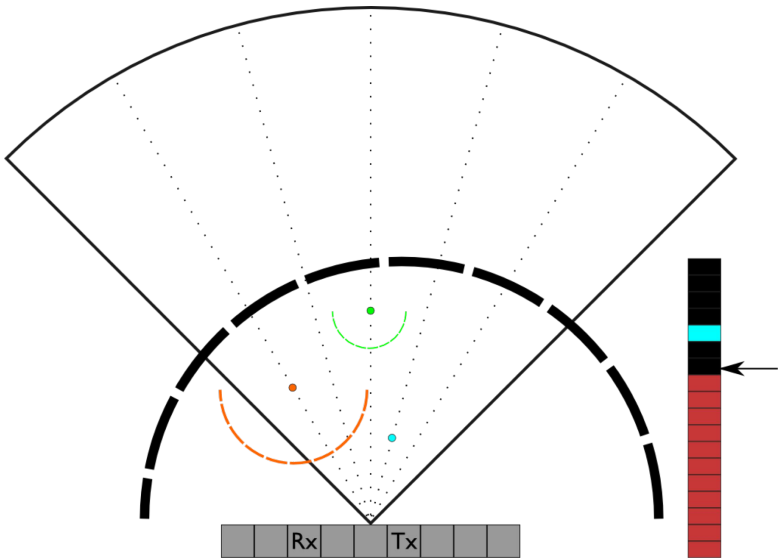
τ

Ultrasound: Transmit and Receive



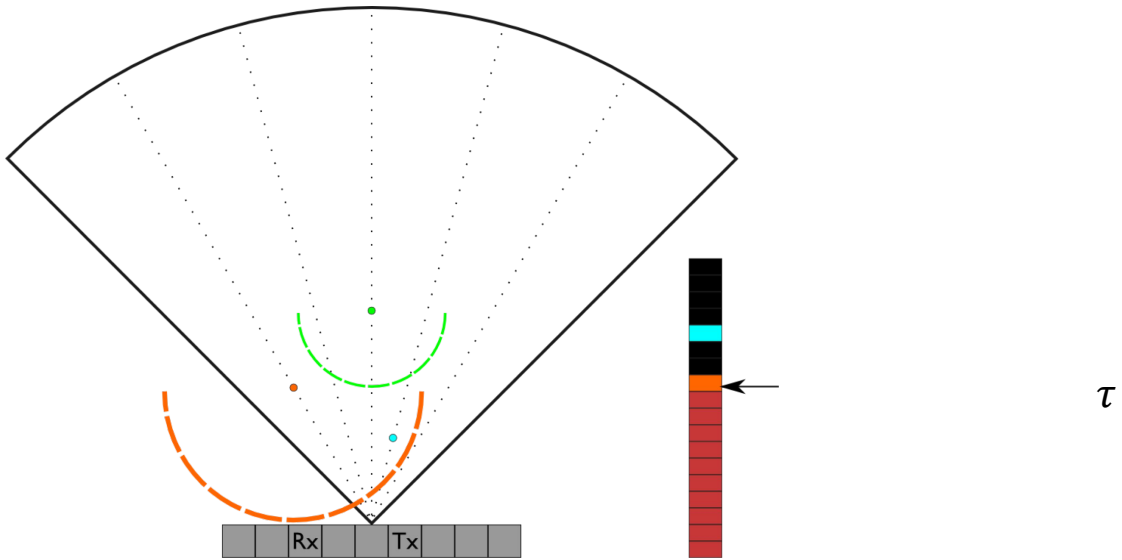
τ

Ultrasound: Transmit and Receive

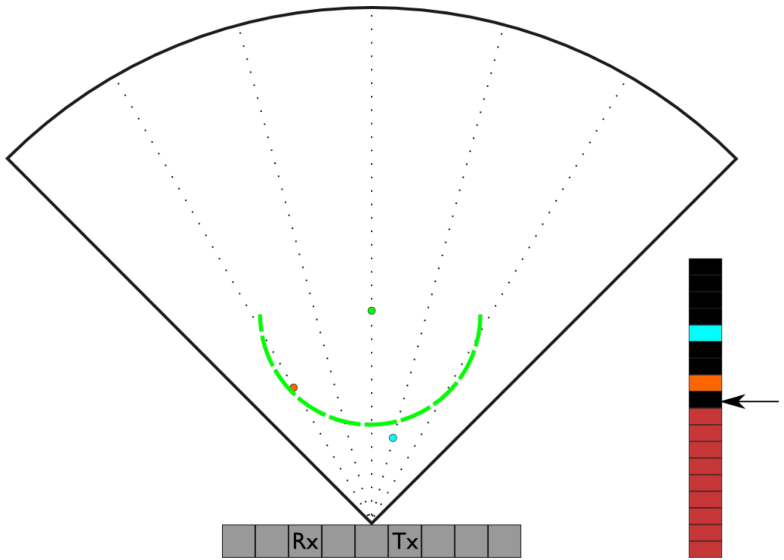


τ

Ultrasound: Transmit and Receive

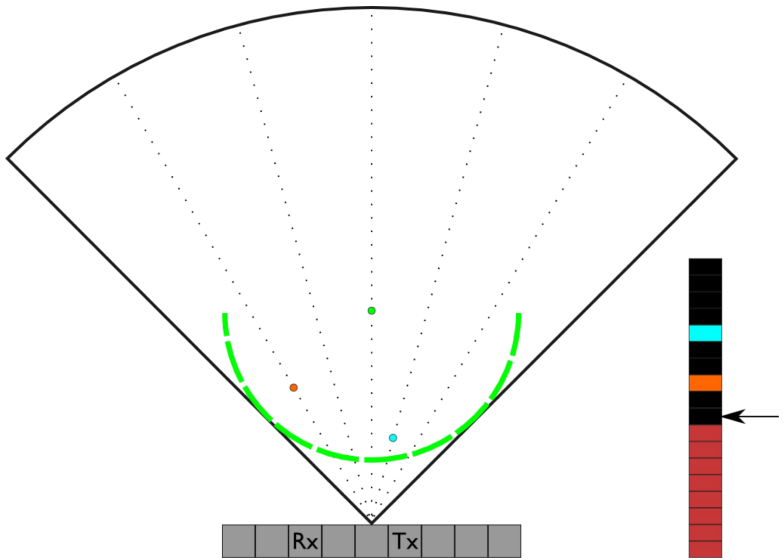


Ultrasound: Transmit and Receive



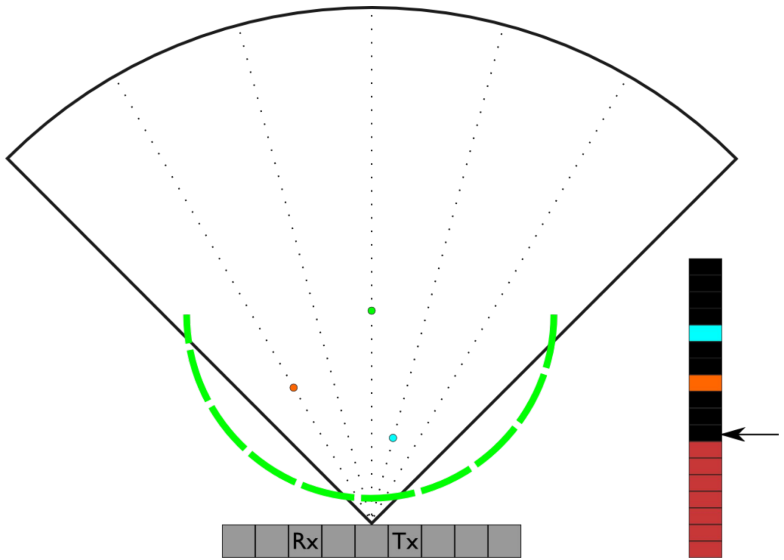
τ

Ultrasound: Transmit and Receive



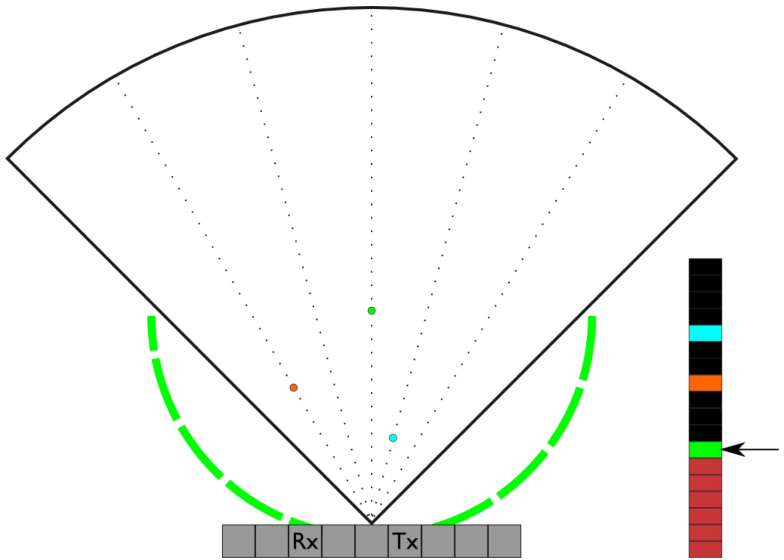
τ

Ultrasound: Transmit and Receive



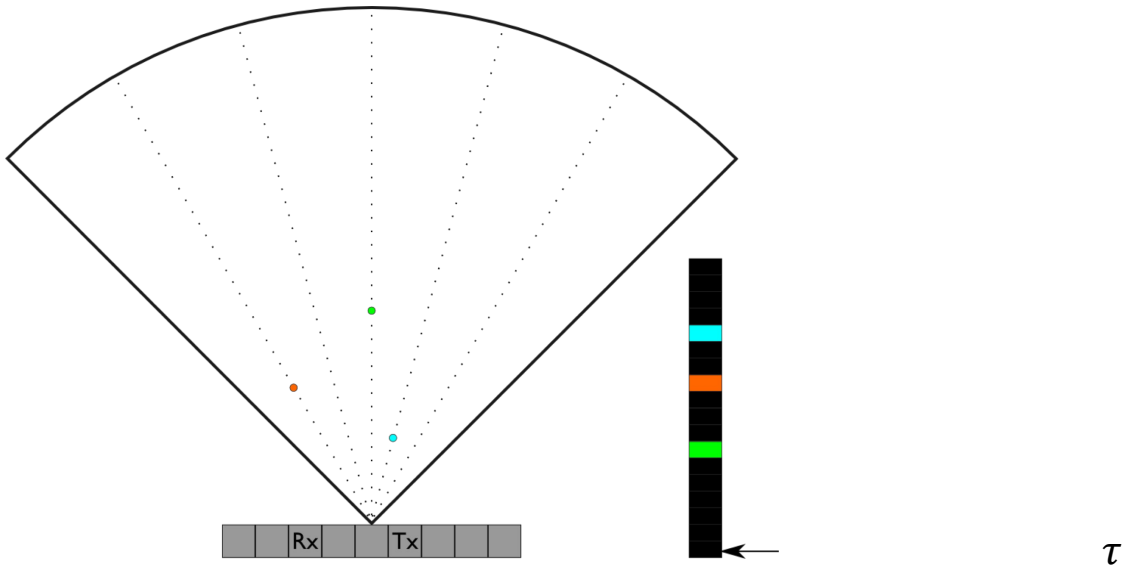
τ

Ultrasound: Transmit and Receive



τ

Ultrasound: Transmit and Receive



Each transducer stores array of raw receive data

Ultrasound: Image Reconstruction

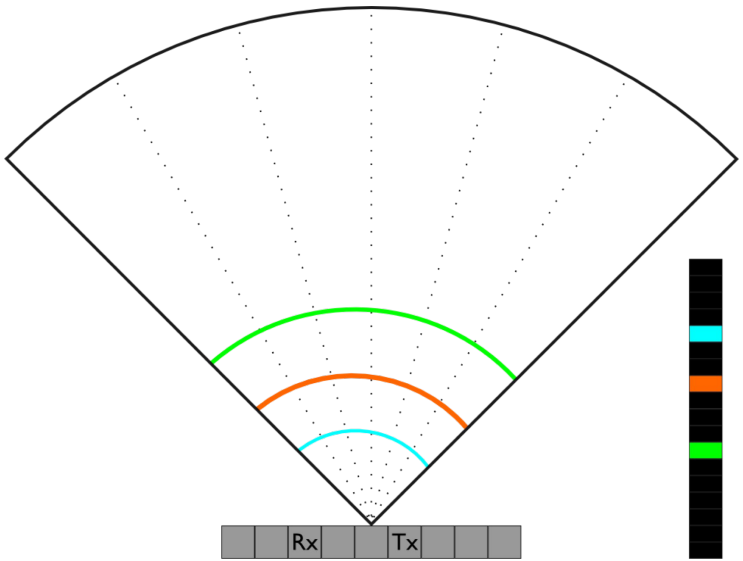
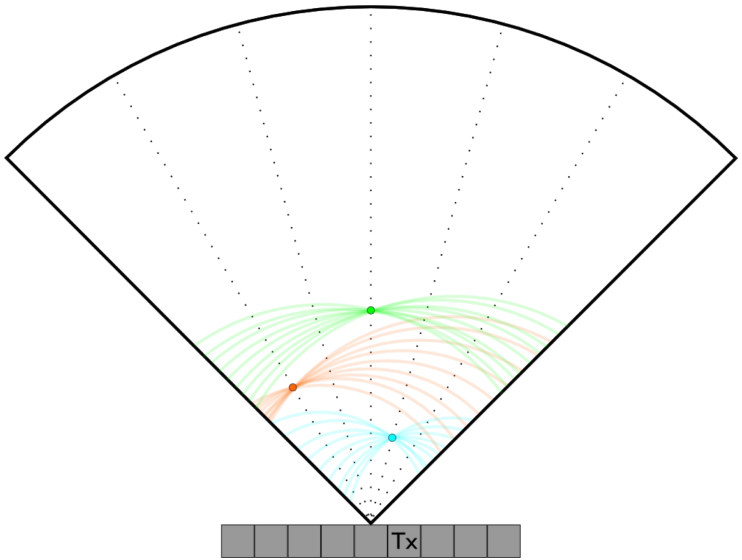


Image reconstructed from data based on round trip delay

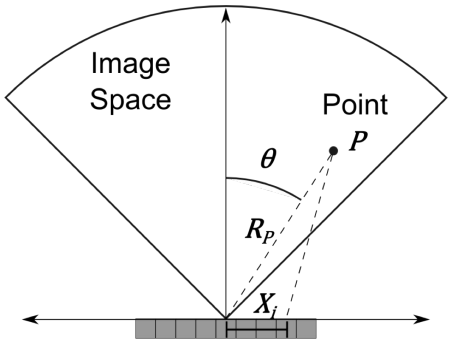
Ultrasound: Image Reconstruction



Images from each transducer combined to produce full frame

Delay Index Calculation

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

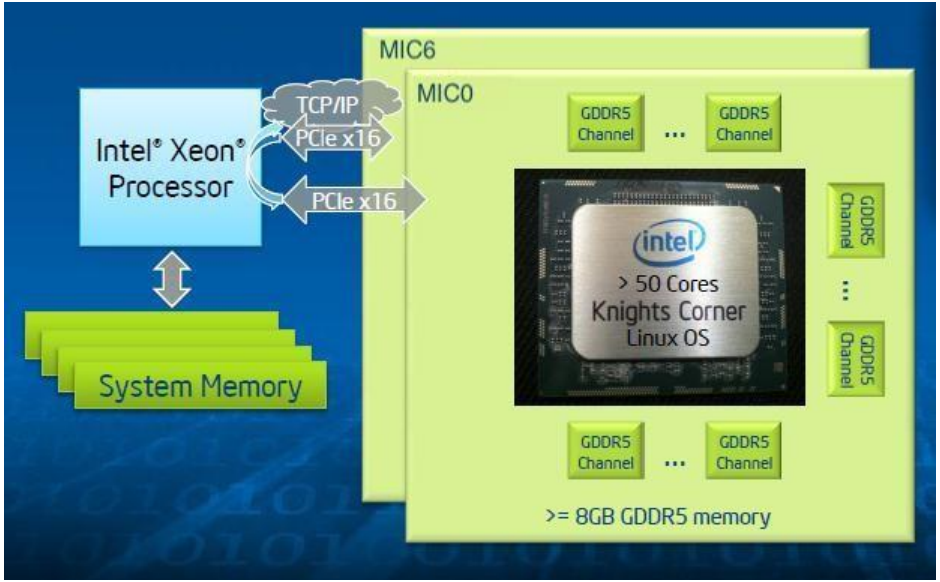


$$\tau_P = \frac{f_s}{c}\left(R_p + \sqrt{R_p^2 + X_i^2 - 2R_p X_i \sin\theta}\right)$$

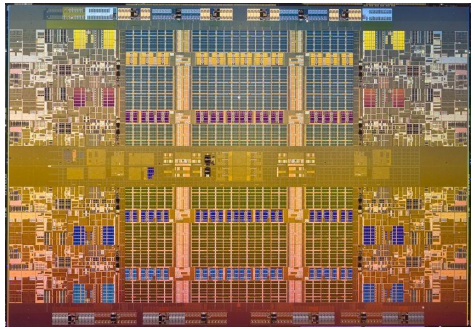
- Often done with lookup tables (LUTs) instead
- 50 GB LUT required for target 3D system


intel®
inside™
Xeon Phi™

Intel Xeon Phi Coprocessors and the MIC Architecture



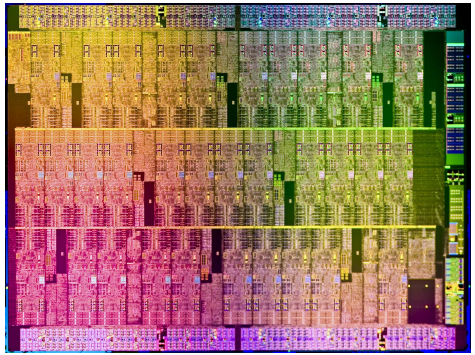
Intex Xeon Processors and the MIC Architecture



Multi-core Intel Xeon processor

- C/C++/Fortran; OpenMP/MPI
- Standard Linux OS
- Up to 768 GB of DDR3 RAM
- ≥ 12 cores/socket ≈ 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 ≈ 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

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% _
```

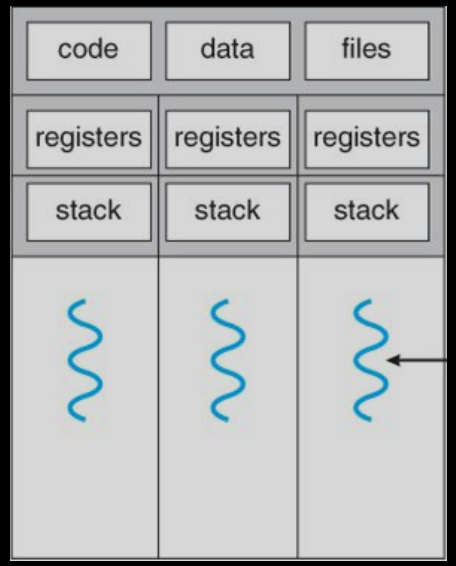
Native Execution

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)



POSIX Threads (Tutorial)

SIMD Operations

SIMD — Single Instruction Multiple Data

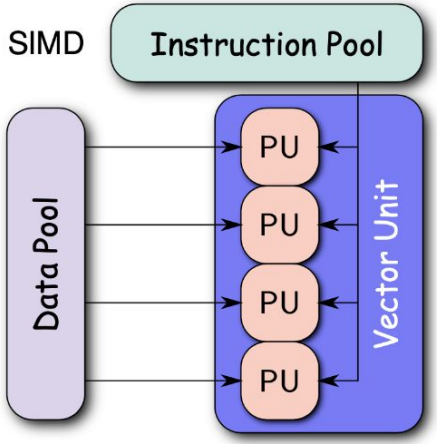
Scalar Loop

```
for (i = 0; i < n; i++)
  A[i] = A[i] + B[i];
```

SIMD Loop

```
for (i = 0; i < n; i += 4)
  A[i:(i+4)] = A[i:(i+4)] + B[i:(i+4)];
```

Each SIMD addition operator acts on 4 numbers at a time.



Bonus

software.intel.com/sites/landingpage/IntrinsicsGuide/

Instruction Sets in Intel Architectures

Instruction Set	Year and Intel Processor	Vector registers	Packed Data Types
MMX	1997, Pentium	64-bit	8-, 16- and 32-bit integers
SSE	1999, Pentium III	128-bit	32-bit single precision FP
SSE2	2001, Pentium 4	128-bit	8 to 64-bit integers; SP & DP FP
SSE3–SSE4.2	2004 – 2009	128-bit	(additional instructions)
AVX	2011, Sandy Bridge	256-bit	single and double precision FP
AVX2	2013, Haswell	256-bit	integers, additional instructions
IMCI	2012, Knights Corner	512-bit	32- and 64-bit integers; single & double precision FP
AVX-512	(future) Knights Landing	512-bit	32- and 64-bit integers; single & double precision FP

Explicit Vectorization: Compiler Intrinsics

SSE2 Intrinsics

```
for (int i=0; i<n; i+=4) {
  __m128 Avec=_mm_load_ps(A+i);
  __m128 Bvec=_mm_load_ps(B+i);
  Avec=_mm_add_ps(Avec, Bvec);
  _mm_store_ps(A+i, Avec);
}
```

IMCI Intrinsics

```
for (int i=0; i<n; i+=16) {
  __m512 Avec=_mm512_load_ps(A+i);
  __m512 Bvec=_mm512_load_ps(B+i);
  Avec=_mm512_add_ps(Avec, Bvec);
  _mm512_store_ps(A+i, Avec);
}
```

- The arrays `float A[n]` and `float B[n]` are aligned on a 16-byte (SSE2) and 64-byte (IMCI) boundary
- `n` is a multiple of 4 for SSE and a multiple of 16 for IMCI
- Variables `Avec` and `Bvec` are
  $128 = 4 \times$ `sizeof(float)` bits in size for SSE2 and
  $512 = 16 \times$ `sizeof(float)` bits for the Intel Xeon Phi architecture