PHI-VSIPL

Vector Signal Image Processing Library (VSIPL) for the Intel PHI Family of Microprocessors

Anderson, Dai, He, Liu, Qiao

VSIPL Summary

VSIPL is a matrix math and signal processing library developed to enable rapid-prototyping of applications (~225 fns).

- **Scalar operations (sin, cos, sqrt, etc)**
- **Vector operations (sin, cos, mul, div, etc)**
- **Matrix operations (mul, div, scatter, gather, etc)**
- **Signal processing**
- **Linear Algebra solvers**

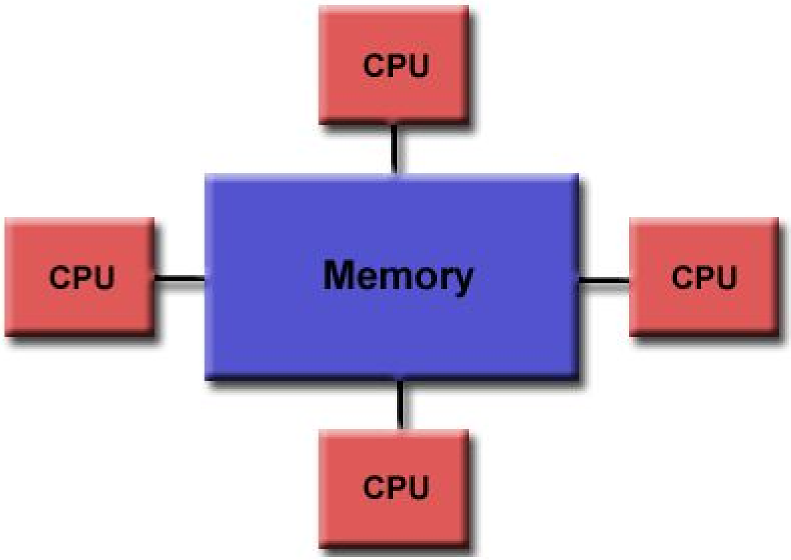
Example uses:

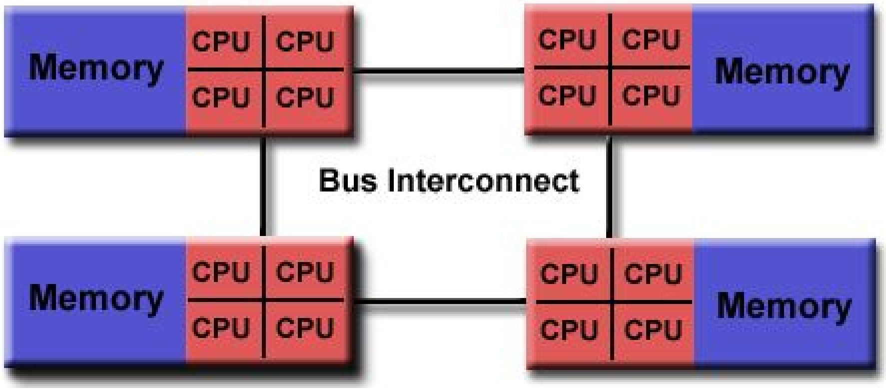
RADAR → Fast Fourier Transform, FIR Filter, Convolution Filter

Spacecraft navigation → LU, Cholesky Decomposition

Parallelized with OpenMP using Intel's thread pool model

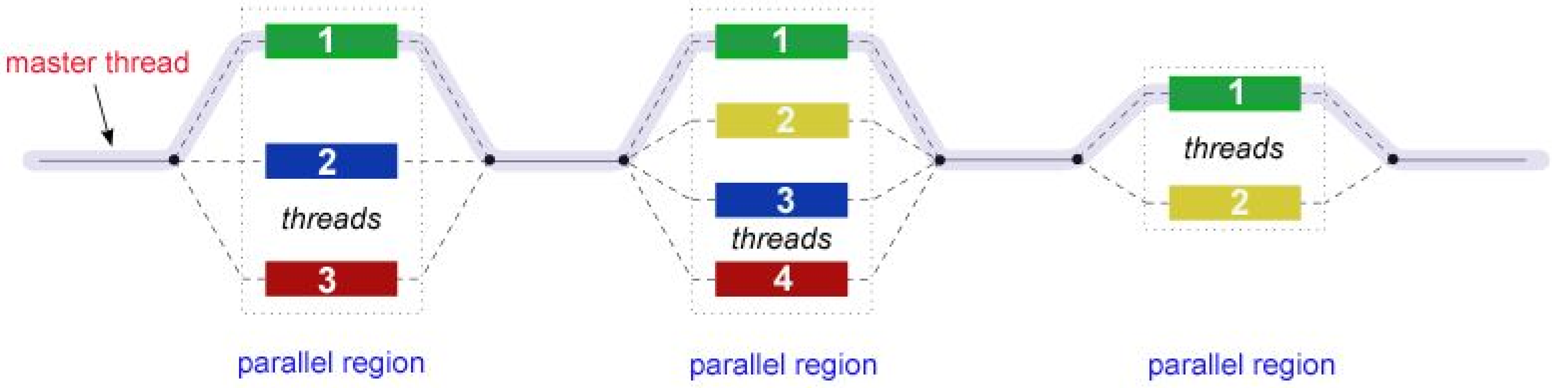
OpenMP





Both NUMA and UMA can use OpenMP.

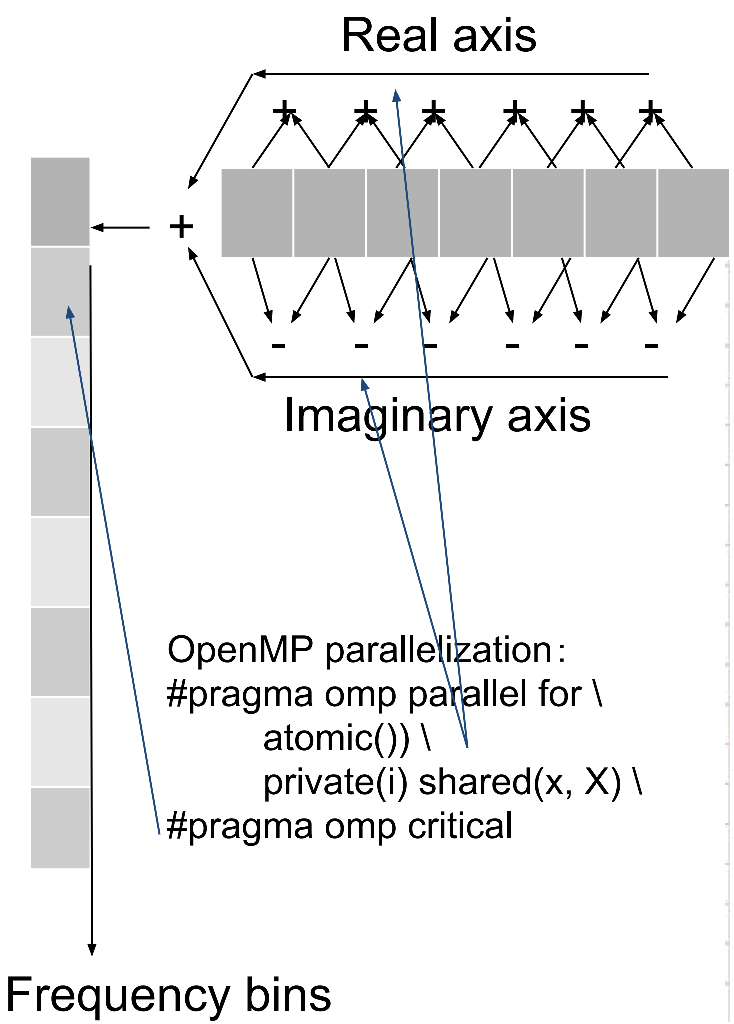
OpenMP



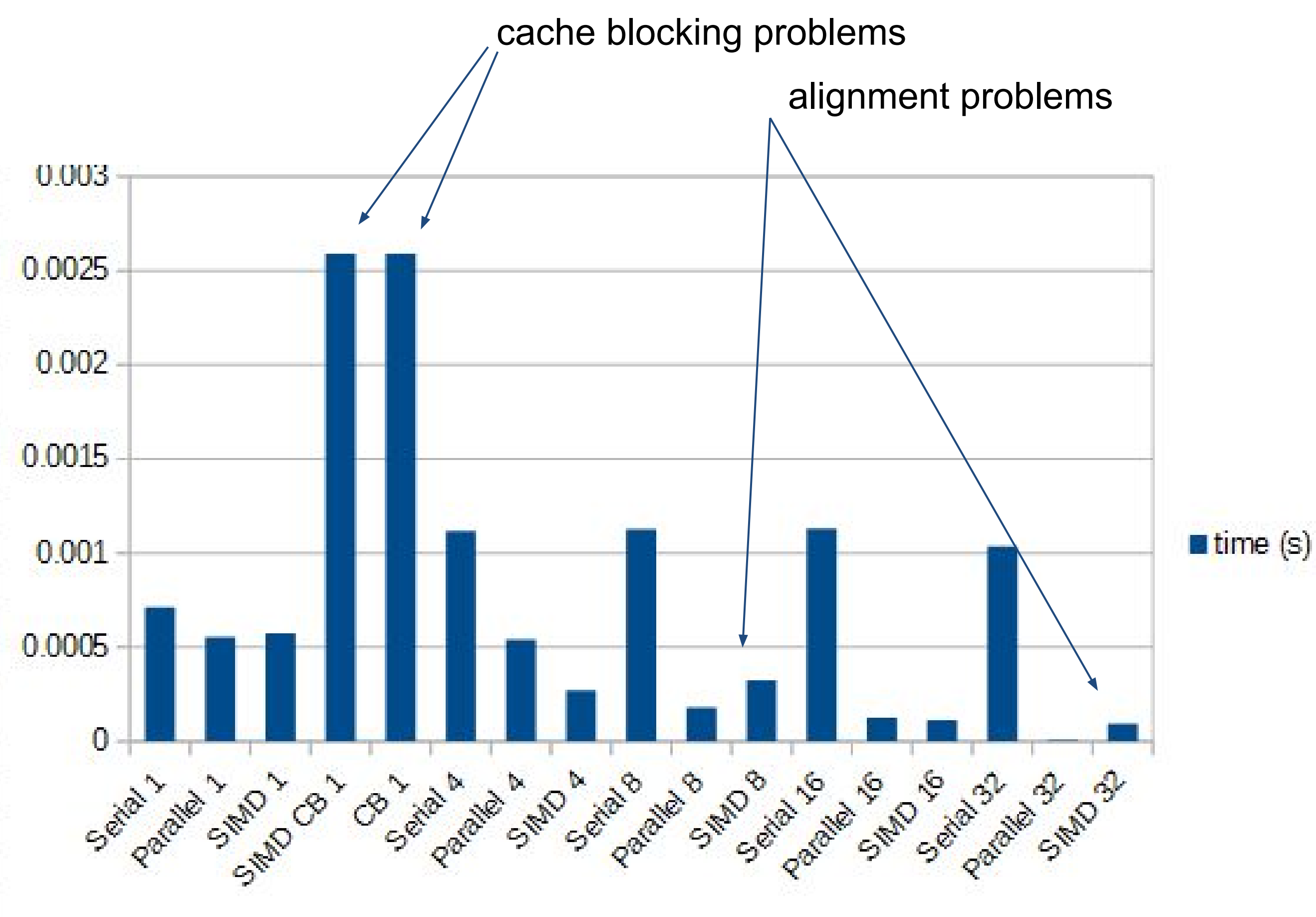
Fork and join model

OpenMP Parallelism Results

Vector operation (DFT real calc): Xre[k] += x[n] * cos(n*k*2Pi)




```
OpenMP parallelization :
#pragma omp parallel for \
        atomic()) \
        private(i) shared(x, X) \
#pragma omp critical
```

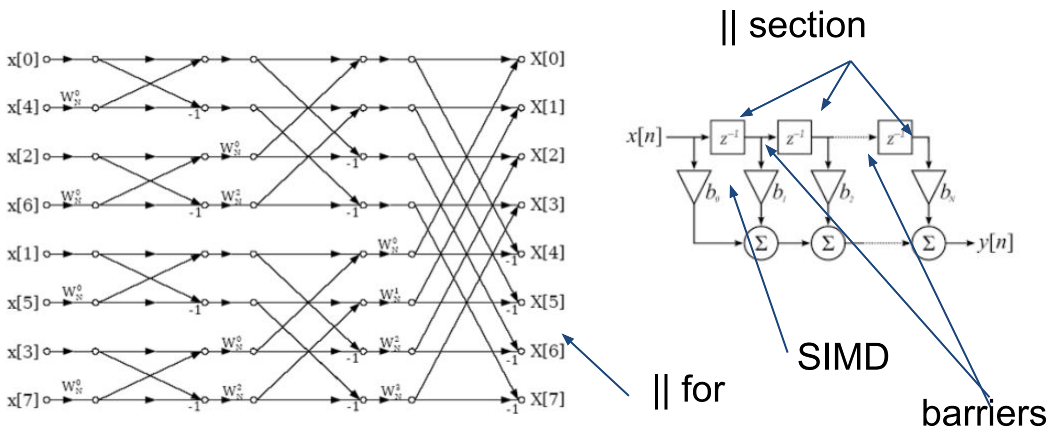


Signals: Fast Fourier Transform and Filters

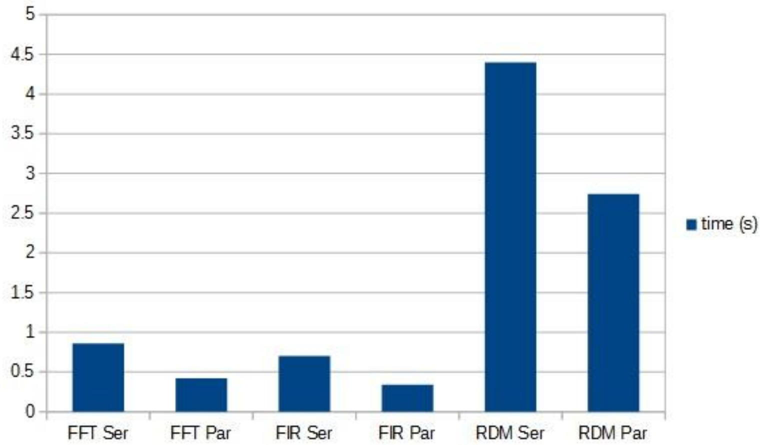
Problem

Parallelize Fast Fourier Transform and FIR filter for use in range doppler matrix software.

Implementation



Test Results



Summary

FFT parallelized for local machine up to 256 threads with 512 bits SIMD per thread. Larger arrays parallelized off-chip with targeted offloading (openMP 4.3).

GA Tech RDM software speedup = 62%.

-FFT not fully optimized

-Some vector functions were serial

Signals: Convolution and Correlation

Problem

Parallelize

Convolution(1D) $\int_{-\infty}^{\infty} f(\tau)g(x-\tau)\,\mathrm{d}\tau$

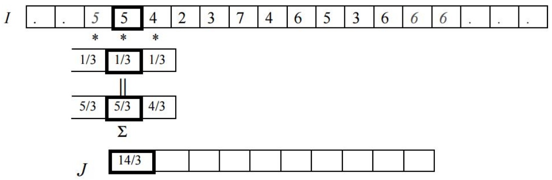
vsip_correlate1d_f_para

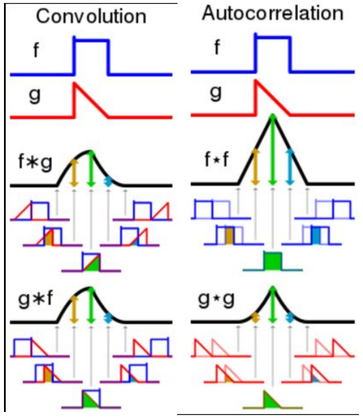
and

Correlation(1D, both Real and Complex)

vsip_correlate1d_f_para, vsip_ccorrelate1d_f_para

Implementation

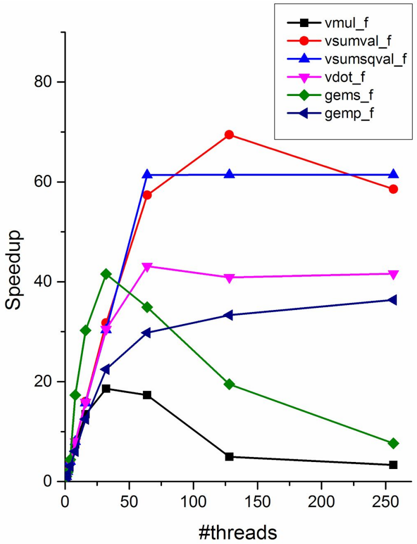


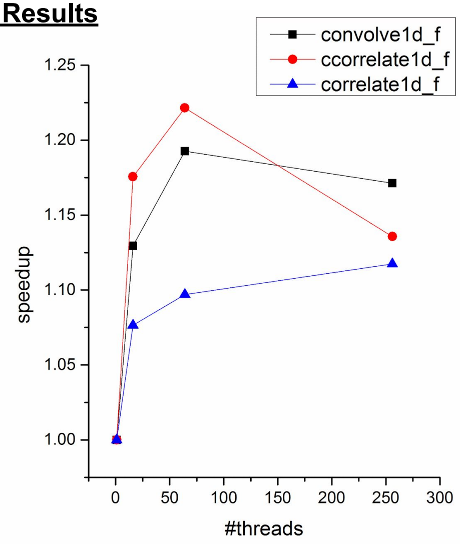


OpenMP parallelization:

```
#pragma omp parallel for \
        reduction(+:var) \
        private(i) shared(j) \
#pragma omp critical (a) {}
```

Test Results





Summary

- simple functions like vector or matrix multiplication nearly linear speedup
- speedup drops after 64 threads(Xeon-Phi)
- complex function speedup performance not quite good, possible reasons:
  - pre-processing of input data are sequential
  - inside loops with break statements, or loop end condition changes while running
  - complex loop (more than 2 layers)
  - data structures are not fit for parallelization, e.g. have to calc address before access an array element

More for this part:
https://github.com/daiszh/TASP_VSIPL_Core

Sizhang Dai

Special Solver: LU Decomposition

Problem

- Parallelize the LU decomposition function in the VSIPL using the OpenMP.
- lud_f & lusol_f
- a key step when inverting a matrix, or computing the determinant of a matrix

Implementation

A=LU: $$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

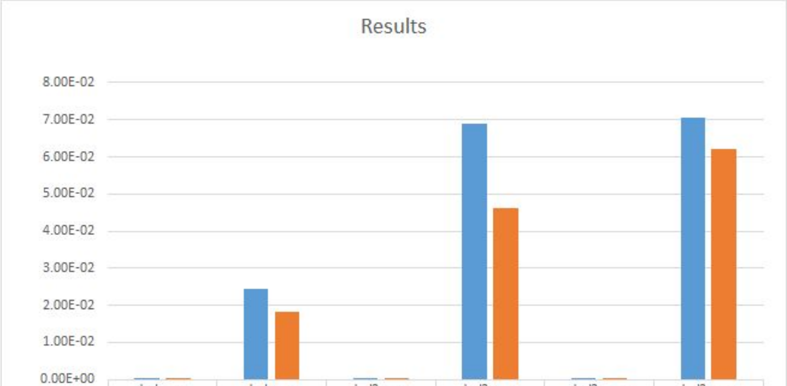
4 steps in the LU decomposition in VSIPL:

1. Created a LUD object
2. the LUD object and the matrix to be decomposed are passed into the decomposition function
3. the matrix equation is solved
4. the LUD object is destroyed

OpenMP Parallelization:

#pragma omp parallel for reduction() / private() / firstprivate(), etc.

Test Results



	ludex_o	ludex_p	lud2ex_o	lud2ex_p	lud3ex_o	lud3ex_p
lud	2.00E-06	2.43E-02	1.00E-06	6.88E-02	2.00E-06	7.07E-02
lusol	2.00E-06	1.83E-02	4.00E-06	4.63E-02	3.00E-06	6.20E-02

Summary

- ❖ The results should show some speedup if we have larger data input and using more threads
- ❖ There are many loops are very hard to parallelize because the structures are not fit for parallelization.
  - ➢ for example data dependencies and race condition
- ❖ The other small functions like vector are more easy to parallelize

Special Solver: Cholesky Decomposition

Problem

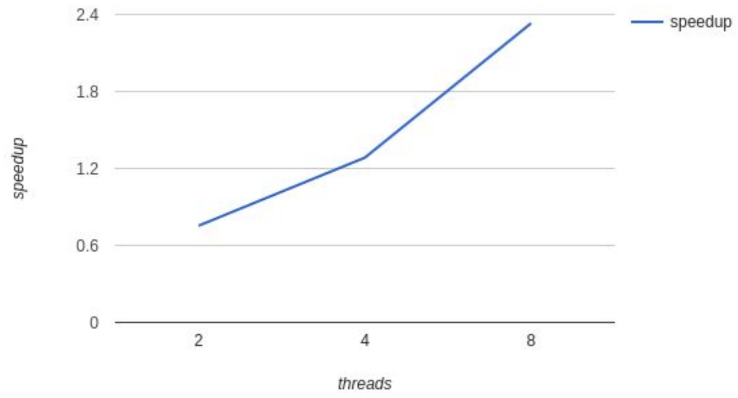
Use OpenMP to parallel the Cholesky Decomposition.

Implementation

#pragma omp parallel for

parallel the process for each column

Evaluation



Summary

1. **for large data use heap rather than stack**
2. **Avoiding data dependencies and race conditions**

Special Solver: QR Decomposition

Problem

- ❏ Parallelize QR Decomposition using OpenMP
- ❏ vsip_qrd_f (QRD operation)
- ❏ vsip_qrsol_f(determine if problem is solved)

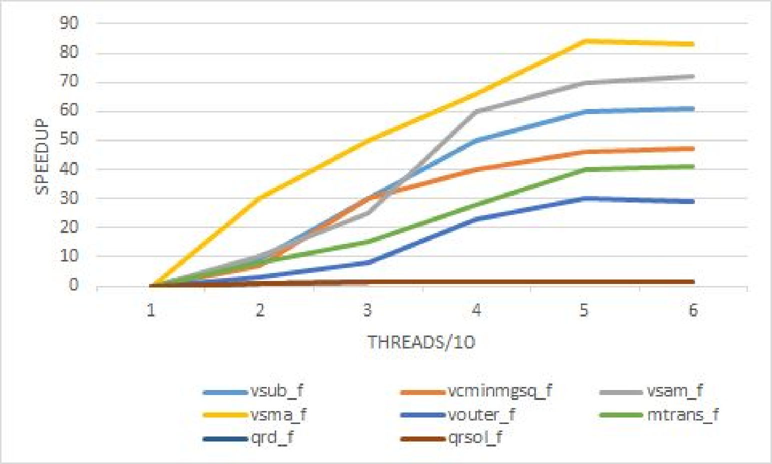
Implementation

★ Replace or modify loop structure using omp command:

- ○ #pragma omp parallel for
- ○ #pragma omp parallel for private ()
- ○ reduction(operator:list) and etc.

★ using function as omp_get_wtime() and etc. to test speedup performance

Test Results



Summary

- ❏ In matrix parallel,each processor works with a subset of the columns with the column cyclic distribution, improving memory access bandwidth and data locality.(all functions have positive speedup)
- ❏
- ❏ simple functions almost linear speedup but all dropped at certain amount of threads
- ❏
- ❏ complex functions just have small speedup

Summary

Approximately 60 VSIPL functions parallelized.
50% signal processing done
40% linear algebra solvers done
10% vector operations done
50% matrix operations done

Infinitely scalable (targeted offloading between processors)

Order of magnitude reduction in execution time when parallelizing

Future work:
Fix cache blocking/data alignment for SIMD instructions

Want More Details?

BACKUP

How Was Parallelism Implemented?

Matrix and Vector memory management:

Cache blocking for matrix → 64 byte cache blocks on PHI

D	D	D	D	D	D	F	F
D	D	D	D	D	D	F	F

Data, followed by fill up to the cache line size

Memory alignment for vector → SIMD requires alignment

Parallel for, SIMD, reduction operators used to parallelize

Targeted offloading → for data larger than the 256-thread pool

Range Doppler Matrix

Ranges are binned based on time to reflect.

Bins are FFT'd and filtered, and Doppler shift is used to track movement of reflector with respect to source.

Parallelization:
FFT-> parallel for (|| execution)
FIR -> parallel section
OMP barrier (pipeline)