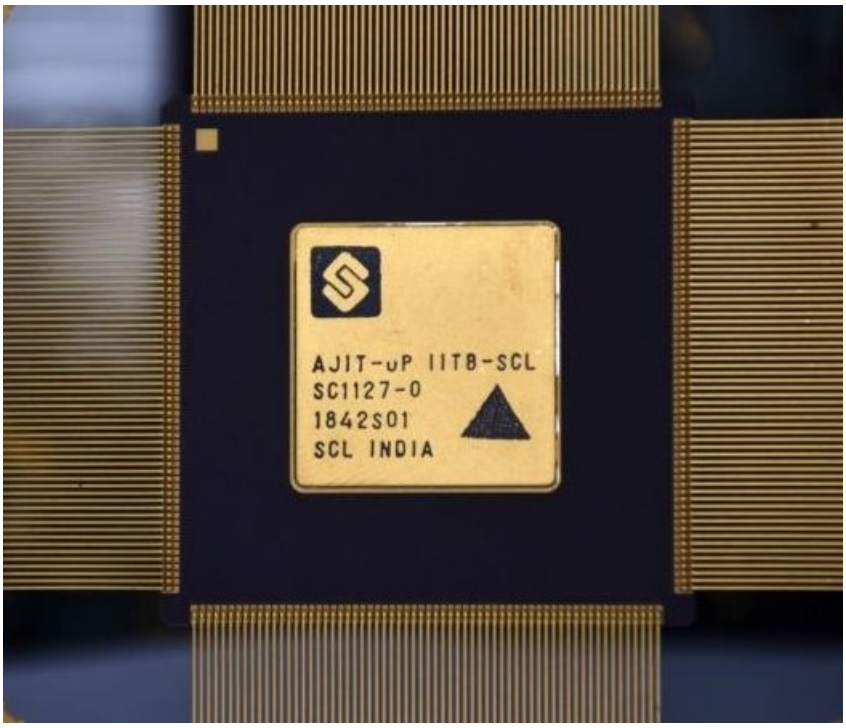
e-YSIP 2021

Implementation of AI/ML and DSP libraries using AJIT vector instructions



Interns:
Ayush Mittal
Krithik Sankar
Mentors:
Dr. Madhav Desai
Gauri Patrikar
Duration of Internship: 20/05/2021 – 08/07/2021

2021, e-Yantra Publication

Implementation of AI/ML and DSP libraries using AJIT vector instructions

Abstract

The AJIT processor (designed in IIT Bombay) has vector instructions for integer (8/16/32/64) bit and floating point arithmetic (half/single/double precision). Vector or SIMD instructions [1] are part of computers architecture with multiple processing elements that perform the same operation on multiple data points simultaneously. The project involves design of libraries that exploit these vector instructions for computations which occur in AI/ML and DSP applications.

Completion status

- Implemented and optimised Level 1 BLAS (Basic linear algebra subprogram) functions using vector instructions.
- Bench marked the implementation against GNU Scientific library.
- Optimized the functions to achieve 40% - 88% improvement in performance on different implementations.
- Developed applications such as FFT and Convolution using vector instructions.
- Completed Documentation.

1.1 Software used

- Ajit toolchain for Sparc v8 on Ubuntu 16.04

  - Detail of software: GitHub link.
  - Installation: follow steps in Github repo

1.2 Implementation

Github link for the repository of code.

BLAS Level-1

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing optimized linear algebra vector and matrix operations. Level-1 BLAS perform scalar, vector and vector-vector operations. The following level-1 BLAS functions were implemented using vector instructions and optimized for the AJIT processor:

- **Xasum** : sum of absolute values of the elements of an array
- **Xscal** : scales the elements of the array by constant alpha; $x = \alpha * x$
- **Xaxpy** : implements the equation $y = \alpha * x + y$ on each element of the array
- **Xdot** : computes the dot product between two arrays; $ans = \sum x_i * y_i$
- **Isamax** : finds the largest element of the array

Note: Each of the mentioned functions have separate implementations for various data types. The functions are mentioned as $X\{function\_name\}$. Here X represents different data types (for example: u8asum, u16asum, u32asum, sasum are implemented variants of Xasum)
u8 - 8 bit unsigned int
u16 - 16 bit unsigned int
u32 - 32 bit unsigned int
s - single precision floating point

DSP Routines

Using the optimizations from the BLAS implementations, applications such as DSP and convolution were implemented and optimized for AJIT. These applications were tested on audio and image samples respectively.

- **Fast Fourier Transform (FFT)** : Implemented Cooley Turkey FFT algorithm (Decimation in Time) with the help of vector instructions
- **Convolution** : Implemented 1-D and 2-D convolution (3x3 kernel) using vector instructions

The programs are written using C and assembly language taking advantage of the **Sparc ABI** [2]. Significant efforts have been made to redesign the function logic such that more data elements can be processed simultaneously with the help of **SIMD instructions** [1]. Observations show that the loops in the function have the most prominent effect on the overall performance. Vectorizing the arithmetic calculations, efficient pointer operations and pre-processing of data elements simultaneously are some of the optimizations which helped us to overall decrease the number of instructions executed to implement the same logic.

1.3 Results

BLAS Level-1

The implemented level-1 BLAS functions were compared against GNU scientific library implementations. Figure 1.1 shows the benchmark results for each BLAS function implementation against its GSL counterpart. Analysis of the table shows that there is a minimum **41% decrease** in the number of instructions executed for the overall program. Also, there is a **87% decrease** in number of instructions executed in cases where 8 data elements are being processed simultaneously.

In Figure 1.1 (and Figure 2.1), **n** is the array length, **VecIns** and **VecCy** are number of instruction executed and cycle count estimate for our optimized vector implementation. **NVecIns** and **NonVecCY** are number of instruction executed and cycle count estimate for standard non-vector function implementation.

Function		Instruction executed(n=1000)					
Type	Data type	VecIns	NVecIns	n=1000	VecCy	NonVecCy	n=1000
xaxpy	float	3587	6098	**41.18%**	362751	645682	**43.82%**
xaxpy	u8	1108	6091	**81.81%**	108622	645745	**83.18%**
xaxpy	u16	2094	6095	**65.64%**	205018	645959	**68.26%**
xaxpy	u32	3339	6095	**45.22%**	345073	645259	**46.52%**
xdot	float	3337	7063	**52.75%**	309441	643497	**51.91%**
xdot	u8	1113	7062	**84.24%**	99177	643356	**84.58%**
xdot	u16	2092	7064	**70.39%**	186046	643498	**71.09%**
xdot	u32	3336	7061	**52.75%**	309160	643285	**51.94%**
xasum	float	3570	9061	**60.60%**	290564	715355	**59.38%**
xasum	u8	1447	11056	**86.91%**	113511	856930	**86.75%**
xasum	u16	2822	11056	**74.48%**	219816	856930	**74.35%**
xasum	u32	4574	9150	**50.01%**	361708	721604	**49.87%**
xscal	float	3073	6066	**49.34%**	291117	573620	**49.25%**
xscal	u8	948	6061	**84.36%**	88302	573685	**84.61%**
xscal	u16	1828	6061	**69.84%**	167582	573125	**70.76%**
xscal	u32	3074	6064	**49.31%**	290768	573058	**49.26%**
isamax	float	9068	12065	**24.84%**	715992	928639	**22.90%**

Figure 1.1: Performance comparison between the optimized implementation and GNU scientific library implementation of BLAS Level-1 functions compiled with o3 optimization flag

DSP ROUTINES

Fast Fourier Transform (FFT) was implemented using **Cooley Tukey (Decimation in Time) algorithm**. Figure 1.2 shows the benchmarking results for the same. Analysis of the table shows that there is a **29% decrease** in the number of instructions executed and estimated cycle count for the vectorized implementation.

FFT (complex)						
N-Point FFT	VecIns	NVecIns	% decrease in instructions	VecCy	NonVecCy	% decreas in cycles
512	51317	72359	**29.08%**	66459	87711	**24.23%**
1024	109889	156025	**29.57%**	125117	171393	**27.00%**
2048	236045	336459	**29.84%**	260179	360803	**27.89%**

Figure 1.2: Performance comparison between the optimized implementation of FFT algorithm and the implementation provided in the book [4]

The FFT function was tested on various inputs, one such example is demostrated below. Figure 1.3 shows the output for a 1024 complex point FFT of signal $y(t) = sin(2\pi \cdot 300 \cdot t) + sin(2\pi \cdot 200 \cdot t)$ sampled at 1023 Hz.

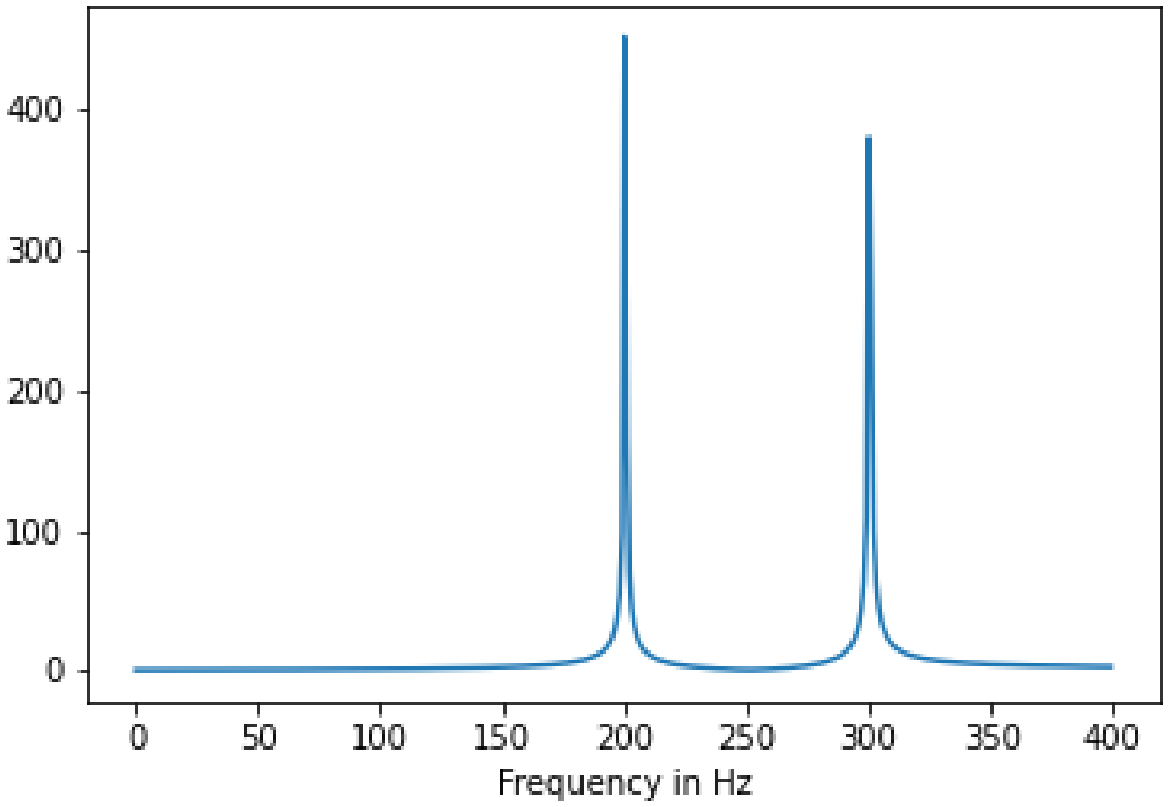


Figure 1.3: Output of FFT

1-D and 2-D convolution (3x3 kernel) were implemented using vector instructions. These functions were implemented for u8 (unsigned 8 bit int) and s16 (signed 16 bit integer) data types. These functions showed **50-60% reduction** in number of instructions and estimated cycle count against naive implementation.

Convolution functions were tested on various test data and images. An example of a 3x3 valid convolution is shown in Figure 1.4:

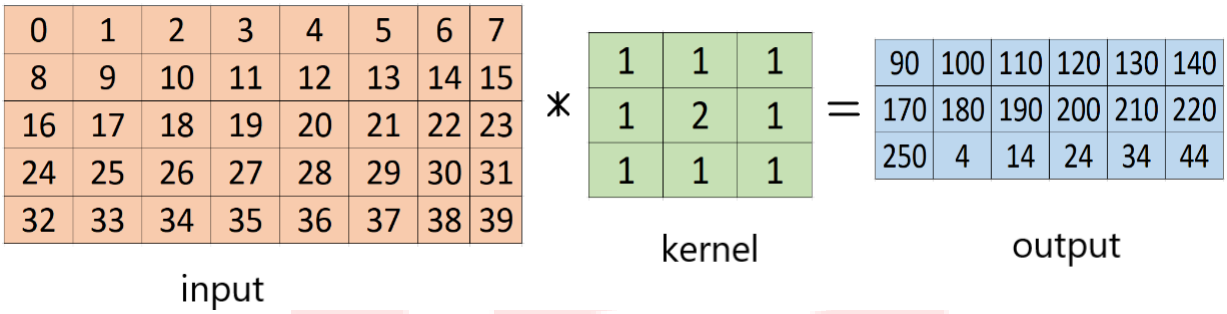


Figure 1.4: Output of unsigned 8-bit convolution (the result of larger are smaller as the range is 0-255, as it is 8 bits)

1.4 Future Work

- Test and verify functions and applications on FPGA implementation of AJIT processor
- Instruction Pipelining to further enhance the performance
- Development of utility functions that would help the user to define the data/array, aligned to 8 byte boundary which is currently accomplished with the help of data.s file

1.5 Bug report and Challenges

Some of the challenges faced were:

- The necessity of double word alignment of data in order to use double load (ldd) and store (std) instructions
- Debugging the code in an efficient way
- Limitations due to bare-bones compiler

Bibliography

[1] Madhav Desai, *64-bit ISA extensions to the AJIT processor, version 2*

[2] Sanjose Mary, Gauri Patrikar, *Calling an Assembly funciton from a C Program using the SPARC ABI*, 2019

[3] Richard P. Paul, *SPARC architecture assembly language programming, and C*, 1994

[4] William H. Press, Brian P. Flannery, Saul Teukolsky, William T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 1986.