Fast Fourier Transform in RISC-V Assembly

Team Syzgy-27172 Saad Thaplawala 24384 Sharjeel Chandna 29068 Saad Inam 29060 Hassan Jabbar

***Abstract*—This report details the complete journey of implementing a 1D Fast Fourier Transform (FFT) in RISC-V Vector Assembly. The report includes the stages of planning, division of tasks, problems encountered during implementation, and strategies adopted for overcoming issues, with a focus on bit-reversal, twiddle factor computation, and the butterfly operation.**

I. Introduction

Fast Fourier Transform (FFT) is a fundamental algorithm in digital signal processing. Implementing it in low-level assembly, particularly on RISC-V vector architectures, provides deep insight into computer architecture and performance optimizations. Our project aimed to implement a 1D FFT in scalar and vector RISC-V assembly.

II. Team Strategy and Initial Problems

Initially, we divided the tasks among team members as follows:

- Bit traversal and bit-reversal permutation
- Twiddle factor computation
- FFT butterfly stage implementation

However, we quickly faced a significant challenge — these components were uneven in difficulty and not modular due to inter-dependencies. This led us to regroup and understand FFT holistically using Saad's DAA notes and several YouTube resources. Despite gaining theoretical clarity, implementing FFT in assembly proved much more complex.

III. Scalar FFT with Hardcoded Elements

Our first working version of FFT used scalar RISC-V assembly and included hardcoded twiddle factors and bit-reversal mappings for $N = 8$. While this version produced correct results, it lacked scalability and failed the project requirement of runtime computation.

IV. Scalable Bit Reversal

To make the FFT scalable, we implemented a runtime bit-reversal computation algorithm. The approach reversed bits for indices i from 0 to $N - 1$, where $N = 2^k$, storing results in a separate array. The scalar assembly code ensured the bit-reversal permutation could dynamically accommodate different input sizes.

V. Runtime Twiddle Factor Calculation

This was the most problematic stage. Our goal was to compute twiddle factors $W_N^k = e^{-2\pi ik/N}$ at runtime for each FFT stage. We faced two major issues:

1) Encoding constants like $\cos(2\pi k/N)$ and $\sin(2\pi k/N)$ for multiple values at each stage.
2) Maintaining precision during floating point multiplication, especially in complex multiplication.

To bypass full dynamic computation (which is extremely complex in pure assembly), we precomputed common values such as ± 1, ± 0.7071 for $N = 8$ in IEEE-754 hex and used conditionals for selection. This approach worked correctly but again limited scalability beyond $N = 8$.

VI. Butterfly Computation

With correct twiddle factors, the butterfly logic was implemented as follows:

- Load real and imaginary values for even and odd indices
- Compute $W \times$ odd using complex multiplication
- Update the even index with even $+ W \cdot$ odd
- Update the odd index with even $- W \cdot$ odd

This logic was verified for correctness at each stage using known FFT test vectors. The computed output matched expected results for $N = 8$.

VII. Challenges

- **Twiddle Factor Issues:** Errors in floating-point precision and incorrect indexing for stage 2 (when $N > 4$) caused incorrect results.
- **Instruction Set Limitation:** Lack of higher-level math functions like sin/cos made real-time twiddle computation impractical.
- **Debugging Complexity:** Debugging was time-intensive due to lack of descriptive output in simulation.

VIII. Conclusion and Future Work

We successfully implemented a scalar 1D FFT in RISC-V assembly for $N = 8$ with runtime bit-reversal and partial runtime twiddle computation. While the solution isn't fully scalable yet, it provides a functional baseline. Future work includes:

- Full dynamic twiddle computation using lookup tables or series approximations.
- Vectorized implementation for performance boost.
- Testing for larger N and improving modularity.

Acknowledgments

We extend our heartfelt thanks to our Teaching Assistant, especially Abdul Wasay, for his continuous guidance and detailed technical support throughout the project. We also thank our peers for insightful discussions and help during debugging. Special thanks to YouTube tutorials on FFT visualization and RISC-V vector assembly, which were crucial in helping us understand the math and hardware-level implementation.