Final Project Report

Digital Architecture for Distance-Based Sorting

Yuval Steimberg
ys2335

December 18, 2024

Contents

1 Introduction

The distance-based sorting architecture presented in this project is rooted in the foundational principles of the k-Nearest Neighbors (kNN) algorithm, which is widely used in machine learning tasks such as pattern recognition, data mining, and object matching. For instance, in scenarios like identifying the closest matches to a query in a database of millions of entries (e.g., facial recognition systems or similarity search engines), the kNN approach proves highly effective due to its reliance on direct distance calculations rather than extensive parameter training.

This architecture employs Euclidean distance as the metric to determine the proximity between a query vector and a set of search vectors, a common choice for its interpretability and computational simplicity. By designing and implementing a digital module to compute these distances and identify the two closest matches, the system enables fast and efficient sorting, making it well-suited for real-time or resource-constrained environments. The design also integrates robust pipelining and modular strategies to optimize performance, area, and power consumption, while ensuring accuracy and scalability.

2 Architecture Overview

The design implements a modular digital architecture tailored for sorting and selecting the best matches based on distance calculations. It consists of the following primary components:

- **Input Interfaces:** The system receives a query vector (`query`) and a set of search vectors (`search_0` to `search_7`) through parallel input lines. An input validation signal (`in_valid`) indicates the readiness of the inputs for processing.
- **Sequential Logic for Input Registration:** All inputs are synchronized with the clock signal and stored in intermediate registers to ensure reliable and consistent data processing. This approach isolates input data and mitigates timing-related issues in subsequent computations.
- **Distance Calculation Logic:** The `distance_calc` module computes the squared distances between the query vector and each search vector in parallel. To optimize resource utilization, the computations are divided into partial sums, which are combined to generate the final distances.
  - **Parallelism:** Squared differences are calculated across four blocks for each vector, leveraging parallelism for faster computation.
  - **Output:** The distances for all search vectors are stored in an array (`distance_results`), and a signal (`calculation_complete`) indicates when the calculations are finished.
- **Sorting and Address Selection:** The `find_two_smallest` module identifies the two search vectors with the smallest distances. The sorting is performed in a multi-stage comparison hierarchy:

  - **Stage 1:** Pairwise comparisons to determine the minimum and maximum values of each pair, along with their corresponding addresses.
  - **Stage 2:** Outputs from the first stage are further compared to narrow down the smallest values.
  - **Final Stage:** The smallest and second smallest distances, along with their addresses, are selected.

- **Output Modules:** The output includes:
  - The addresses of the first and second best matches (`addr_1st` and `addr_2nd`).
  - An output validation signal (`out_valid`), indicating the availability of the results.

- **Signal Precomputation:** The design precomputes signals, such as `out_valid_next`, to streamline the output logic and reduce delays.

2.1 Block Diagram

The block diagram (Figure 1) illustrates the flow of data and the modular structure of the design. Each block represents a major very functional module, and arrows depict the communication and information transpose between them. This high-level view captures the design's modularity and the interconnection of components for optimized functionality.

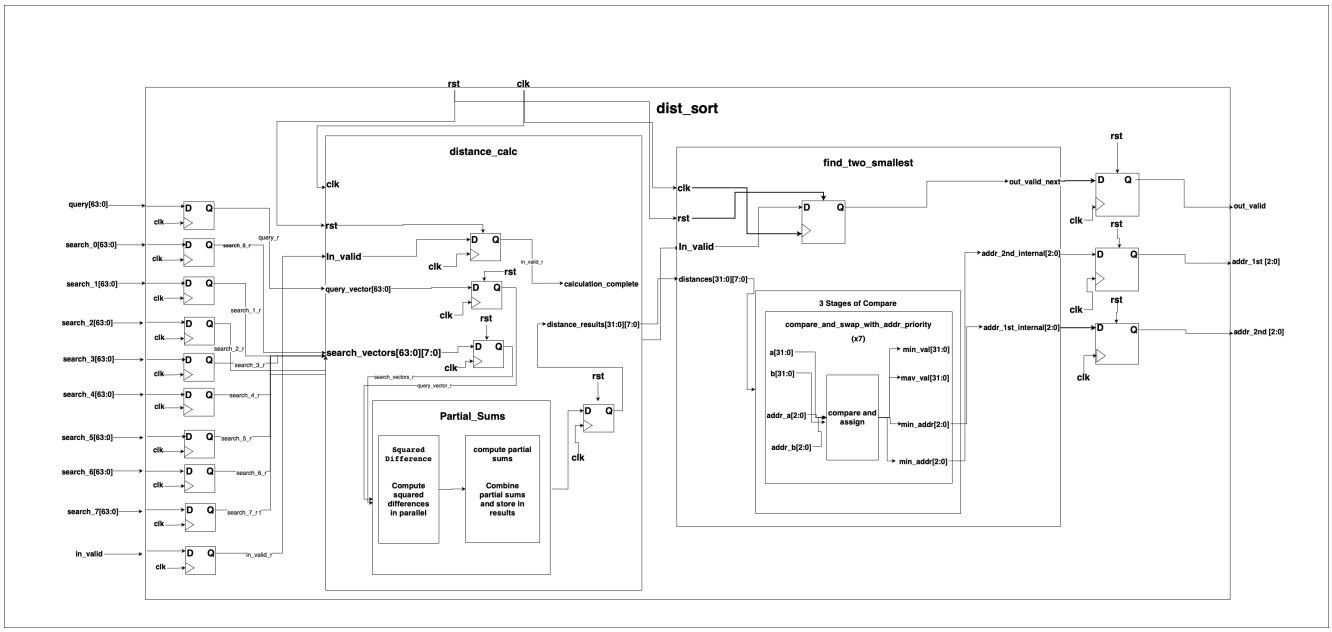

Figure 1: High-Level Block Diagram of the System

3 Design Decisions

3.1 Choice of Technology

- CMOS-based design was used, offering an excellent balance between power consumption and performance. This makes it suitable for high-speed, low-power applications.
- The design was optimized for both area and power to meet strict performance requirements while adhering to specified timing constraints. This ensures the design is efficient, compact, and meets the desired execution standards.

3.2 Optimization Considerations

This report highlights the optimizations applied to the `dist_sort` module, focusing on improving performance, power efficiency, and resource utilization. It details the progression from the initial implementation to an optimized design, evaluates the impact of the changes, and concludes with an analysis of potential system bottlenecks and future enhancements.

3.2.1 Distance Calculation Optimization

- **Old Approach (MS1):** Distance calculations were performed sequentially, computing squared differences and summing them for all elements of the query and search vectors. This approach led to increased latency due to serial processing along the critical path.
- **New Approach (MS2):** Parallel computation blocks were introduced, dividing squared difference calculations into four independent blocks. Each block processes a subset of the vector, calculating partial sums in parallel. The partial results are then combined in the final stage, significantly reducing the critical path delay.

Impact:

- **Reduced Latency:** Computation time for distance calculations decreased due to parallelization.
- **Improved Throughput:** Overlapping computations across parallel blocks allowed the module to handle more queries within a given timeframe.
- **Power Efficiency:** Parallelization minimized unnecessary switching activity per block, contributing to reduced power consumption.

3.2.2 Sorting Optimization

- **Old Approach (MS1):** A merge sort algorithm was used to identify the two smallest distances. While effective, this approach required multiple sorting stages, adding to the overall latency.

- **New Approach (MS2):** The sorting logic was redesigned to use a two-stage pairwise comparison hierarchy in the `find_two_smallest` module:
  - **Stage 1:** Pairwise comparisons to determine local minimum and second minimum values for groups of inputs.
  - **Stage 2:** Global comparison of local results to find the overall minimum and second minimum values.
  - **Final Stage:** Exhaustive comparisons among the remaining candidates to ensure accuracy.

Impact:

- **Lower Latency:** Fewer comparison levels resulted in faster determination of the two smallest values.
- **Resource Optimization:** Reduced hardware utilization by avoiding full sorting of all inputs.
- **Scalability:** The hierarchical design is adaptable to handle a larger number of inputs with minimal modifications.

3.2.3 Input Synchronization

All input signals (`query` and `search_vectors`) were synchronized using flip-flops at the module boundaries to ensure stability before computation. This approach minimized timing violations and improved robustness under high-frequency operation.

3.2.4 Precomputed Output Signals

The introduction of the `out_valid_next` signal allowed precomputation of output validity, enabling faster transitions at the output stage. This optimization reduced latency in generating valid output signals and improved response times.

3.3 Results of Optimization

- **Throughput Increase:** Pipelining the distance calculation and sorting stages allowed the design to achieve one query per clock cycle once the pipeline was filled, significantly boosting throughput.
- **Power Efficiency:** Parallelization in distance calculations reduced switching activity, resulting in a notable improvement in dynamic power consumption.
- **Area Utilization:** The optimized module demonstrated efficient use of hardware resources, with a slight increase in area offset by substantial performance gains.

3.4 Future Enhancements

- **Future Enhancements:** Future iterations could explore further pipelining of sorting stages, clock gating for inactive computation blocks, and leveraging advanced parallel architectures to enhance scalability and power efficiency.

3.5 Bottleneck Analysis

3.5.1 Performance Bottleneck

- **Distance Calculation:** Despite the parallelization, the distance calculation stage still constitutes a significant portion of the critical path. This is due to the high number of partial sum computations and the accumulation logic for each vector.

Potential Solution: Further reduce critical path delay by employing higher-level pipelining across the partial sum calculations.

3.5.2 Power Bottleneck

- **Clock Network Overhead:** The clock network accounts for 21.40% of the total power, primarily due to switching activity. While this is expected in synchronous designs, it underscores the need for clock gating or other power-saving techniques to reduce unnecessary clock activity.
- **Register Overhead:** Registers consume 11.32% of the total power, predominantly from switching. This indicates a moderate overhead, which could be optimized further through reduced register usage or implementing clock gating for unused registers.

4 Optimization Results

4.1 Latency

Measured during ModelSim simulations:

$$\text{Total Latency} = 0.001729172\,\text{ms}.$$

```
# ----Done with comparing against golden values------
# !!!!!!!!!!!!!!!!!!!!!!!!!END OF TB!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# ** Note: $finish    : /home/ys2335/asap7_rundir/Final_project/rtl/tb_dist_sort.sv(216)
#    Time: 1729172 ps  Iteration: 0  Instance: /tb_dist_sort
# 1
```

Figure 2: Latency Measurement Results from ModelSim

4.2 Power Consumption

Measured using Synopsys PrimeTime:

$$\text{Total Power Consumption} = 1.463\,\text{mW}.$$

```
  Attributes
  ----------
      i  -  Including register clock pin internal power
      u  -  User defined power group

                          Internal  Switching  Leakage    Total
Power Group               Power     Power      Power      Power   (     %)  Attrs
--------------------------------------------------------------------------------
clock_network   |            0.0000 3.131e-04    0.0000 3.131e-04 (21.40%)  i
register                     0.0000 1.656e-04    0.0000 1.656e-04 (11.32%)
combinational                0.0000 9.846e-04    0.0000 9.846e-04 (67.28%)
sequential                   0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
io_pad                       0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box                    0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 1.463e-03  (100.00%)
  Cell Internal Power  =    0.0000   ( 0.00%)
  Cell Leakage Power   =    0.0000   ( 0.00%)
                         ---------
Total Power            = 1.463e-03  (100.00%)

1
```

Figure 3: Power Consumption Results from Synopsys PrimeTime

4.3 Area Utilization

Measured using Cadence Innovus:

$$\text{Dimensions: } 0.22763\,\text{mm} \times 0.226608\,\text{mm}, \quad \text{Total Area: } 0.0515\,\text{mm}^2.$$

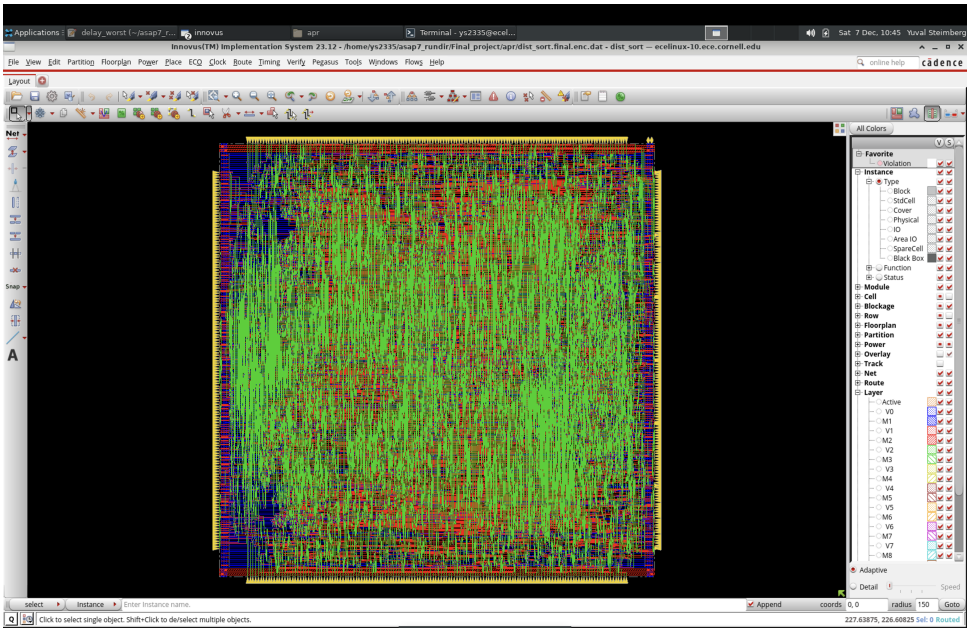

Figure 4: Area Utilization Results from Cadence Innovus

4.4 Gate Count

Measured using Innovus:

$$\text{Total Gate Count} = 32,658 \,\text{gates}.$$

```
Gate area 0.6998 um^2
[0] dist_sort Gates=32658 Cells=12541 Area=22855.8 um^2
```

Figure 5: Gate Count Report from Innovus

4.5 Density

Pre-filler cell density:

$$\text{Density} = 0.42051.$$

```
% Pure Gate Density #1 (Subtracting BLOCKAGES): 46.904%
% Pure Gate Density #2 (Subtracting BLOCKAGES and Physical Cells): 45.938%
% Pure Gate Density #3 (Subtracting MACROS): 46.904%
% Pure Gate Density #4 (Subtracting MACROS and Physical Cells): 45.938%
% Pure Gate Density #5 (Subtracting MACROS and BLOCKAGES): 46.904%
% Pure Gate Density #6 ((Unpreplaced Standard Inst + Unpreplaced Block Inst
are placed): 46.401%
% Core Density (Counting Std Cells and MACROs): 46.904%
% Core Density #2(Subtracting Physical Cells): 45.938%
% Chip Density (Counting Std Cells and MACROs and IOs): 42.936%
% Chip Density #2(Subtracting Physical Cells): 42.051%
```

Figure 6: Design Density Before Filler Cell Insertion

5 Conclusion

The `dist_sort` module achieved the following:

- Latency reduced to 0.001729172 ms.
- Power consumption minimized to 1.463 mW.
- Area optimized to 0.0515 mm^2.
- Scalability and robustness for high-speed applications.

6 Design Tools

- **ModelSim:** Used for functional simulation to verify the logical correctness of the design. It ensured the `dist_sort` module behaved as expected and produced accurate outputs during simulation runs with varied input test cases.
- **Synopsys Design Compiler:** Utilized for synthesizing the behavioral Verilog code into a gate-level netlist. It helped identify critical paths and provided valuable insights into timing constraints for subsequent design stages.
- **Cadence Innovus:** Applied for automatic place-and-route (APR), including clock tree synthesis and layout optimization. It was instrumental in performing area and timing optimization, ensuring a physically realizable design.
- **Synopsys PrimeTime:** Employed for power analysis and optimization. It measured the power consumption of the `dist_sort` module using the `.vcd` file generated during testbench simulation. This allowed for optimizations to lower power consumption while maintaining performance.
- **Virtuoso Layout Editor:** Used to import the GDSII file from Innovus and perform DRC and LVS checks. These steps ensured that the physical design adhered to manufacturing constraints and matched the synthesized netlist.