Design and Integration of an Accelerator for MRI with ESP

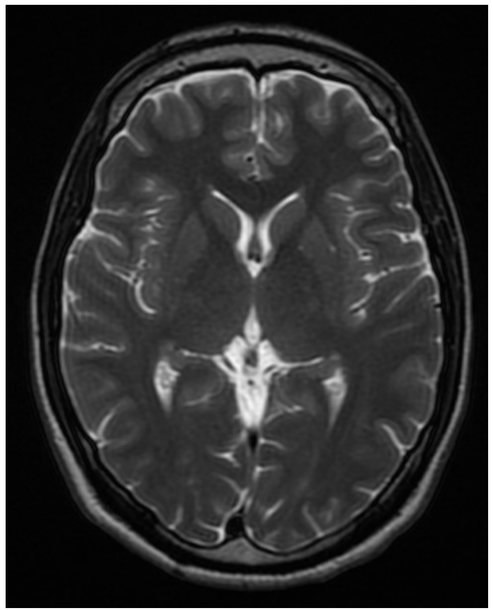
Pei Liu

Background

Magnetic **R**esonance **I**maging

Scanning

Image reconstruction



For the advanced MRI image reconstruction algorithm,
the Q-matrix computation is the most computation-intensive part

Datasets for MRI-Q from Parboil Benchmarks

Name	Image Size	# of pixels (numX)	K-space Dimension (numK)
small	32 x 32 x 32	32768 (32 K)	3072 (3 K)
large	64 x 64 x 64	262144 (256 K)	2048 (2 K)
128x128x128	128 x 128 x 128	2097152 (2048 K)	2097152 (2048 K)

Algorithm for computing the MRI-Q matrix

```
For( i = 0; i < numX; i++)
    For( k = 0; k < numK; k ++)
        expArg = 2π *(kx[k]* x[i]+ky[k]*y[i]+kz[k]* z[i])
        cosArg = cos( expArg );
        sinArg = sin( expArg );
        phiMag = phiR[k]^2 + phiI[k]^2
        Qracc += phiMag * cosArg
        Qiacc += phiMag * sinArg
    Qr[i] = Qracc
    Qi[i] = Qiacc
```

K-space : Frequency space

X-space : 3D sampling space

MRI-Q Accelerator Progress

- Stratus HLS SystemC Flow:
  1. Designed a set of accelerators that can process input data with arbitrary size
  2. Design Space Exploration: Pareto-optimal curve of 12 Designs
  3. Baremetal application passed test on FPGA
  4. Linux application tested on FPGA and showed acceleration

Stratus HLS Flow: 12 designs

- 2 dma sizes:
  - 32-bits working with Leon3
  - 64-bits working with Ariane
- 2 architectures :
  - A0 — storing all the k-space variables in the PLM
  - A1 — storing partial of k-space variables in the PLM
- 3 parallelism Level:
  - P4 — loop unrolling with factor = 4
  - P8 — loop unrolling with factor = 8
  - P16—loop unrolling with factor = 16

In hw/hls/project.tcl file, HLS configuration

```
################################################
# HLS and Simulation configurations
################################################
set DEFAULT_ARGV ""

foreach dma [list 32 64] {
    foreach arch [list 0 1] {
        foreach para [list 4 8 16] {
            define_io_config * IOCFG_P$para\_A$arch\
            define_system_config tb TESTBENCH_P$para
            foreach tb $TESTBENCHES {
```

Design Challenges

1. Determine the configuration parameters
2. Design the Sine function
3. Work with fixed-point data types
4. Design Space Exploration (DSE)

1. Configuration Parameters

- **batch_size_x** : batch size of the x-space variables loaded into the PLM
- **num_batch_x**: total number of batches of x variables to be loaded and computed

num_batch_x * batch_size_x = numX

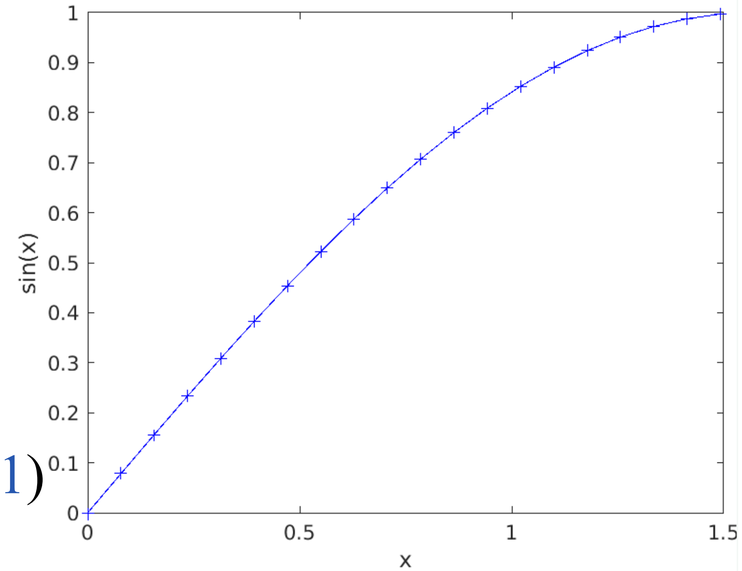
- **batch_size_k** : batch size of the k-space variables loaded into the PLM
- **num_batch_k**: total number of batches of k variables to be loaded and computed

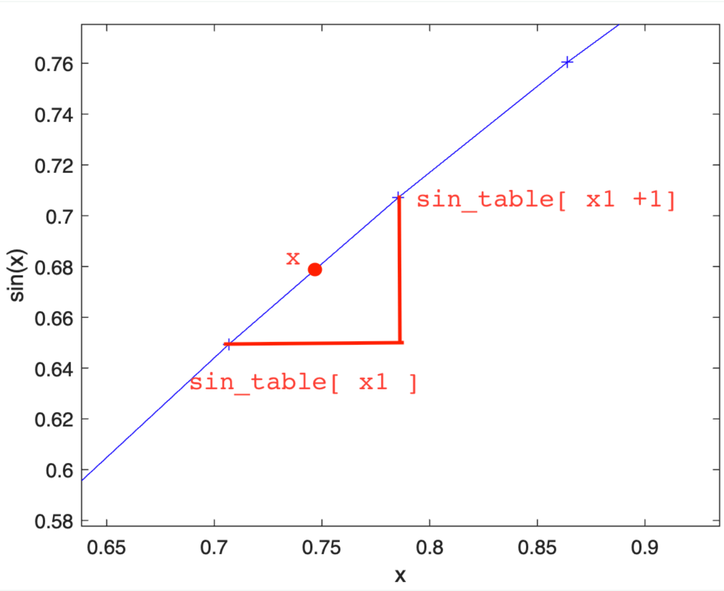
num_batch_k * batch_size_k = numK

Params	A0	A1
batch_size_x	128	128
num_batch_x	numX / 128	numX / 128
batch_size_k	numK	1024
num_batch_k	1	numK / 1024

2. Compute Sine and Cosine

1. Convert an input to [0 ~ $\frac{\pi}{2}$] (x)
2. Find the closet lower point to x (x1)
3. Get sin[x1] and sin[x1 + 1] from the look-up table.
4. Interpolate to get sin(x), cos(x) = sin($\frac{\pi}{2}$ - x)





sine look-up table with 20 points with x in [0, $\frac{\pi}{2}$]

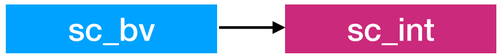
```
FPDATA_S sin_table[20];
HLS_FLATTEN_ARRAY(sin_table);

sin_table[0] = 0.000000000000;
sin_table[1] = 0.078459098935;
sin_table[2] = 0.156434476376;
sin_table[3] = 0.233445376158;
sin_table[4] = 0.309017002583;
sin_table[5] = 0.382683455944;
sin_table[6] = 0.453990519047;
sin_table[7] = 0.522498548031;
sin_table[8] = 0.587785243988;
sin_table[9] = 0.649448037148;
sin_table[10]= 0.707106769085;
sin_table[11]= 0.760405957699;
sin_table[12]= 0.809017002583;
sin_table[13]= 0.852640211582;
sin_table[14]= 0.891006529331;
sin_table[15]= 0.923879504204;
sin_table[16]= 0.951056540012;
sin_table[17]= 0.972369909286;
sin_table[18]= 0.987688362598;
sin_table[19]= 0.996917307377;
```

3. Work with Fixed-point data types

- Data Conversion

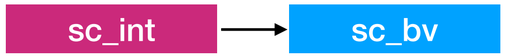
Testbench: *load_memory* function

Accelerator: *load_input* process



Accelerator: *compute_kernel* process

Accelerator: *store_output* process



Testbench: *dump_memory* function

4. Design Space Exploration

- Reduce Area:
  - Customize the PLM for different variables
  - Reduce the fixed-point data precision
- Reduce Latency:
  - Unroll and pipeline loops

Reduce Area:

PLM design is customized for different variables.

```
mriq_x_dma32 128 32 1w:0r 0w:1r
mriq_x_dma64 128 32 2w:0r 0w:2r

mriq_LK_P4_k_dma32 1024 32 1w:0r 0w:4r
mriq_LK_P8_k_dma32 1024 32 1w:0r 0w:8r
mriq_LK_P16_k_dma32 1024 32 1w:0r 0w:16r

mriq_SK_P4_k_dma32 3072 32 1w:0r 0w:4r
mriq_SK_P8_k_dma32 3072 32 1w:0r 0w:8r
mriq_SK_P16_k_dma32 3072 32 1w:0r 0w:16r

mriq_LK_P4_k_dma64 1024 32 2w:0r 0w:4r
mriq_LK_P8_k_dma64 1024 32 2w:0r 0w:8r
mriq_LK_P16_k_dma64 1024 32 2w:0r 0w:16r

mriq_SK_P4_k_dma64 3072 32 2w:0r 0w:4r
mriq_SK_P8_k_dma64 3072 32 2w:0r 0w:8r
mriq_SK_P16_k_dma64 3072 32 2w:0r 0w:16r
```

- PLMs storing x-space variables
  - Different PLMs for different DMA widths
- PLMs storing k-space variables
  - Work with different DMA widths
    - dma32: 1 write port, 1 read port
    - dma64: 2 write ports, 2 read ports
  - Work with different architectures:
    - SK (Small-K) works for A0
    - LK (Large-K) works for A1
  - Work with different parallelism level:
    - P4, 4 read ports (enable parallel reading)
    - P8, 8 read ports
    - P16, 16 read ports

Reduce Area:

Optimization of fixed-point data types

- Checked 3 datasets, the values of all the variables are within a specific range
- Defined 2 types of fixed-point data representations: FPDATA_S(mall) and FPDATA_L(arge), WL = 24
- Both area and latency are improved with WL=24
- RTL simulation passed when WL is 24
- Baremetal test failed (not supported)

Two representations of fixed-point data types

	WL	IL	Fractional Bits
FPDATA_S	24	5	19
FPDATA_L	24	11	13

Performance Gain by reducing fixed-point precision

	$WL = 32$	$WL = 24$	Gain
area	0.1185	0.0812	-31.50%
latency (ns)	1740	1540	-11.49%

Reduce Latency:

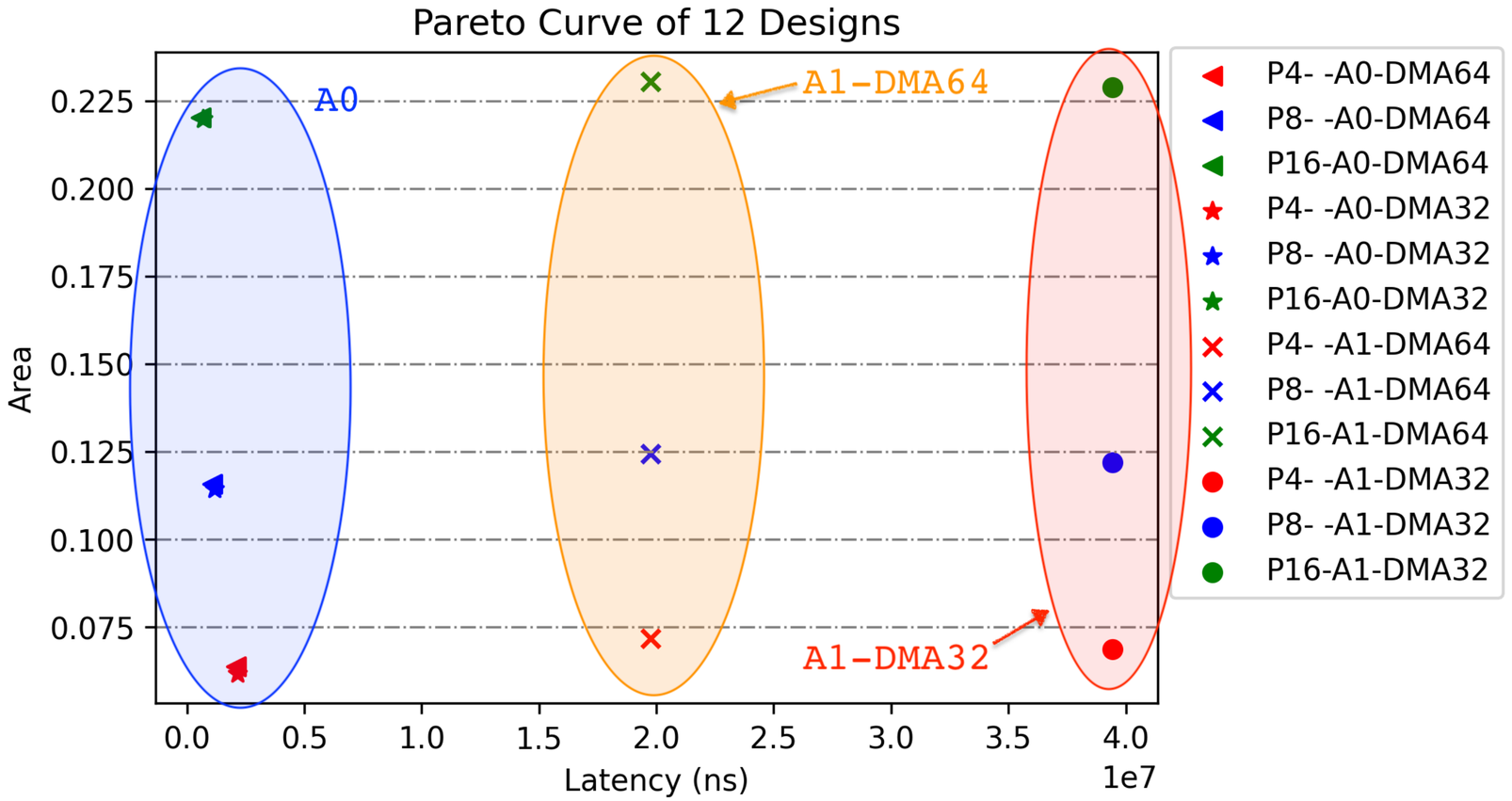
Parallelism Exploration:

- Loop unrolling and loop pipelining.
- 3 unroll_factors: [4, 8, 16]
- Each parallelism level has the corresponding set of PLMs

Pseudo Code where we applied loop unrolling and pipelining

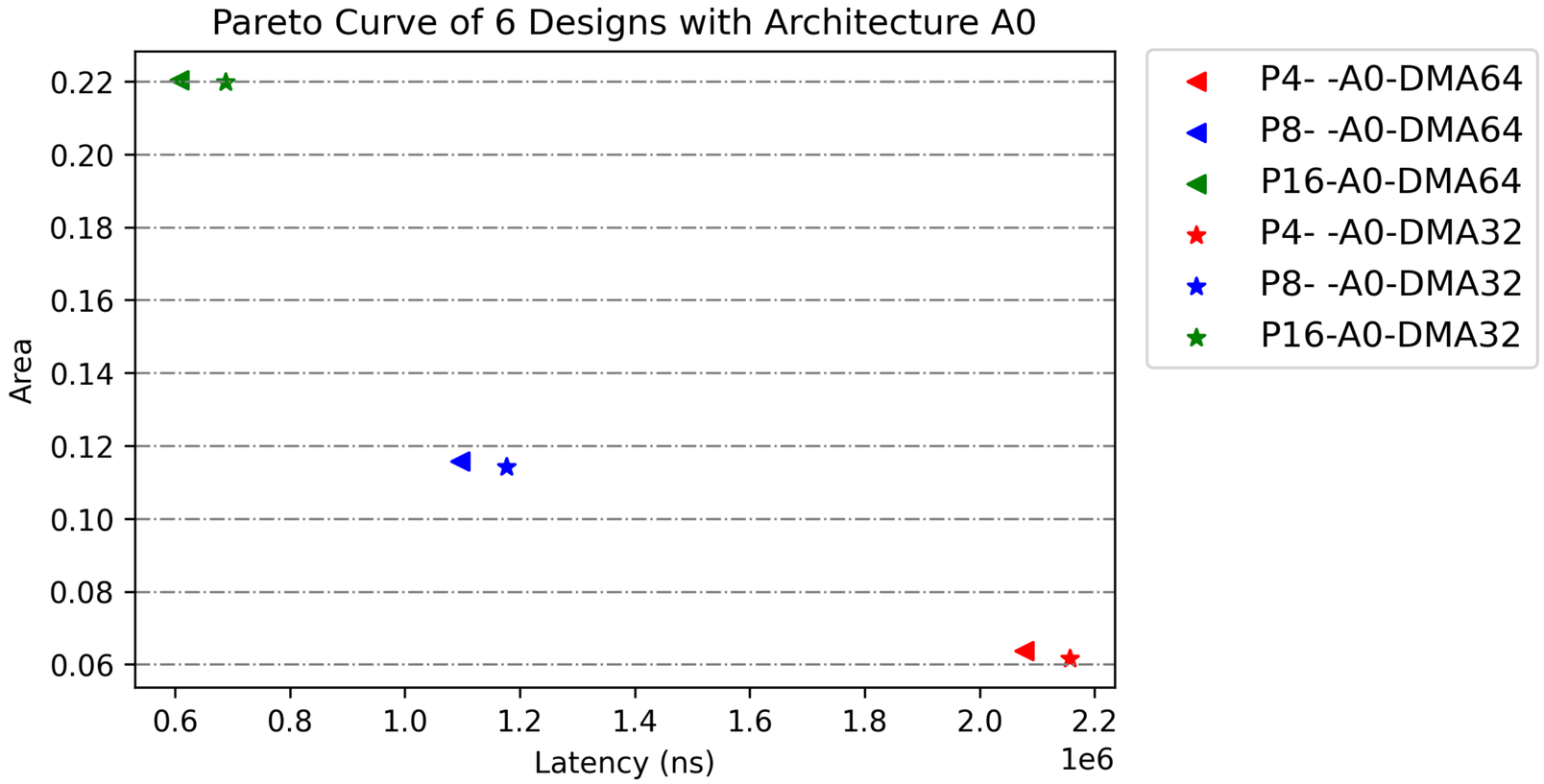
```
for (i=0; i < numK; i += unroll_factor) do
    HLS_LOOP_PIPELINE
    for (i = 0; i < unroll_factor; i++) do
        HLS_LOOP_UNROLL
        read from PLMs into registers
          kx[i],ky[i],kz[i],phiR[i],phiI[i]
    end
    for (i = 0; i < unroll_factor; i++) do
        HLS_LOOP_UNROLL
        compute Qracc_p[i], Qiacc_p[i]
    end
    for (i = 0; i < unroll_factor; i++) do
        HLS_LOOP_UNROLL
        Qracc += Qracc_p[i]
        Qiacc += Qiacc_p[i]
    end
end
```

Pareto Curve

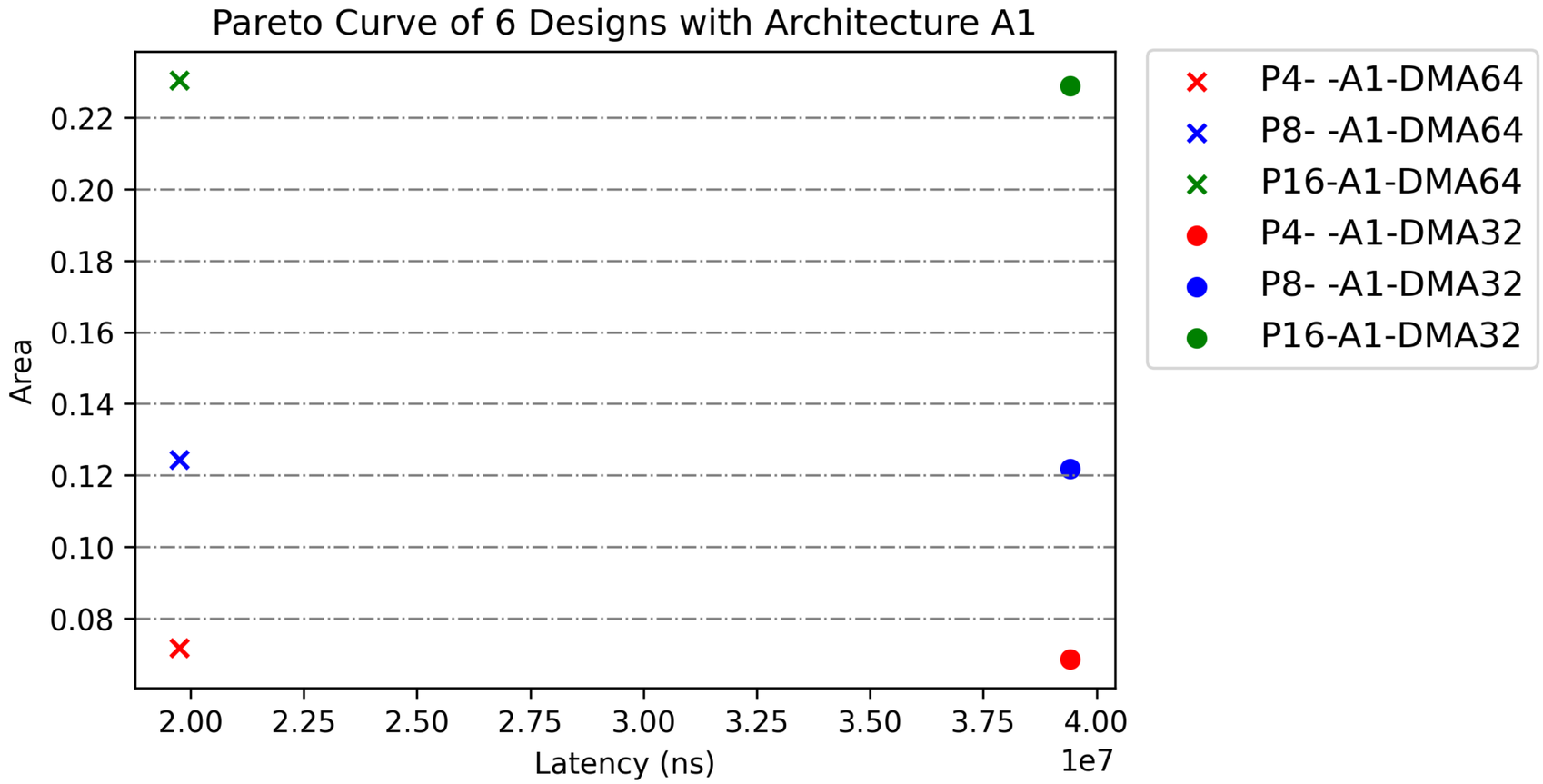


data collected through the RTL synthesis result and RTL simulation
data size: numX = 256, numK = 3072

- A0 has much smaller latency than A1
- A1 has slightly higher area, but much higher latency than A0



1. A0 can process “small” and “large” dataset (numK = 3 K, 2 K).
2. Changing the parallelism level impacts latency significantly.
3. Changing the DMA width does not affect much the latency.
4. A0 is computation-bounded.

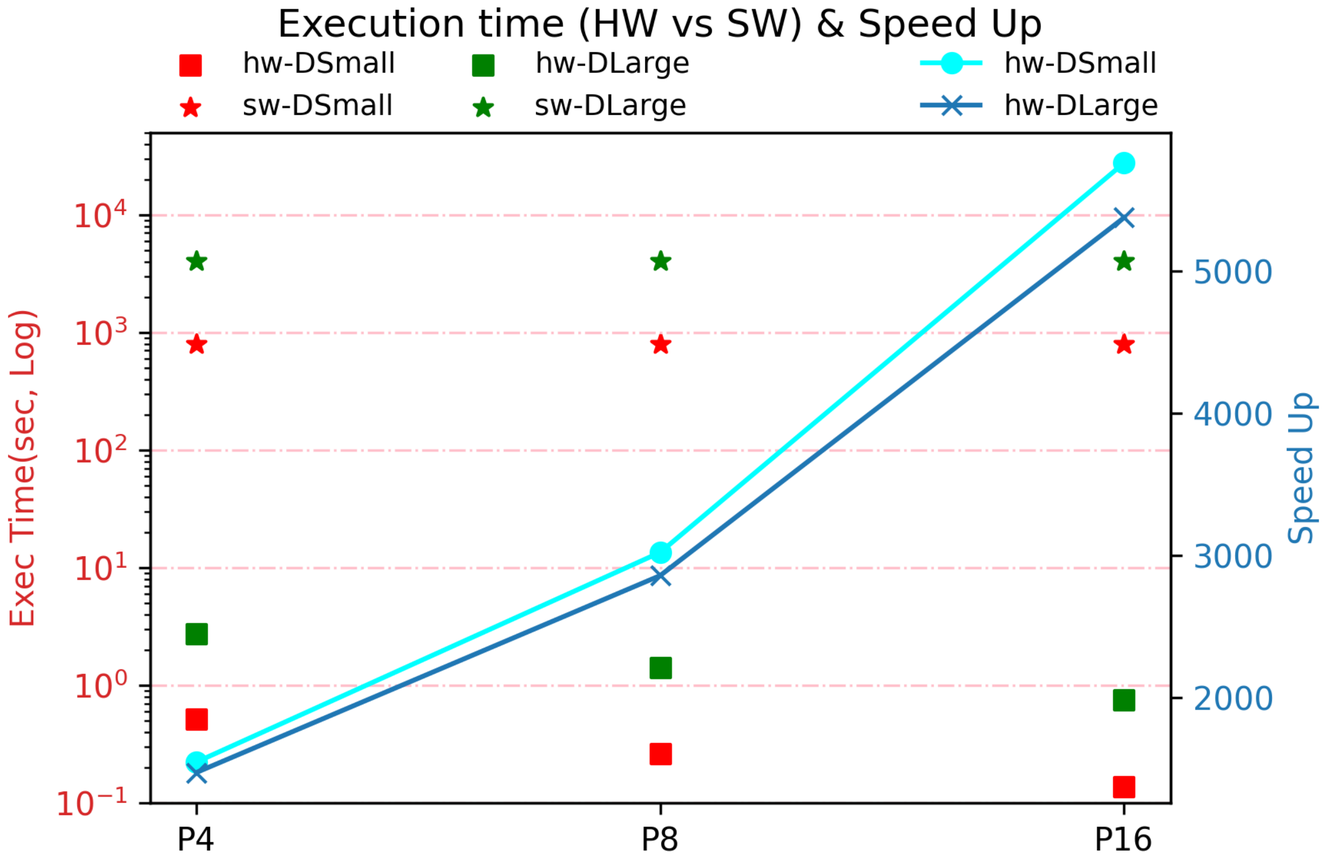


1. A1 can process “128x128x128” dataset. (numK = 2 M, storage requirement for k-space data is 40 MB)
2. Changing the DMA width significantly impacts the latency.
3. A1 is memory-bounded.

Testing Results collected on FPGA

- Invoke the MRI-Q accelerator in Linux on the FPGA
- Measure the speed-up against software running on Ariane

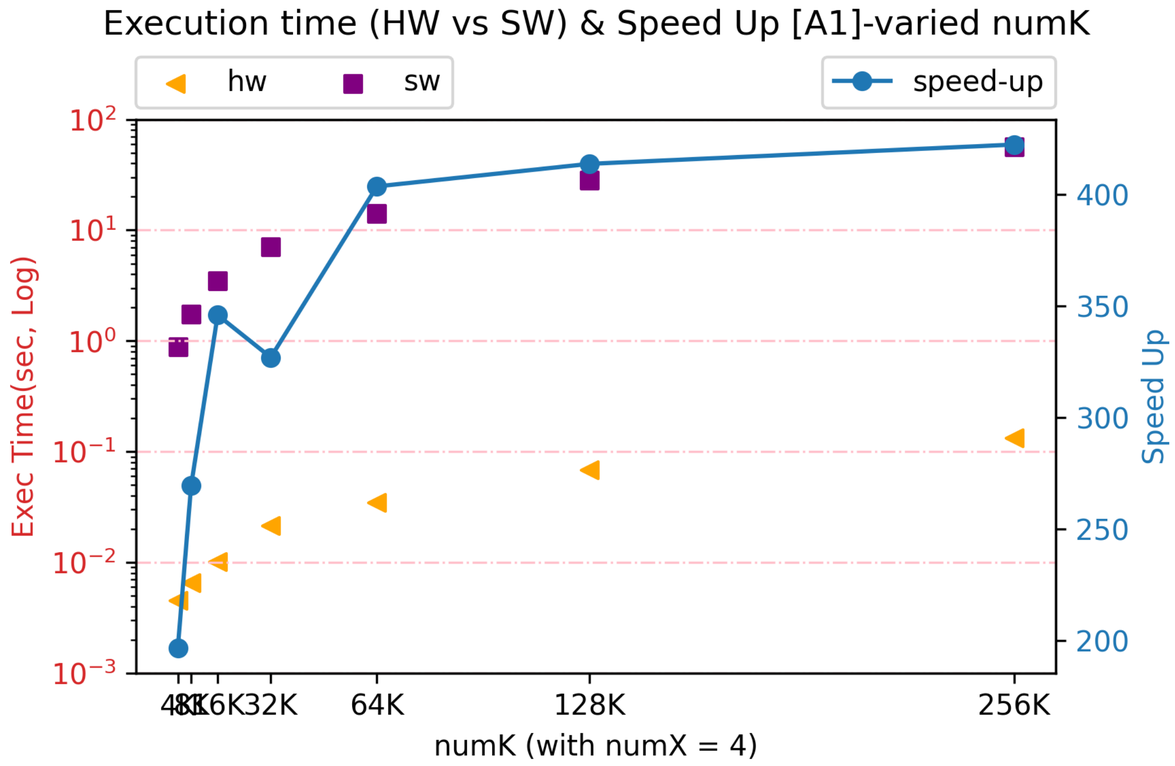
A0 architecture

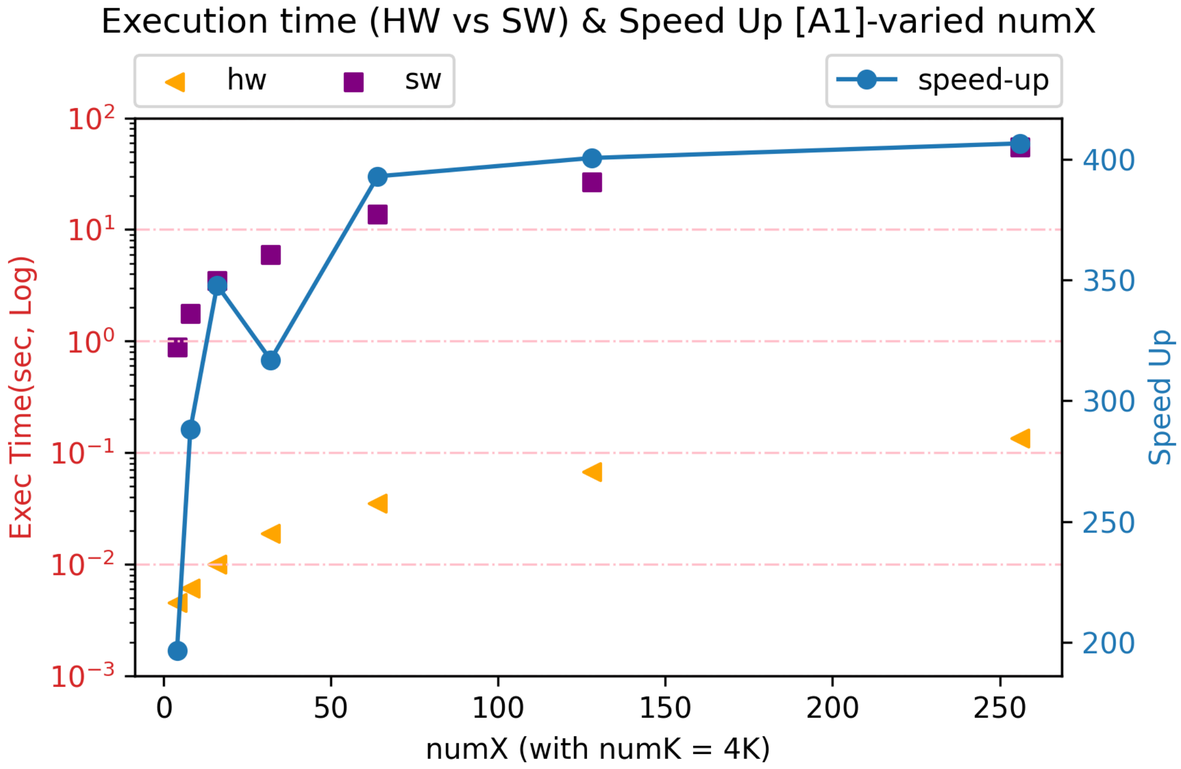


- The speedup increases over the parallelism level
- With A0_P16_DMA64 accelerator, the speed-up is > 5000X for both datasets

Testing Results collected on FPGA

A1 architecture





- A1 should work with very high resolution images
- The speed-up is > 400X compared to the software program

Summary

1. Designed MRI-Q accelerator with Stratus HLS Flow
2. Obtained multiple Pareto-optimal implementations
3. Integrated the accelerators in the ESP and tested them on one FPGA
4. One accelerator shows a speed-up of ~5000X