Fast and Scalable Architectures and Algorithms for the Computation of the Forward and Inverse Discrete Periodic Radon Transform with Applications to 2D Convolutions and Cross-Correlations.

Cesar Carranza

December 17th, 2015

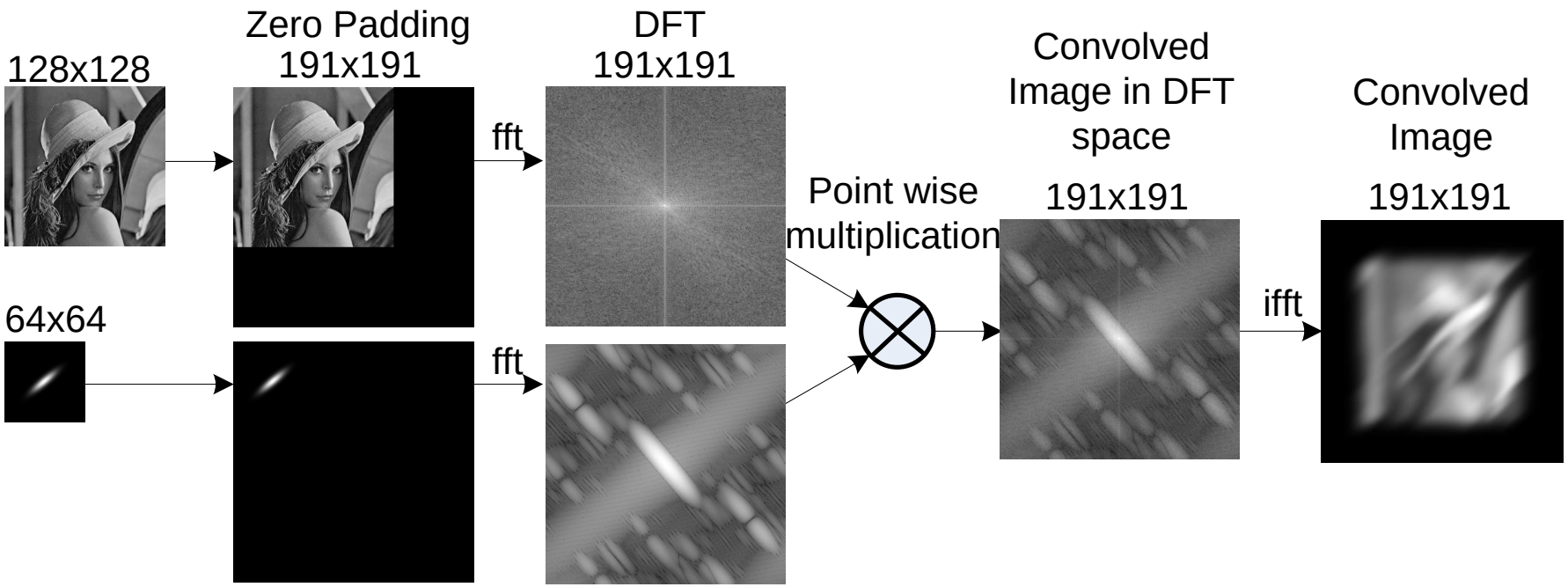
This work was supported by the National Science Foundation through the Division of Computer and Network Systems under Grant NSF AWD CNS -1422031.

Outline

- Motivation
- Thesis statement
- Prior work
- Contributions
- Methods
- Results
- Future work
- Concluding remarks
- Publications

Motivation

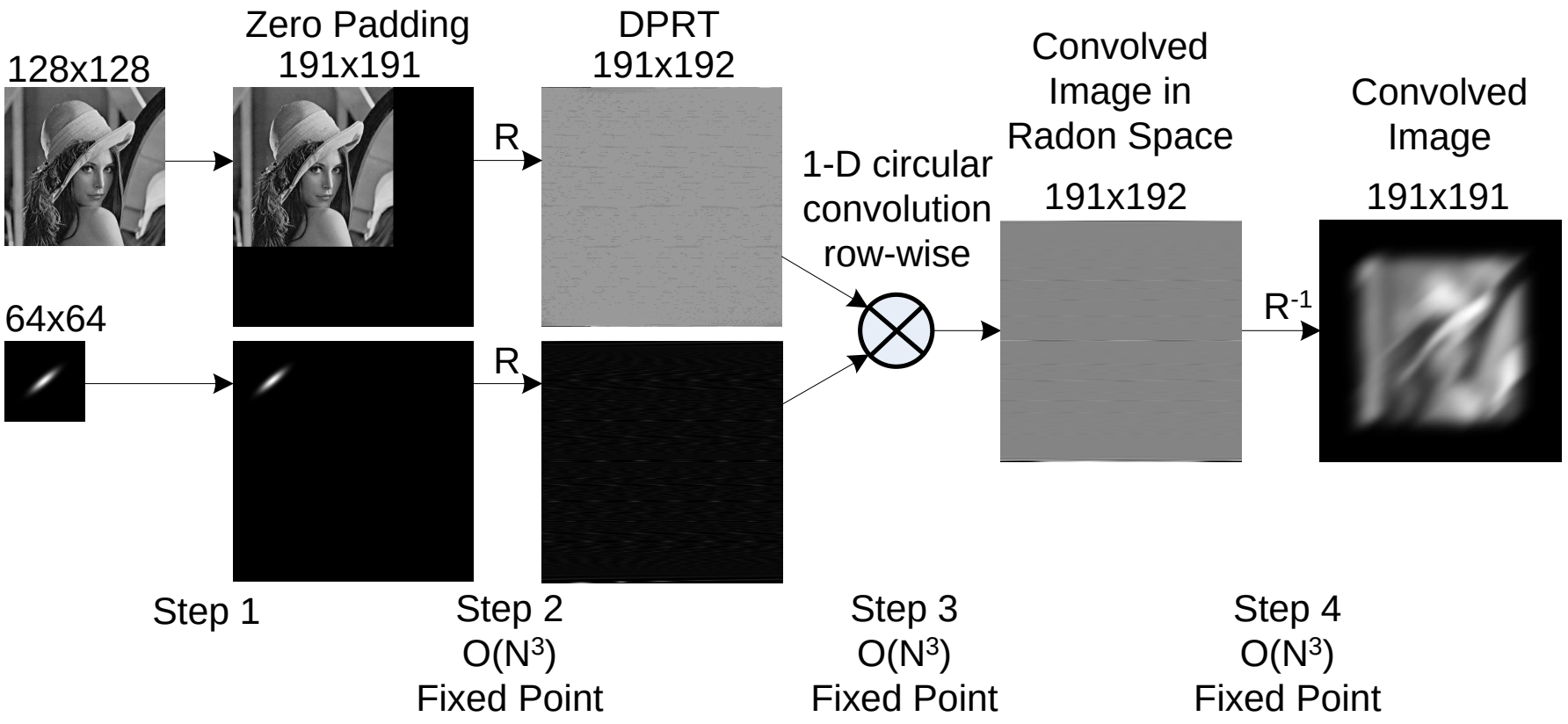
2-D Convolution using the FFT



Expensive butterfly implementation limit parallelism (1 to 8)
Floating point units cost 20x fixed-point units [Vera, 2011]

Motivation

2-D Convolution using the DPRT



Can we speedup this approach on modern devices so that it outperforms everything else?
Basic concept: Develop optimized I/O, parallel and pipelined fixed point architectures.

Thesis Statement

I believe that it is possible to develop fast and scalable architectures and algorithms for the computation of the Discrete Periodic Radon Transform (DPRT) and its inverse (iDPRT), that will enable the application of the DPRT in different areas where its use was limited due the lack of a fast implementation (e.g., in 2D convolutions and cross-correlations).

Furthermore, I believe that it is possible to develop highly efficient, parallel DPRT and iDPRT algorithms on existing GPUs and multi-core CPUs.

Prior Work

Mathematical background and algorithms:

- [Grigoryan, 1984] 2-D DFT using the DPRT for prime numbers and power of two sized images.
- [Matus, 1993] DPRT and its inverse for prime sized images. Sequential algorithm $O(N^3)$
- [Hsung, 1996] DPRT and its inverse for power of two sized images.
- [Pattichis, 2000] DPRT (power of two sizes) in $O(N^2 \log_2 N)$
- [Kingston, 2006] Generalized the DPRT to square arrays of arbitrary size.

DPRT for prime sizes: Minimum set of prime directions, no redundancy, closed form for the inverse.

Prior Work

Architecture implementations (Image size N x N)

- [Wisinger 2003]
  - Running time (RT): $O(N^4)$, Resource usage (RU): $O(N^2)$.
- [Rahman 2004]
  - RT: $O(N^3)$, RU: $O(N^2)$. Also, RT: $O(N^2)$, RU: $O(N^2)$. Fixed size 7x7.
- [Uzun 05]
  - RT: $O(N^3)$, RU: $O(N^2)$.
- [Chandrasekaran 05]
  - **Serial, parameterized and power efficient architecture, RT: $N(N^2 + 2N + 1)$, RU: $O(N)$.**
  - Parallel non parametrizable architecture, RT: $O(N^2)$, RU: $O(N^2)$.
- [Chandrasekaran 08]
  - **Systolic, parameterized architecture, RT: $(N^2 + N + 1)$, RU:$O(N^2)$.**

All of these methods are based on a non scalable, sequential algorithm that cannot be effectively parallelized.

Prior work

2-D convolutions and Cross-correlations (non-separable kernels):

- [Keung1990] Serial systolic array that removes the I/O issues.
  Running time: $O(N^2)$, resources $O(N^3)$
- [Mohanty1996] Parallel and Scalable systolic array.
  Running time $O(N) – O(N^2)$, resources $O(N^3) – O(N^2)$
- [Fowers2015] Sliding window in the spatial domain.
  Running time $O(N^2)$, resources $O(N^2)$.
- [Rao2010] Fast Fourier Transform (FFT). $O(N^2 \log_2 N)$
- [Uzun2005,Xilinx2012] FPGA implementations of FFT. Serial and Parallel $O((N^2 \log_2 N)/p)$. p is the number of parallel 1-D FFTs.
  Resource usage $O(N) – O(N^2)$.

I propose new frameworks that can provide faster running time with lower resource usage

Contributions

Scalable frameworks for computing:

- Forward DPRT
- Inverse DPRT
- 2-D Convolutions and Cross-correlations.

Scalability based on:

- Pareto optimality that provides the fastest possible implementation based on the available HW.
- Vastly reduction on resources that can fit now in devices.

Contributions

Fast frameworks:

- An array of shift registers and adder trees to compute up to N^2 additions per clock cycle.
  I. No address calculations
  II. No I/O delays
- A custom SRAM able to provide up to N values per clock cycle.
  I. Full row of an image.
  II. Full column of an image (fast transposition).

Contributions

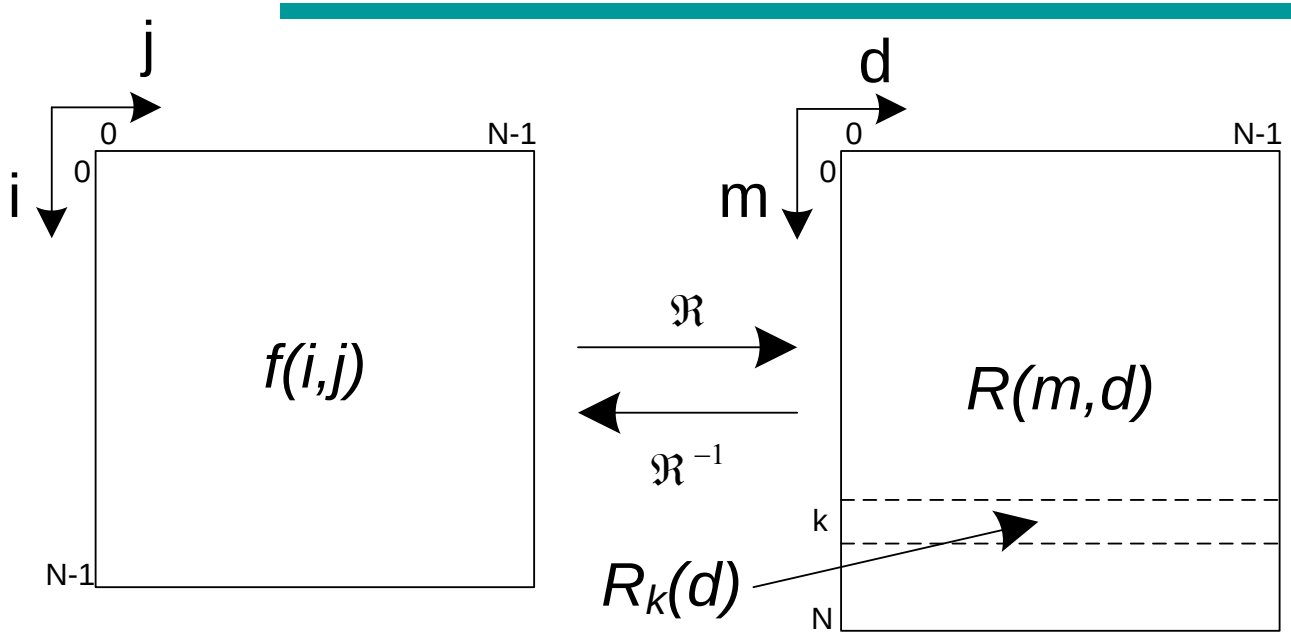
- Fast 1-D convolvers
  - Parallel and pipelined convolvers that compute one output pixel per clock cycle.
- Custom SRAMs to perform the fast transposition needed in the LU framework.

Contributions

For the computation of the DPRT and its inverse on CPUs/GPUs:

- Distributed processing of prime directions to CPU-cores and GPU-Multiprocessors.
- **Parallel** and **syncronous computation** of rays of the prime directions by cores inside the GPU-Multiprocessors

Methods: Background - DPRT



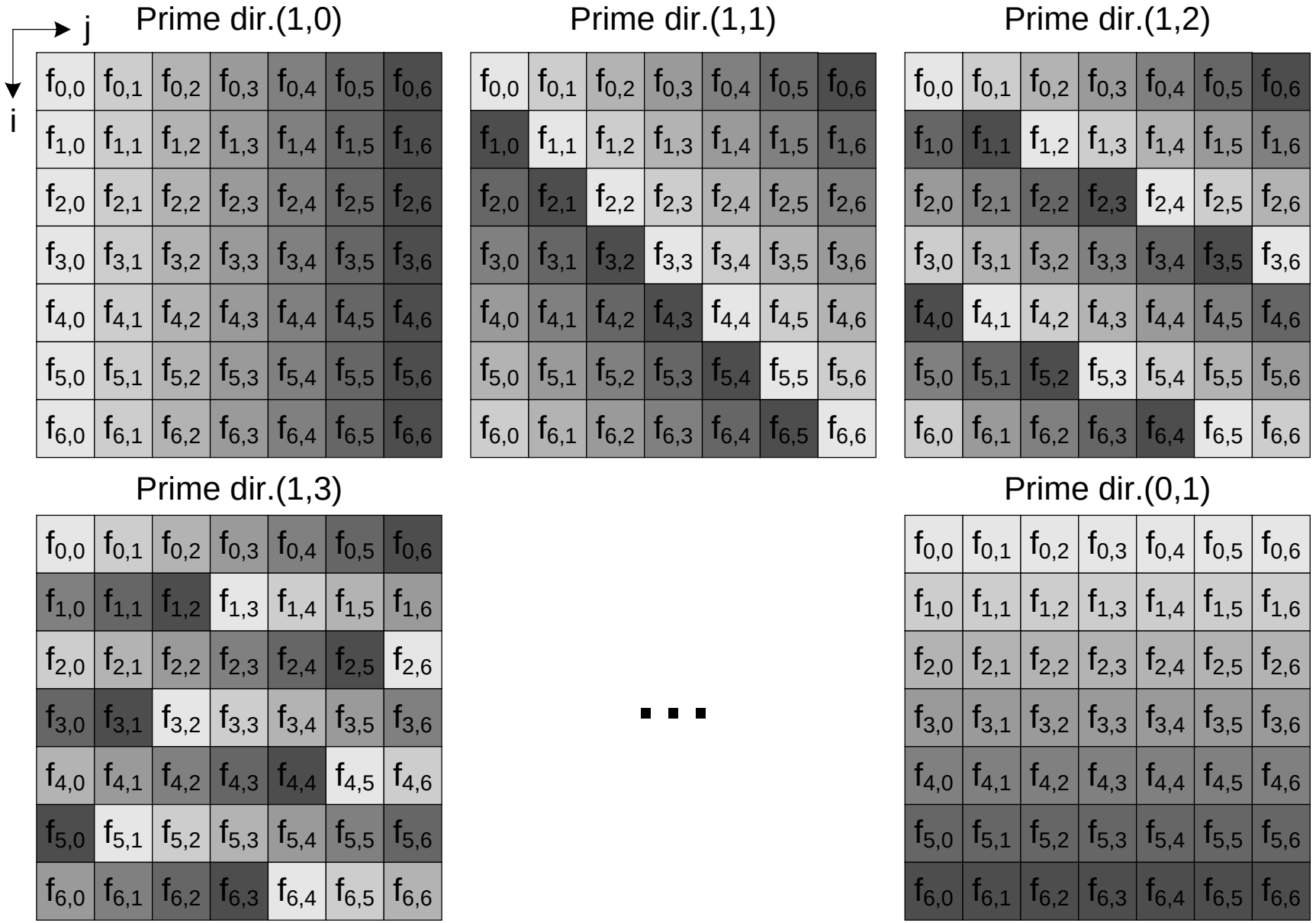
$$R(m,d) = \begin{cases} \sum_{i=0}^{N-1} f(i, \langle d + mi \rangle_N), & 0 \le m < N \\ \sum_{j=0}^{N-1} f(d,j), & m = N \end{cases}$$

$$f(i,j) = \frac{1}{N}\left[\sum_{m=0}^{N-1} R(m, \langle j - mi \rangle_N) - S + R(N,i)\right]$$

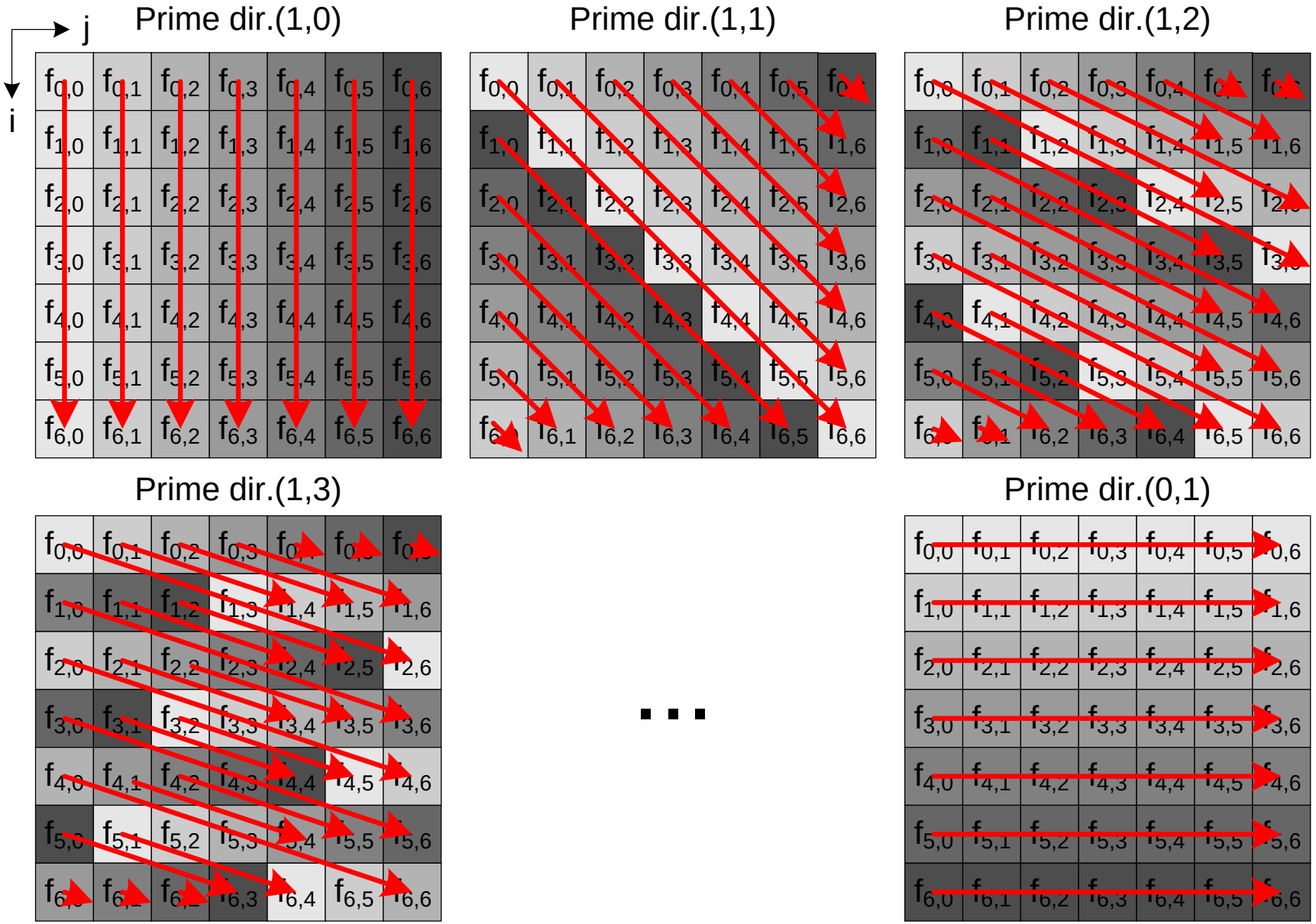
$$S = \sum_{j=0}^{N-1}\sum_{i=0}^{N-1} f(i,j) = \sum_{d=0}^{N-1} R(m,d)$$

Illustration of the DPRT and its iDPRT for an image f of size $N \times N$. The row vector k of $R(m,d)$ is denoted as $R_k(d)$ which represents the k projection of $f(i,j)$.

Background: Prime directions



Background: Prime directions



Background: DPRT

Forward DPRT (Input: NxN):

- N+1 Prime directions
- N rays per prime direction
- N pixels to be added per ray

Number of additions: $(N+1)N(N-1)$

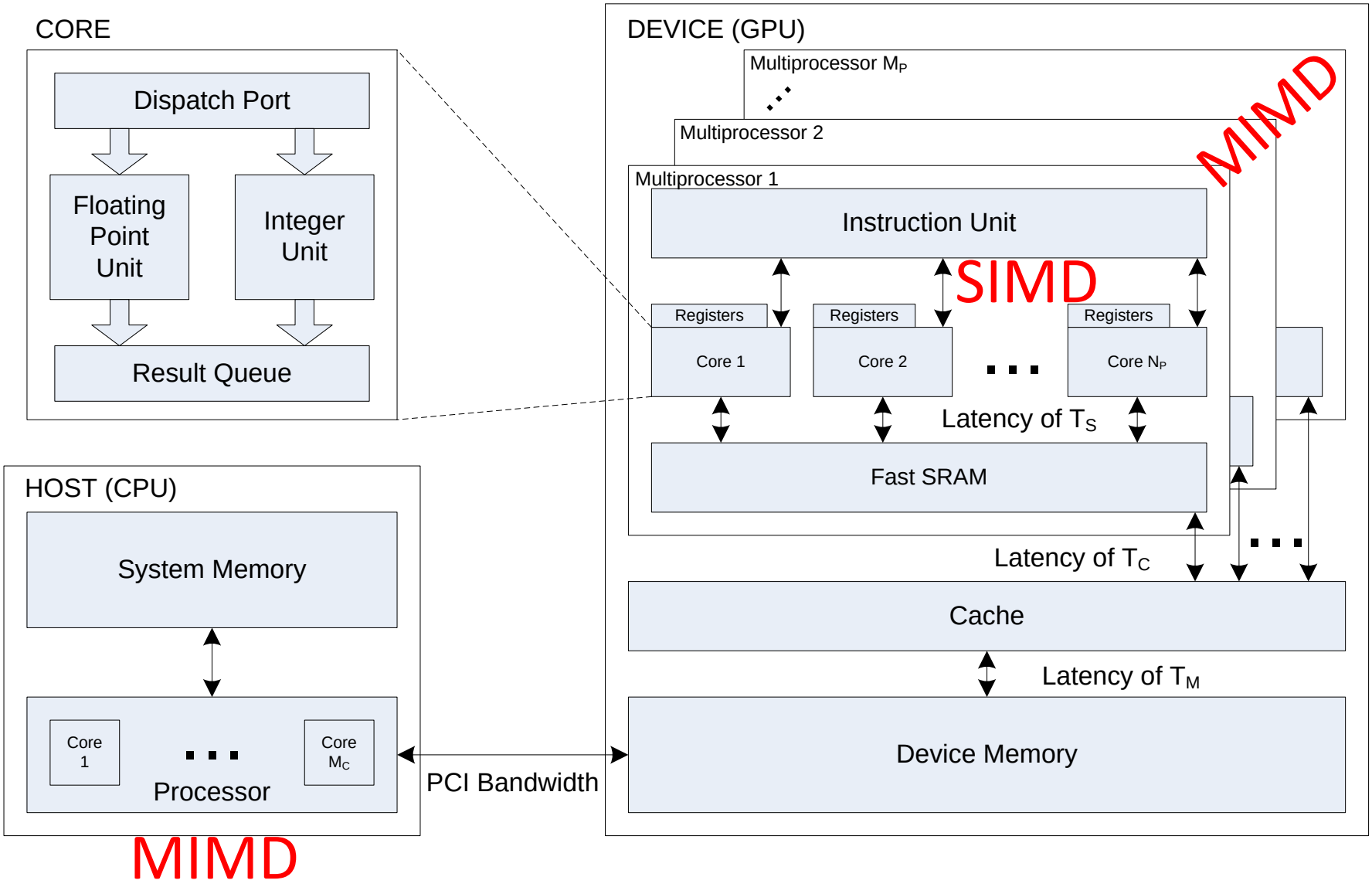
Inverse DPRT (Input (N+1)xN):

- N Prime directions
- N rays per prime direction
- N pixels to be added per ray + 2 extra additions and one division

Number of additions $N^2(N+1)$

Number of divisions: N^2

Background: CPUs/GPUs



Methods: Parallel DPRT on CPUs

Parallel algorithms for computing the forward DPRT of an NxN image

1: Partition the set of $N+1$ prime directions into M_C sets of consecutive prime directions
2: Launch M_C threads, assing each partitioned set to each thread
3: Wait for threads to finish

Main parallel algorithm for computing the forward Discrete Periodic Radon Transform $R(m,d)$ of the image $f(i,j)$ of size $N \times N$ on a CPU with M_C cores.

$O(N^3)$ additions
$O(N^3/M_C)$ additions per core
Theoretical speedup: M_C

Methods: Parallel DPRT on GPUs

Parallel algorithms for computing the forward DPRT of an NxN image

STEP1: Partition prime directions into M_P sets
STEP2: Launch the sets on the Multiprocessors.
Each prime direction requires the computation of N rays.
STEP3: Wait for threads to finish

```
1:  procedure fDPRT_DEVICE_Kernel(m, d)
2:      sum = 0
3:      if m = N then
4:          for j = 0 to N − 1 do
5:              sum = sum + f(d, j)
6:          end for
7:      else
8:          for i = 0 to N − 1 do
9:              sum = sum + f(i, ⟨d + m × i⟩_N)
10:         end for
11:     end if
12:     R(m, d) = sum
13: end procedure
```

Single core computes one ray ***d*** of prime direction ***m***

M_P = Number of multiprocessors
N_P = Number of Cores inside an MP

$O(N^3)$ additions
$O(N^3/M_P)$ additions per MP
$O(N^3/(M_P x N_P))$ per core

Ideal speedup = $M_P x N_P x (f_{GPU}/f_{CPU})$
Assumes no I/O issues, latencies.

Methods: Parallel DPRT on GPUs

Issues:

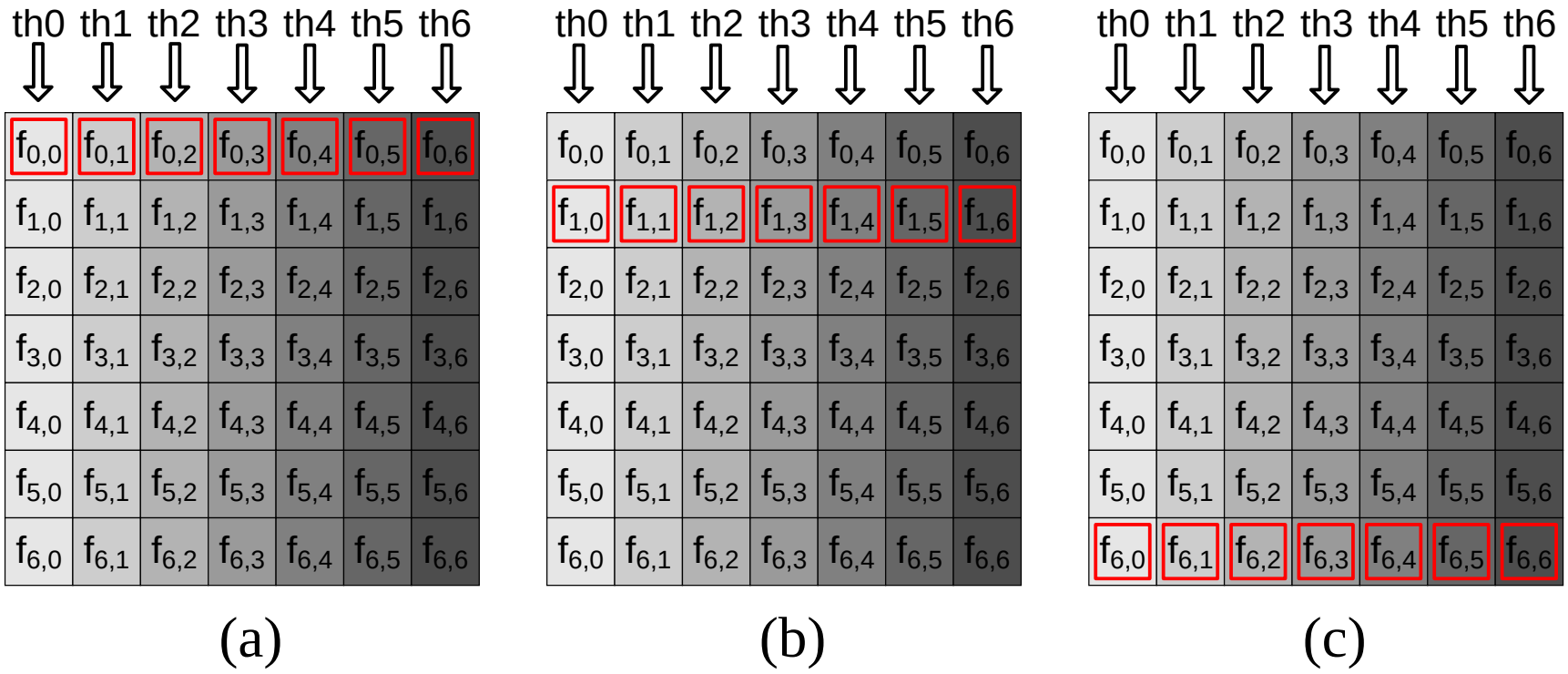
- Current GPUs have a relatively small Fast SRAM to load the whole image.
- Accesses to Cache or Global memory are costly.

Solution:

- Synchronize threads at the ray computation level exploiting locality of the data per row of the image.

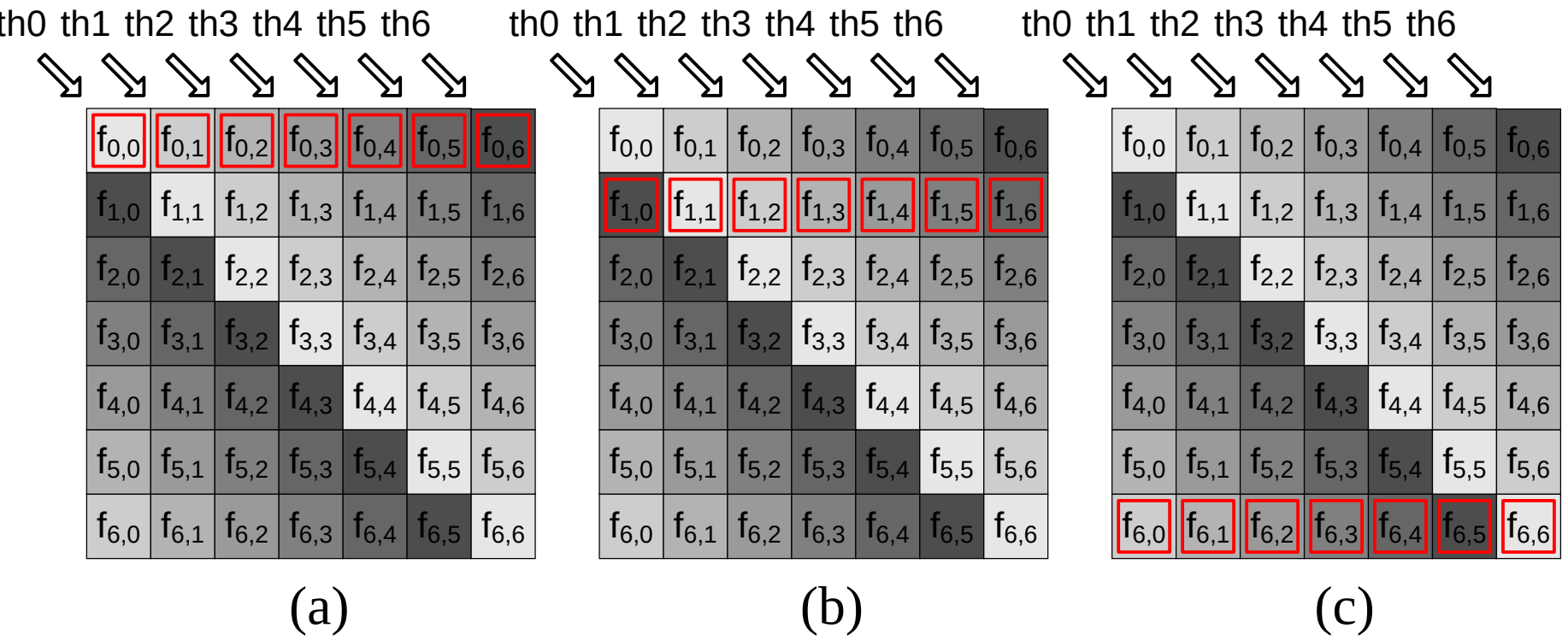
Methods: Parallel DPRT on GPUs

Memory pattern access by a set of threads computing the same prime direction but different rays



Methods: Parallel DPRT on GPUs

Memory pattern access by a set of threads computing the same prime direction but different rays



Methods: Parallel DPRT on GPUs

Kernel algorithm for computing the forward DPRT of an NxN image

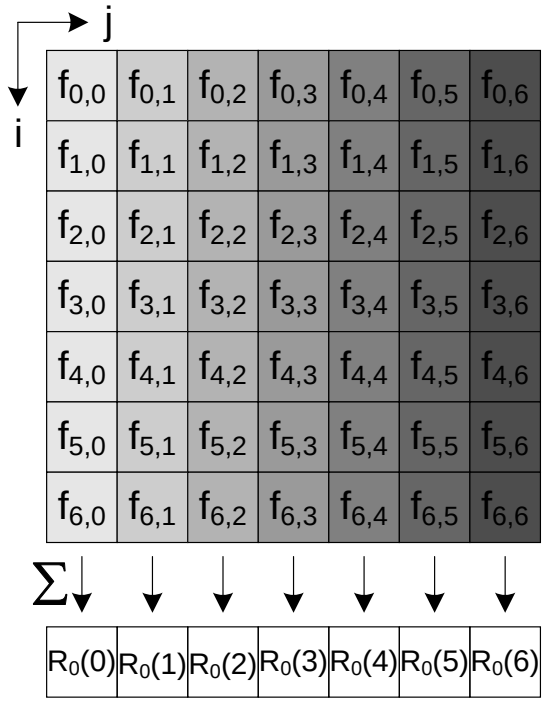
1: **procedure** $fDPRT\_GPU\_Kernel(radon, img, N, m, d)$
2: **if** $m = N$ **then**
3: $offs = 0$
4: $incr = 1$
5: $init = d \times N$;
6: **else**
7: $offs = d$
8: $incr = N$
9: $init = 0$;
10: **end if**
11: **Synchronize threads**
12: $k = d + m \times N$
13: $sum = 0$
14: **for** $i = 0$ to $N - 1$ **do**
15: $sum = sum + img[init + offs]$
16: $offs = \langle offs + m \rangle_N$
17: $init = init + incr$
18: **end for**
19: $radon[k] = sum$
20: **end procedure**

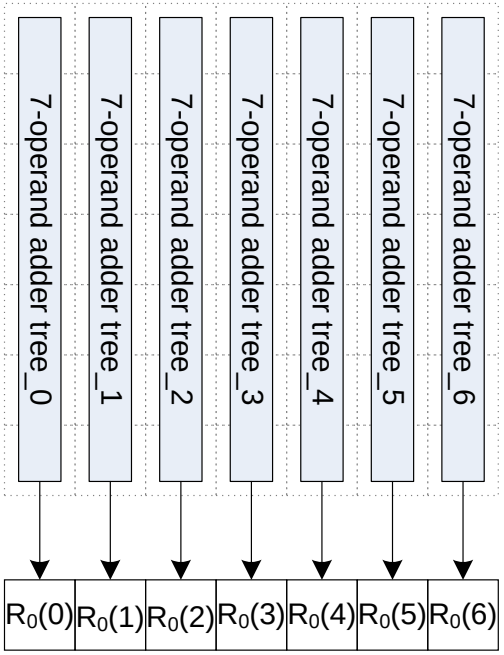
Kernel algorithm for each core on the GPU to compute one ray of the forward Discrete Periodic Radon Transform $R(m, d)$ of the image $f(i, j)$ of size $N \times N$. $R(m, d)$ is mapped to a vector $radon[k]$ and $f(i, j)$ is mapped to a vector $img[k]$, both using row-major order.

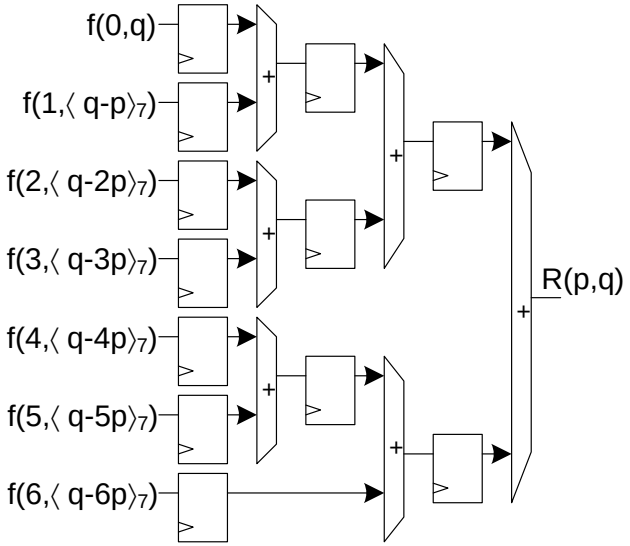
- Synchronous computation
- Simplified address calculation
- Row-major memory access
- Concurrent reads and writes
- No writes conflicts

Note: Partial DPRTs are not possible.

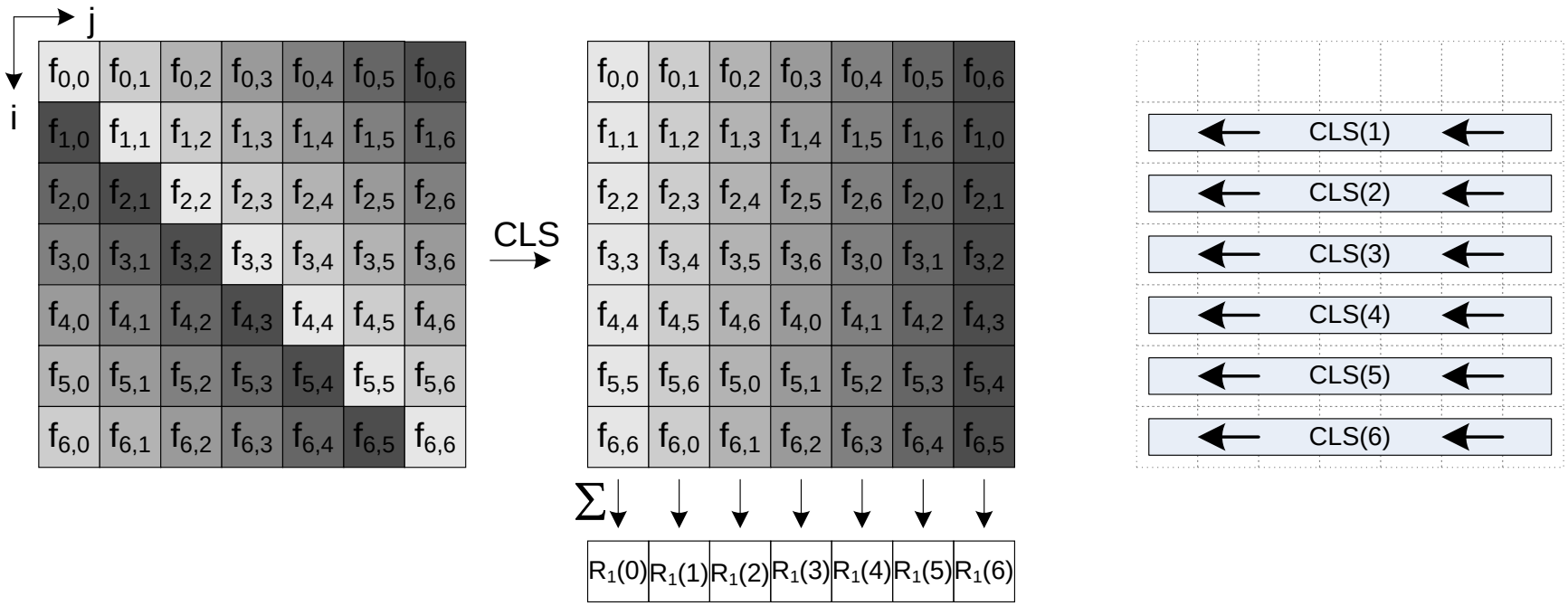
Methods: Fast DPRT



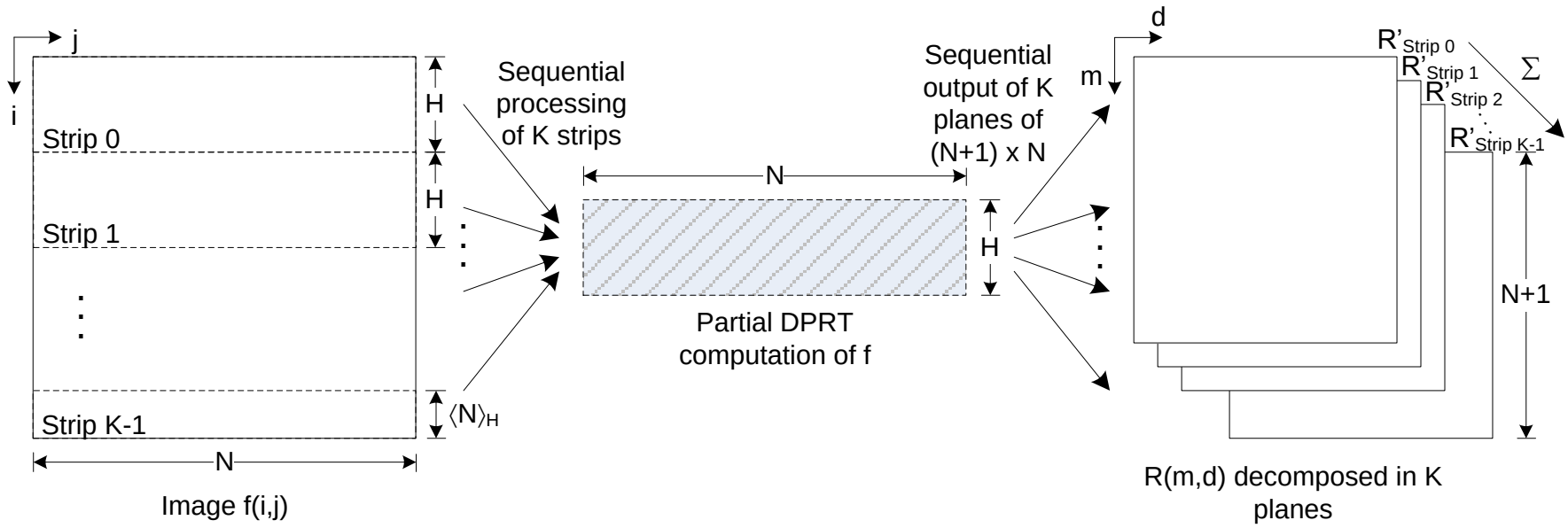




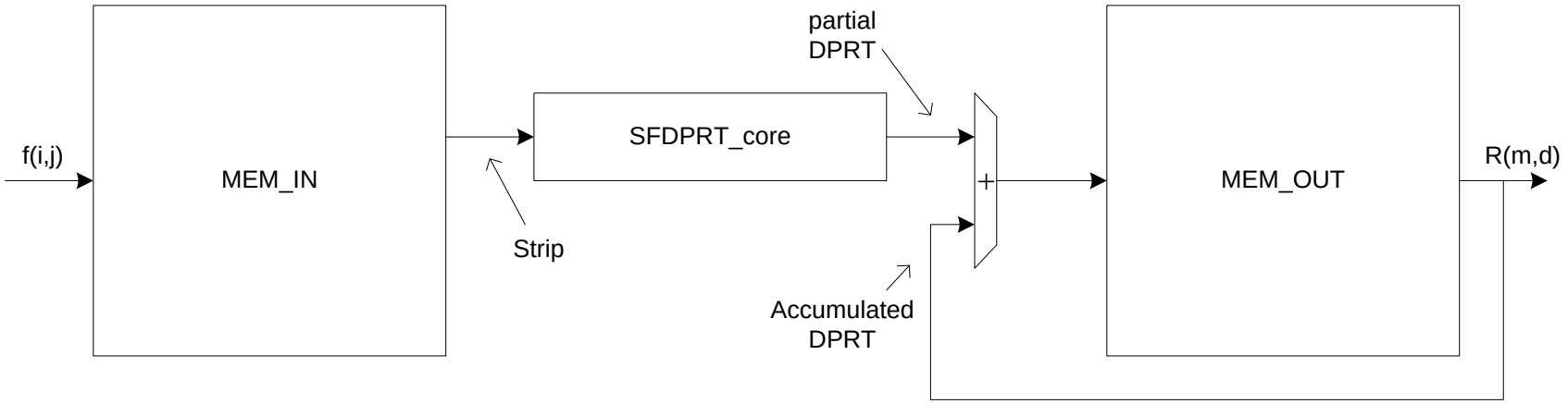
Fast DPRT


j
i
$f_{0,0}$ $f_{0,1}$ $f_{0,2}$ $f_{0,3}$ $f_{0,4}$ $f_{0,5}$ $f_{0,6}$
$f_{1,0}$ $f_{1,1}$ $f_{1,2}$ $f_{1,3}$ $f_{1,4}$ $f_{1,5}$ $f_{1,6}$
$f_{2,0}$ $f_{2,1}$ $f_{2,2}$ $f_{2,3}$ $f_{2,4}$ $f_{2,5}$ $f_{2,6}$
$f_{3,0}$ $f_{3,1}$ $f_{3,2}$ $f_{3,3}$ $f_{3,4}$ $f_{3,5}$ $f_{3,6}$
$f_{4,0}$ $f_{4,1}$ $f_{4,2}$ $f_{4,3}$ $f_{4,4}$ $f_{4,5}$ $f_{4,6}$
$f_{5,0}$ $f_{5,1}$ $f_{5,2}$ $f_{5,3}$ $f_{5,4}$ $f_{5,5}$ $f_{5,6}$
$f_{6,0}$ $f_{6,1}$ $f_{6,2}$ $f_{6,3}$ $f_{6,4}$ $f_{6,5}$ $f_{6,6}$
CLS
$f_{0,0}$ $f_{0,1}$ $f_{0,2}$ $f_{0,3}$ $f_{0,4}$ $f_{0,5}$ $f_{0,6}$
$f_{1,1}$ $f_{1,2}$ $f_{1,3}$ $f_{1,4}$ $f_{1,5}$ $f_{1,6}$ $f_{1,0}$
$f_{2,2}$ $f_{2,3}$ $f_{2,4}$ $f_{2,5}$ $f_{2,6}$ $f_{2,0}$ $f_{2,1}$
$f_{3,3}$ $f_{3,4}$ $f_{3,5}$ $f_{3,6}$ $f_{3,0}$ $f_{3,1}$ $f_{3,2}$
$f_{4,4}$ $f_{4,5}$ $f_{4,6}$ $f_{4,0}$ $f_{4,1}$ $f_{4,2}$ $f_{4,3}$
$f_{5,5}$ $f_{5,6}$ $f_{5,0}$ $f_{5,1}$ $f_{5,2}$ $f_{5,3}$ $f_{5,4}$
$f_{6,6}$ $f_{6,0}$ $f_{6,1}$ $f_{6,2}$ $f_{6,3}$ $f_{6,4}$ $f_{6,5}$
Σ
$R_1(0)$ $R_1(1)$ $R_1(2)$ $R_1(3)$ $R_1(4)$ $R_1(5)$ $R_1(6)$
CLS(1)
CLS(2)
CLS(3)
CLS(4)
CLS(5)
CLS(6)

Background: Scalable DPRT

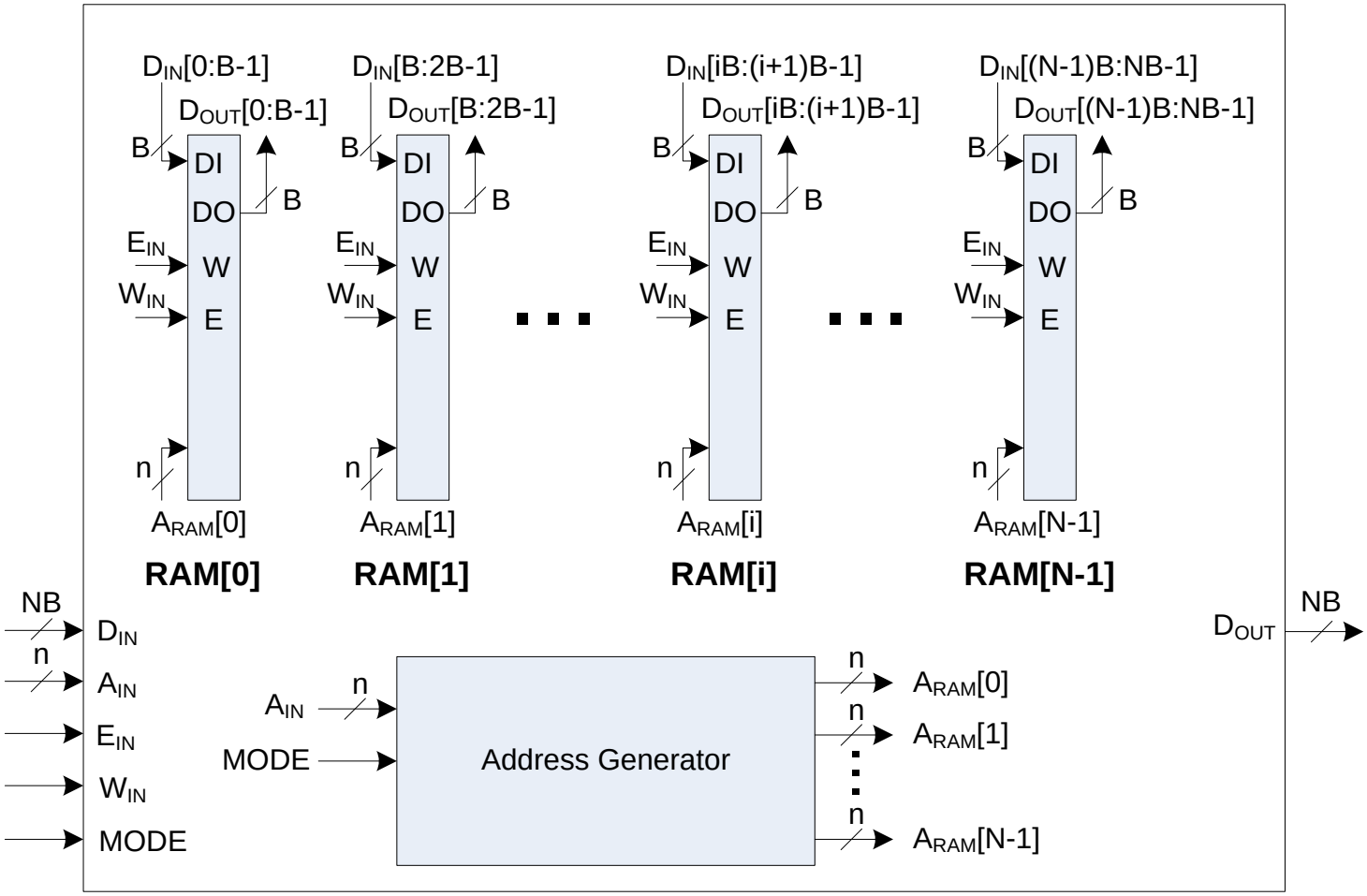


Scalable FDPRT



MEM_IN design

Custom SRAM for single cycle access to rows and columns



Row/Column access example

7x7 Image shifting patterns for fast access of rows and columns (transpose of the image)

$f_{0,0}$	$f_{0,1}$	$f_{0,2}$	$f_{0,3}$	$f_{0,4}$	$f_{0,5}$	$f_{0,6}$
$f_{1,0}$	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$	$f_{1,6}$
$f_{2,0}$	$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$	$f_{2,5}$	$f_{2,6}$
$f_{3,0}$	$f_{3,1}$	$f_{3,2}$	$f_{3,3}$	$f_{3,4}$	$f_{3,5}$	$f_{3,6}$
$f_{4,0}$	$f_{4,1}$	$f_{4,2}$	$f_{4,3}$	$f_{4,4}$	$f_{4,5}$	$f_{4,6}$
$f_{5,0}$	$f_{5,1}$	$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$	$f_{5,6}$
$f_{6,0}$	$f_{6,1}$	$f_{6,2}$	$f_{6,3}$	$f_{6,4}$	$f_{6,5}$	$f_{6,6}$

f(i,j)

$f_{0,0}$	$f_{0,1}$	$f_{0,2}$	$f_{0,3}$	$f_{0,4}$	$f_{0,5}$	$f_{0,6}$
$f_{1,6}$	$f_{1,0}$	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
$f_{2,5}$	$f_{2,6}$	$f_{2,0}$	$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$
$f_{3,4}$	$f_{3,5}$	$f_{3,6}$	$f_{3,0}$	$f_{3,1}$	$f_{3,2}$	$f_{3,3}$
$f_{4,3}$	$f_{4,4}$	$f_{4,5}$	$f_{4,6}$	$f_{4,0}$	$f_{4,1}$	$f_{4,2}$
$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$	$f_{5,6}$	$f_{5,0}$	$f_{5,1}$
$f_{6,1}$	$f_{6,2}$	$f_{6,3}$	$f_{6,4}$	$f_{6,5}$	$f_{6,6}$	$f_{6,0}$

Shifted *f*

$f_{0,0}$	$f_{0,1}$	$f_{0,2}$	$f_{0,3}$	$f_{0,4}$	$f_{0,5}$	$f_{0,6}$
$f_{1,6}$	$f_{1,0}$	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
$f_{2,5}$	$f_{2,6}$	$f_{2,0}$	$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$
$f_{3,4}$	$f_{3,5}$	$f_{3,6}$	$f_{3,0}$	$f_{3,1}$	$f_{3,2}$	$f_{3,3}$
$f_{4,3}$	$f_{4,4}$	$f_{4,5}$	$f_{4,6}$	$f_{4,0}$	$f_{4,1}$	$f_{4,2}$
$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$	$f_{5,6}$	$f_{5,0}$	$f_{5,1}$
$f_{6,1}$	$f_{6,2}$	$f_{6,3}$	$f_{6,4}$	$f_{6,5}$	$f_{6,6}$	$f_{6,0}$

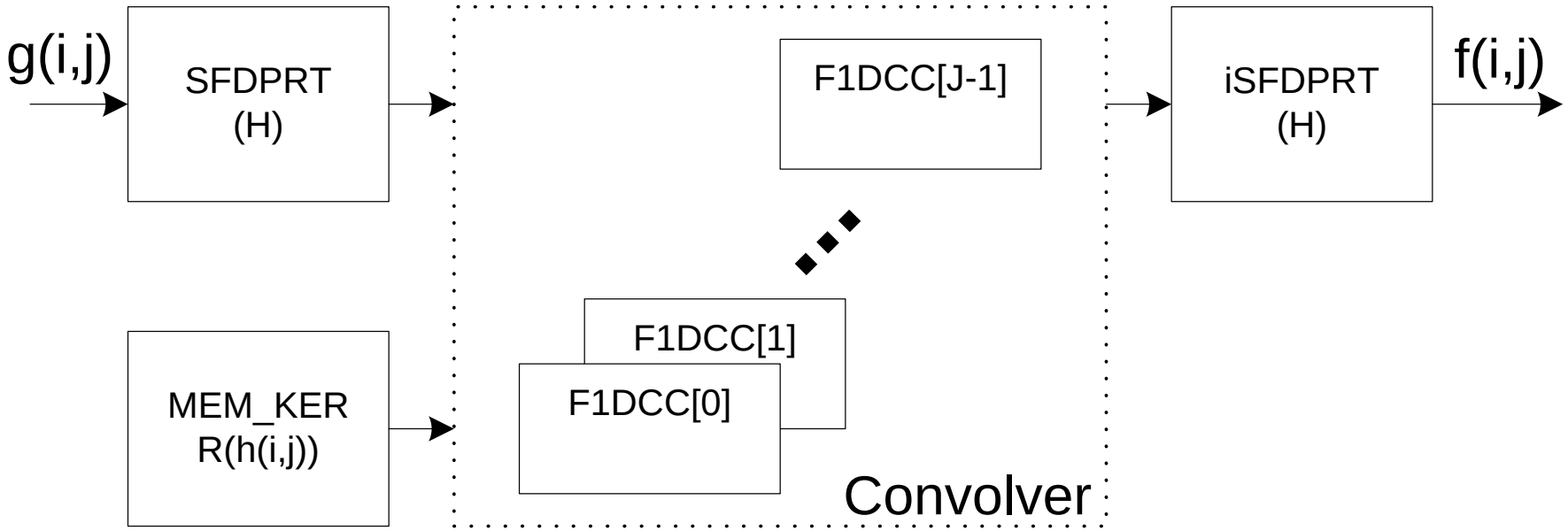
$A_{RAM}[0] = 4$, row_mode

$f_{0,0}$	$f_{0,1}$	$f_{0,2}$	$f_{0,3}$	$f_{0,4}$	$f_{0,5}$	$f_{0,6}$
$f_{1,6}$	$f_{1,0}$	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$
$f_{2,5}$	$f_{2,6}$	$f_{2,0}$	$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$
$f_{3,4}$	$f_{3,5}$	$f_{3,6}$	$f_{3,0}$	$f_{3,1}$	$f_{3,2}$	$f_{3,3}$
$f_{4,3}$	$f_{4,4}$	$f_{4,5}$	$f_{4,6}$	$f_{4,0}$	$f_{4,1}$	$f_{4,2}$
$f_{5,2}$	$f_{5,3}$	$f_{5,4}$	$f_{5,5}$	$f_{5,6}$	$f_{5,0}$	$f_{5,1}$
$f_{6,1}$	$f_{6,2}$	$f_{6,3}$	$f_{6,4}$	$f_{6,5}$	$f_{6,6}$	$f_{6,0}$

$A_{RAM}[0] = 4$, column_mode

Methods: 2-D Convolution/X-corr DPRT

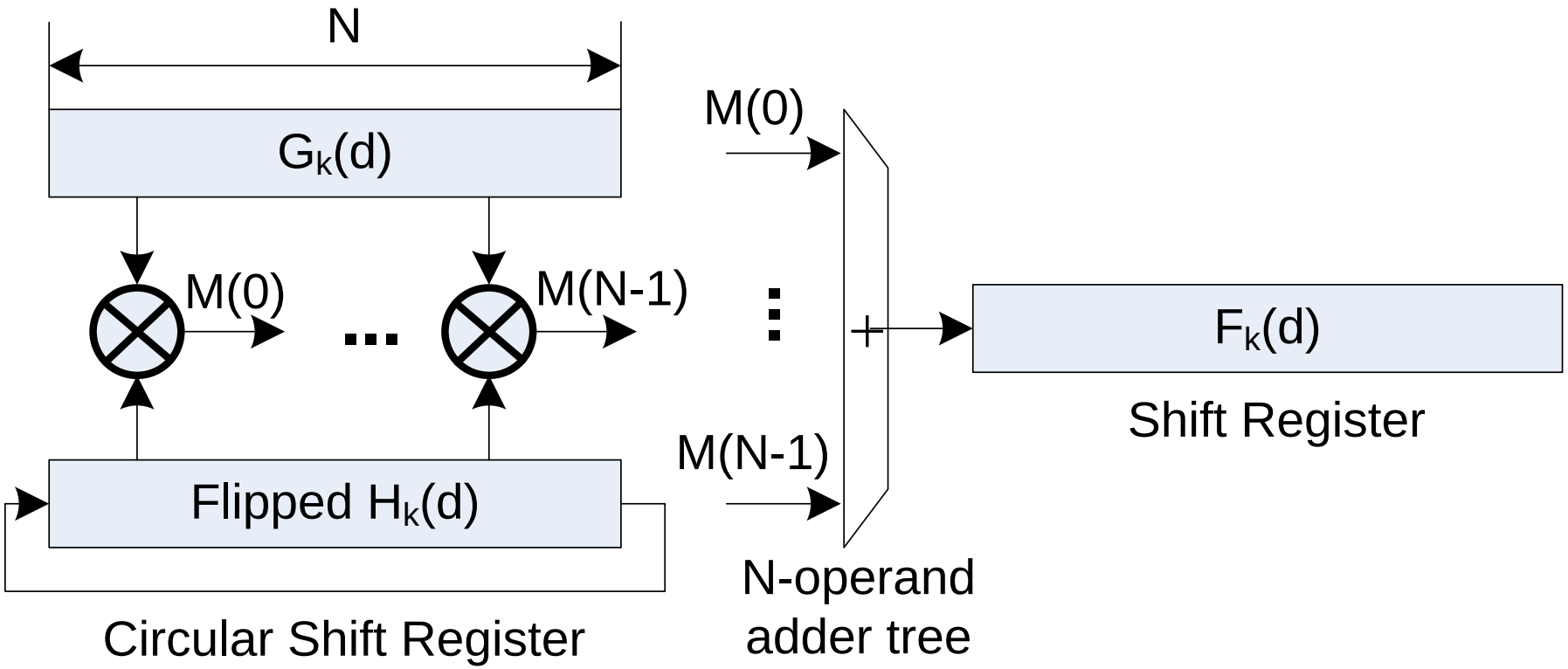
Fast and scalable architecture system for computing 2D convolutions/Cross-correlations **using the DPRT**.



SFDPRT: Scalable Fast DPRT, i for inverse

F1DCC: Fast 1-D Circular convolver

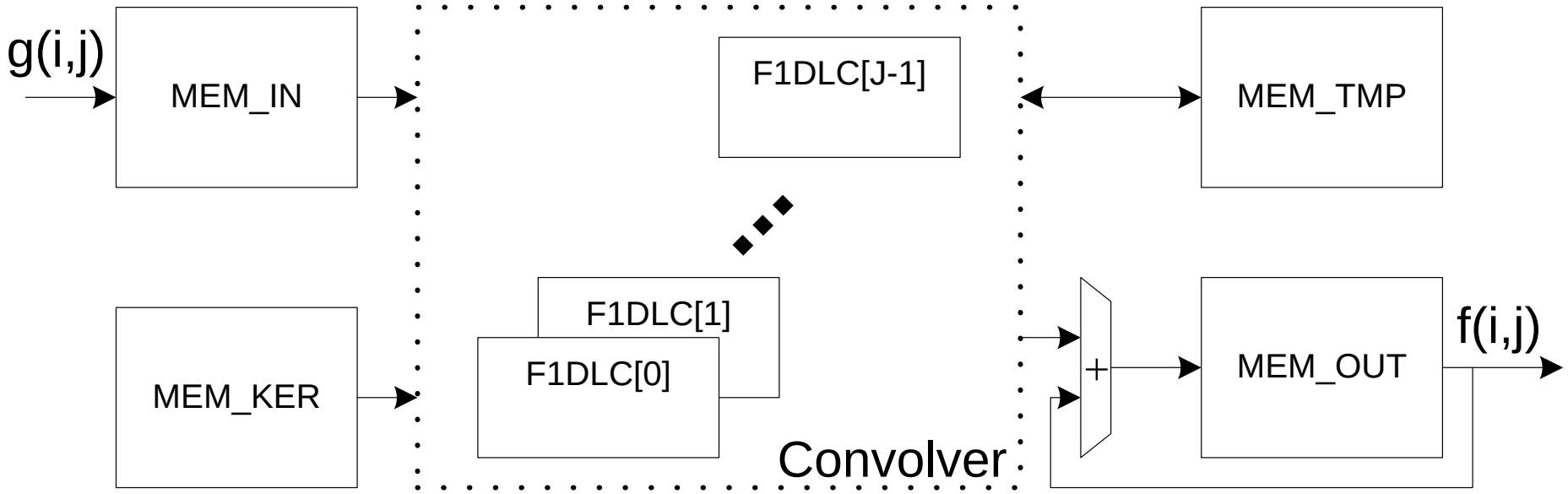
Methods: 1-D Circular convolver (F1DCC)



$$F_m(d) = \sum_{k=0}^{N-1} G_m(k)\, \breve{H}_m^{d+1}(k).$$

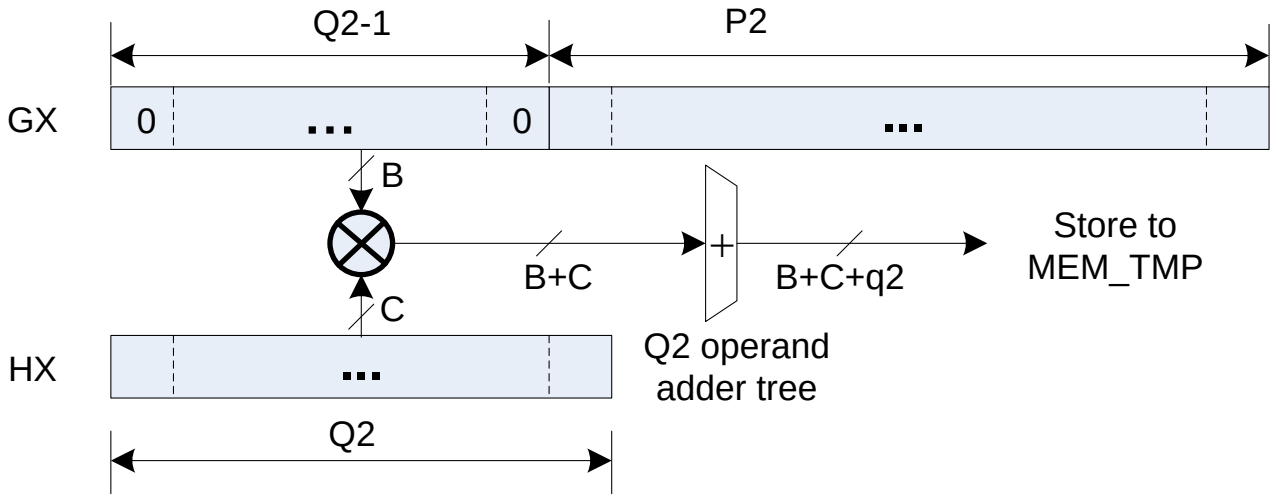
Methods: 2-D Convolution/X-corr LU

Fast and scalable architecture system for computing 2D convolutions/Cross-correlations **using LU decomposition**.

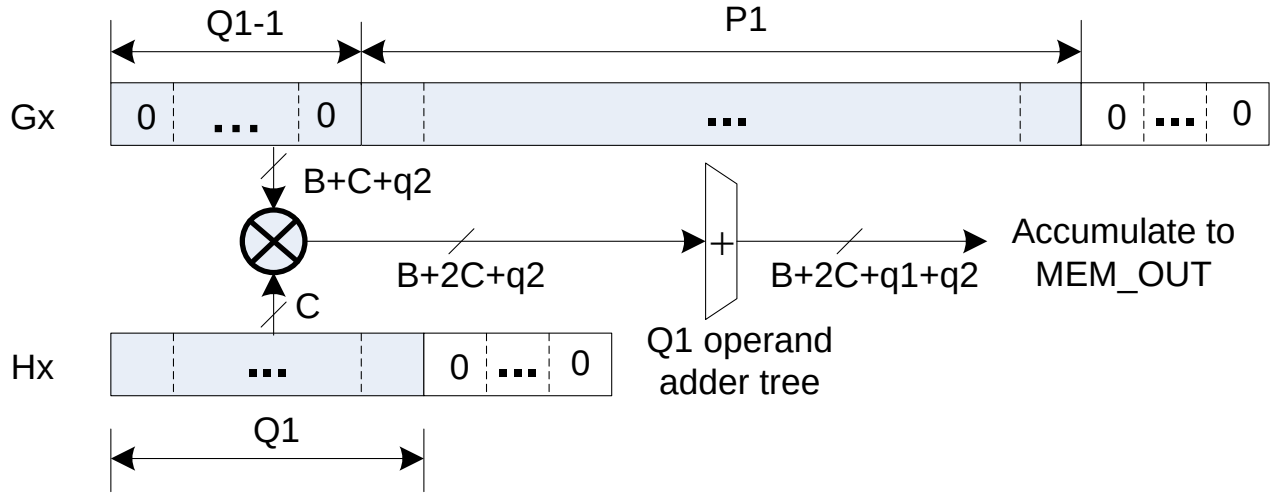


F1DLC: Fast 1-D Linear convolver

Methods: 1-D Linear convolver F1DLC

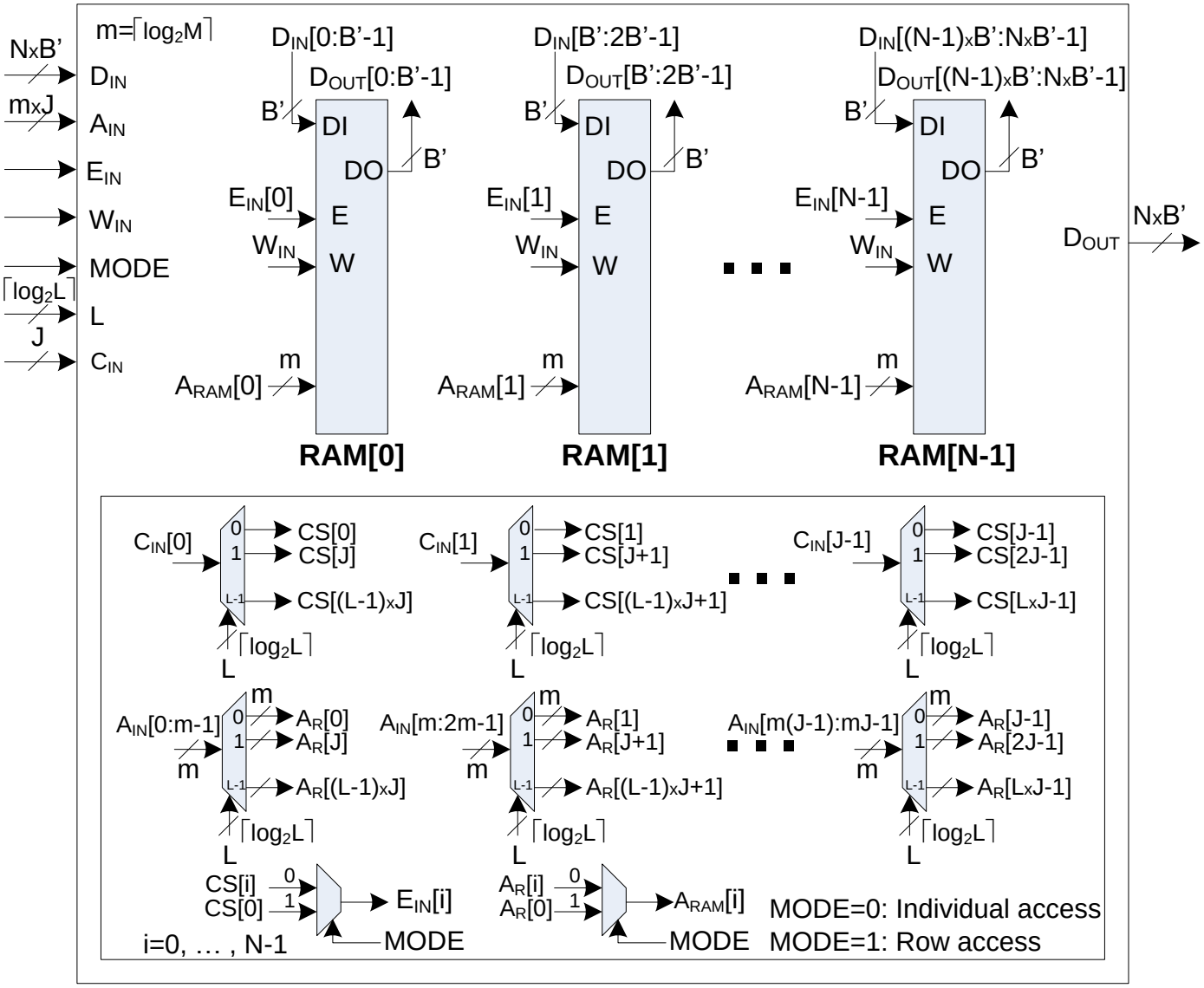


(a)



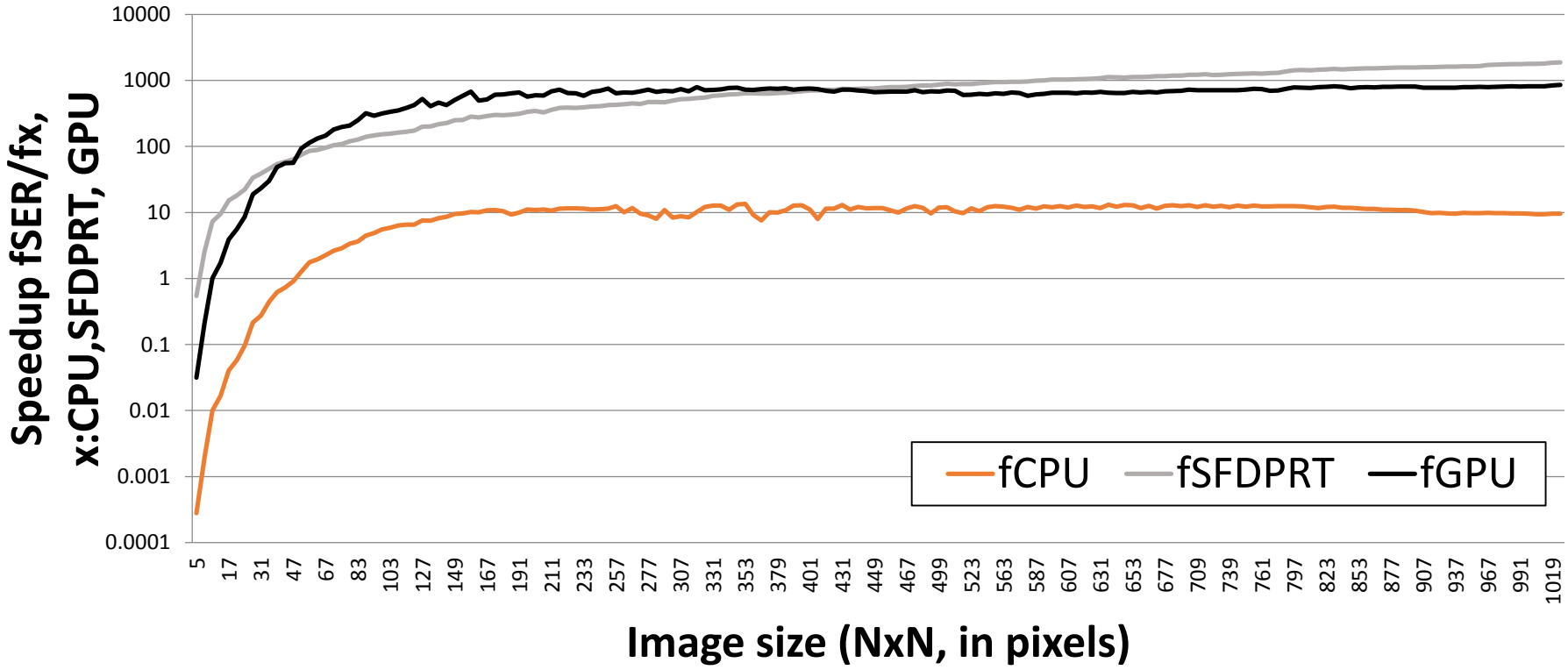
(b)

Temporal/Transpose SRAM



Results – CPU/GPUs

Speedup for the forward DPRT for different implementations with respect to the serial implementation



fSER,fCPU: Xeon E5-2630
8 Cores, 16 Logical cores
f_{CLK} = 3.2GHz

fGPU: GeForce GTX980
2048 cores
f_{CLK} = 1.37GHz

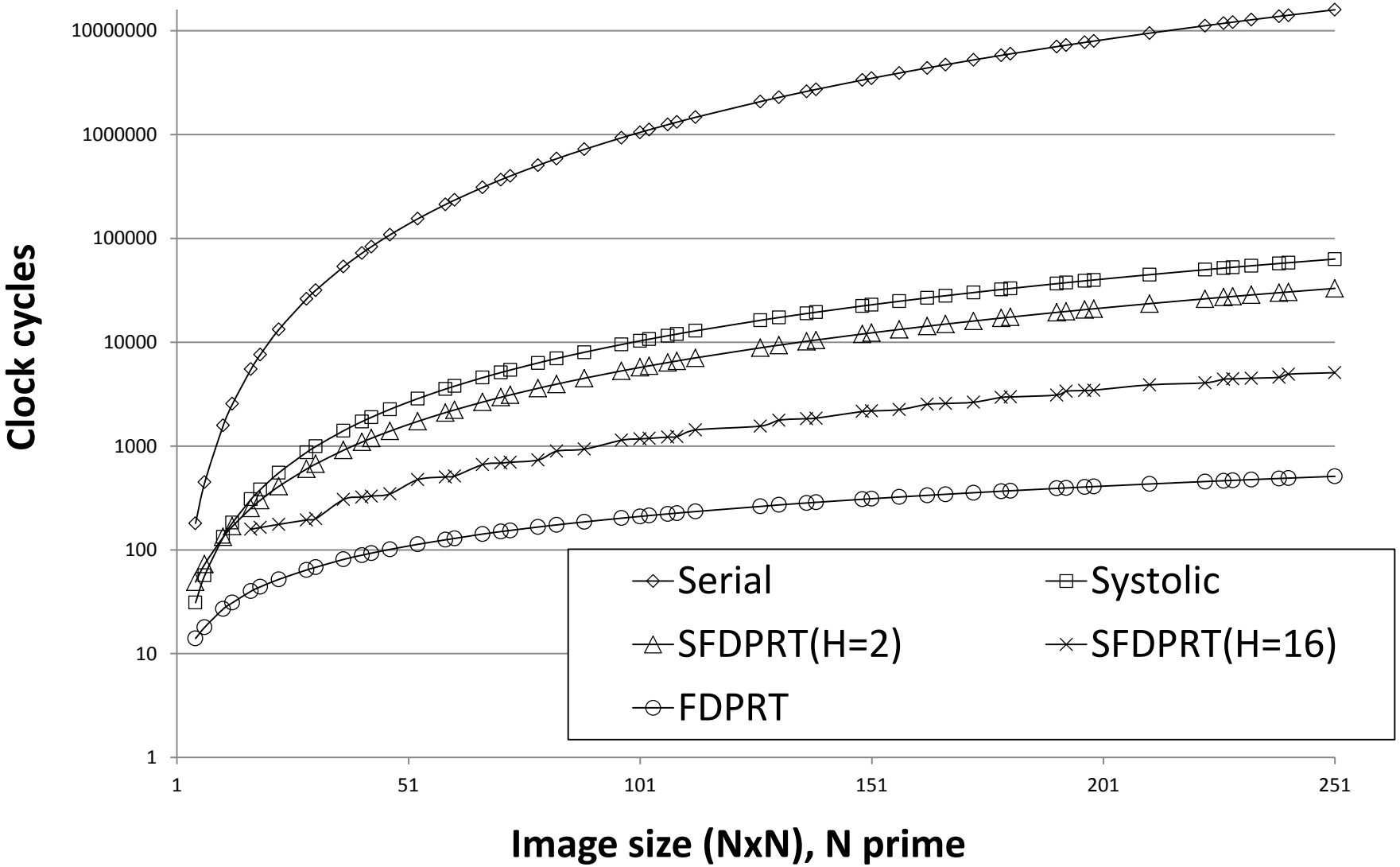
fSFDPRT: Xilinx – Virtex6, H=2
custom
f_{CLK} = 0.1GHz

Results – SFDPRT & iSFDPRT

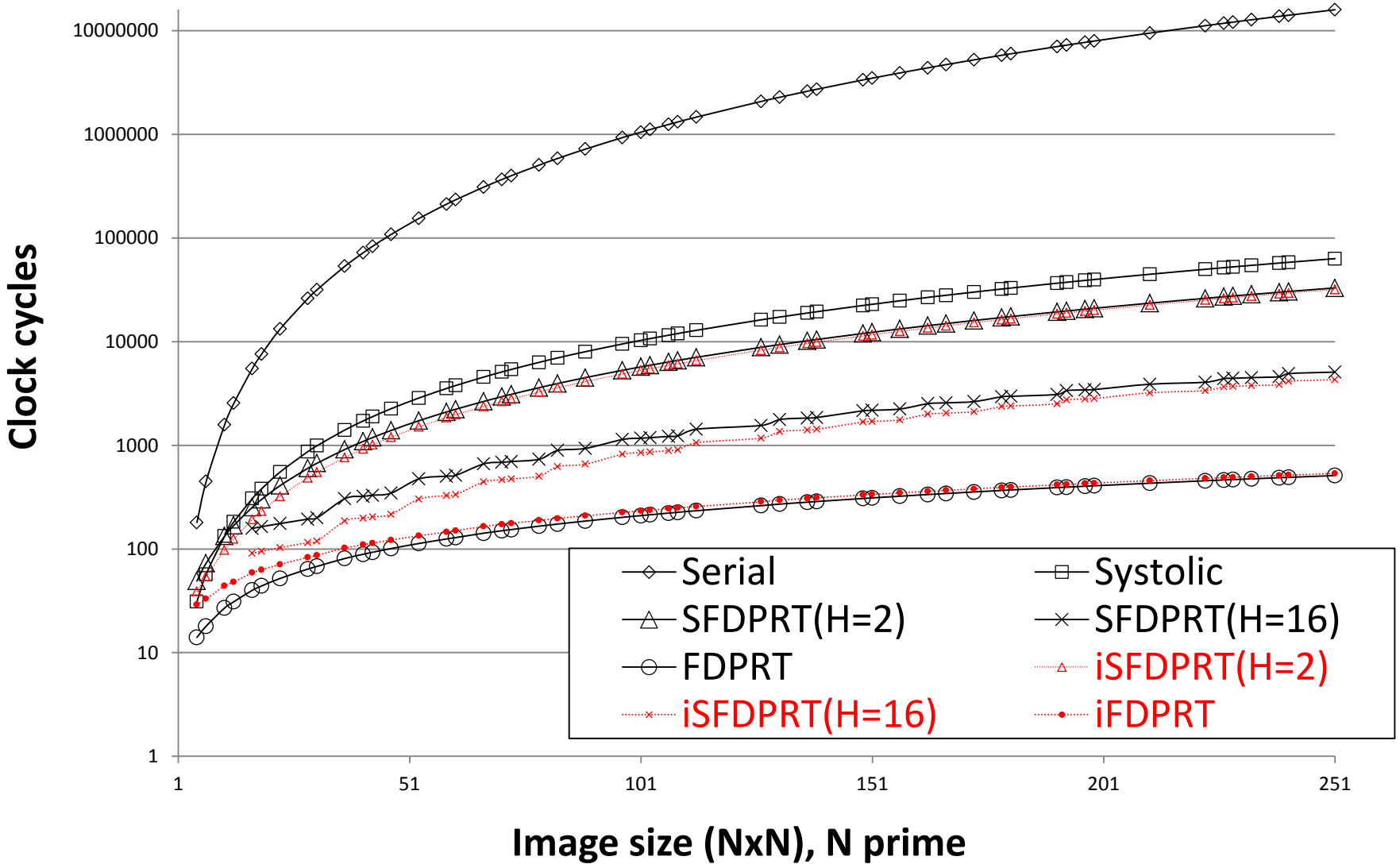
Table 1: Total number of clock cycles for computing the DPRT. In all cases, the image is of size $N \times N$, and $H = 2, \ldots, N$ is the scaling factor for the SFDPRT.

Previous work	Clock cycles
Serial [Matus93],[Chandrasekaran05]	$N^3 + 2N^2 + N$
Systolic [Matus93],[Chandrasekaran08]	$N^2 + N + 1$
This work	
SFDPRT	$\lceil N/H \rceil (N + 3H + 3) + N + \lceil \log_2 H \rceil + 1$
SFDPRT $(H = 2)$lowest resource usage	$\lceil N/2 \rceil (N + 9) + N + 2$
SFDPRT $(H = N)$fastest running time	$5N + \lceil \log_2 N \rceil + 4$
FDPRT	$2N + \lceil \log_2 N \rceil + 1$

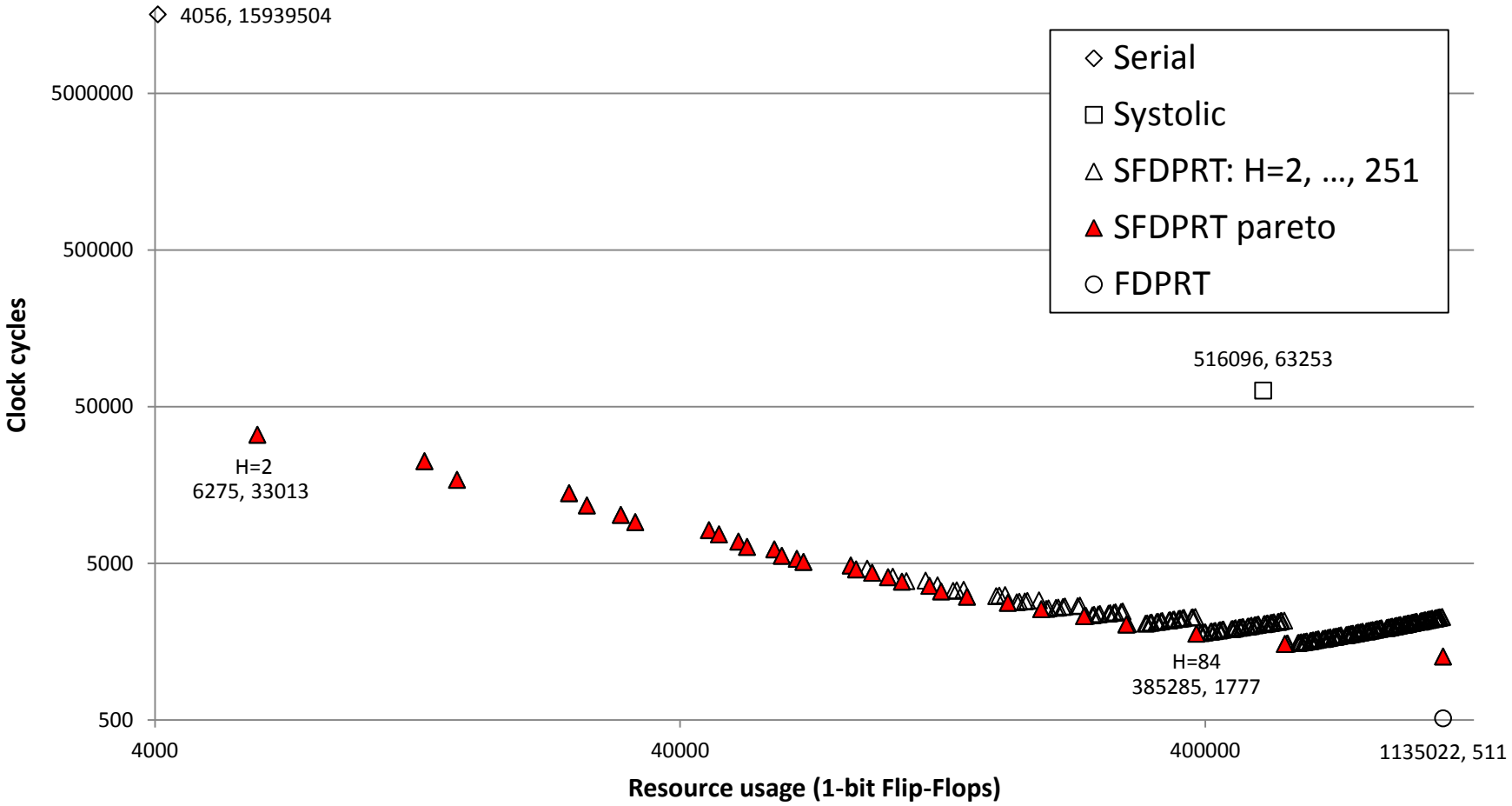
Results – Running time foward DPRT



Results – Running time inverse DPRT

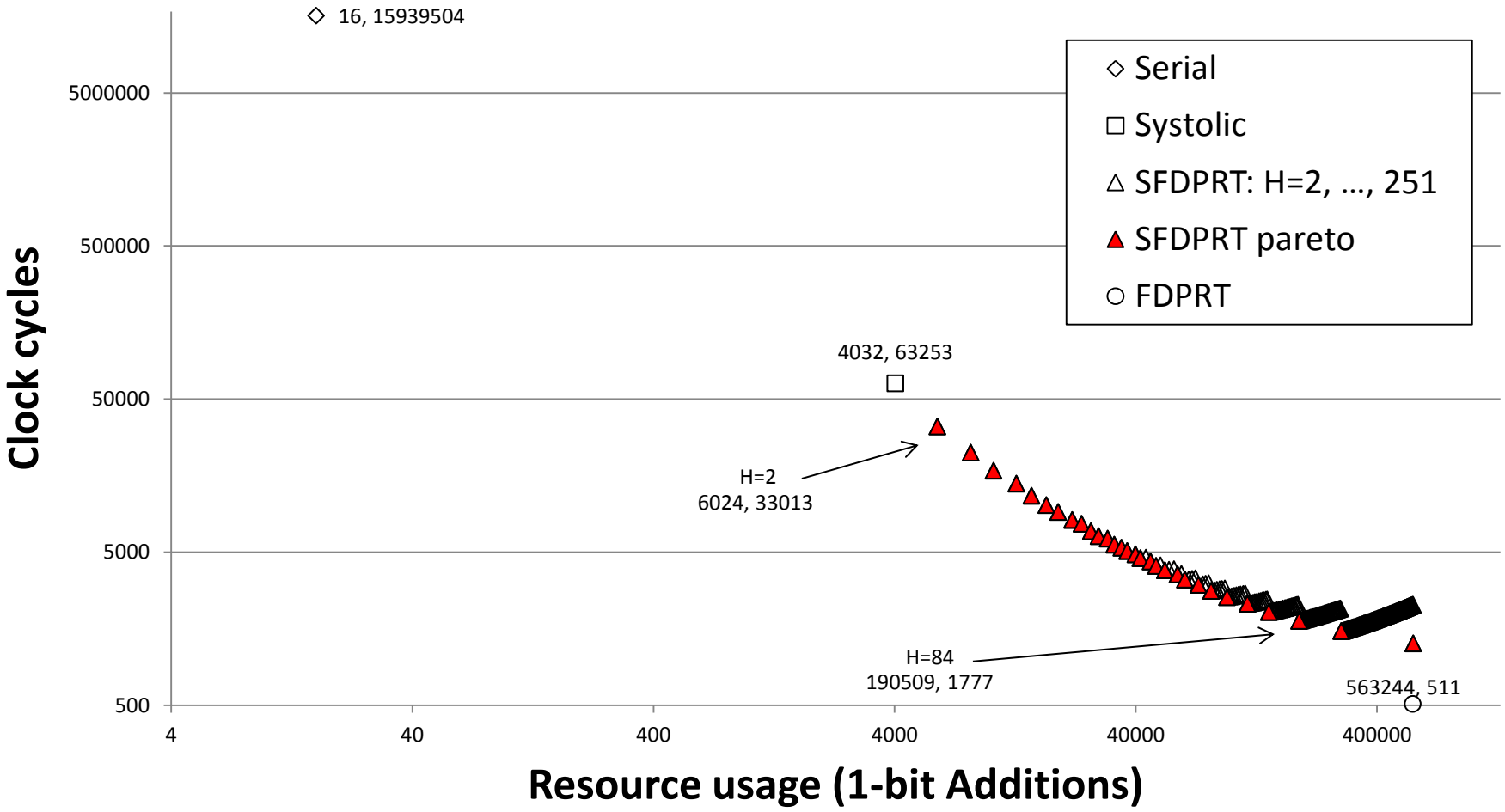


Results - Pareto



Comparative plot for the different implementations based on the number of cycles and the number of flip-flops only. The plot shows the Pareto front (in red) for the proposed SFDPRT for H = 2, …, 251, for an image of size 251x251.

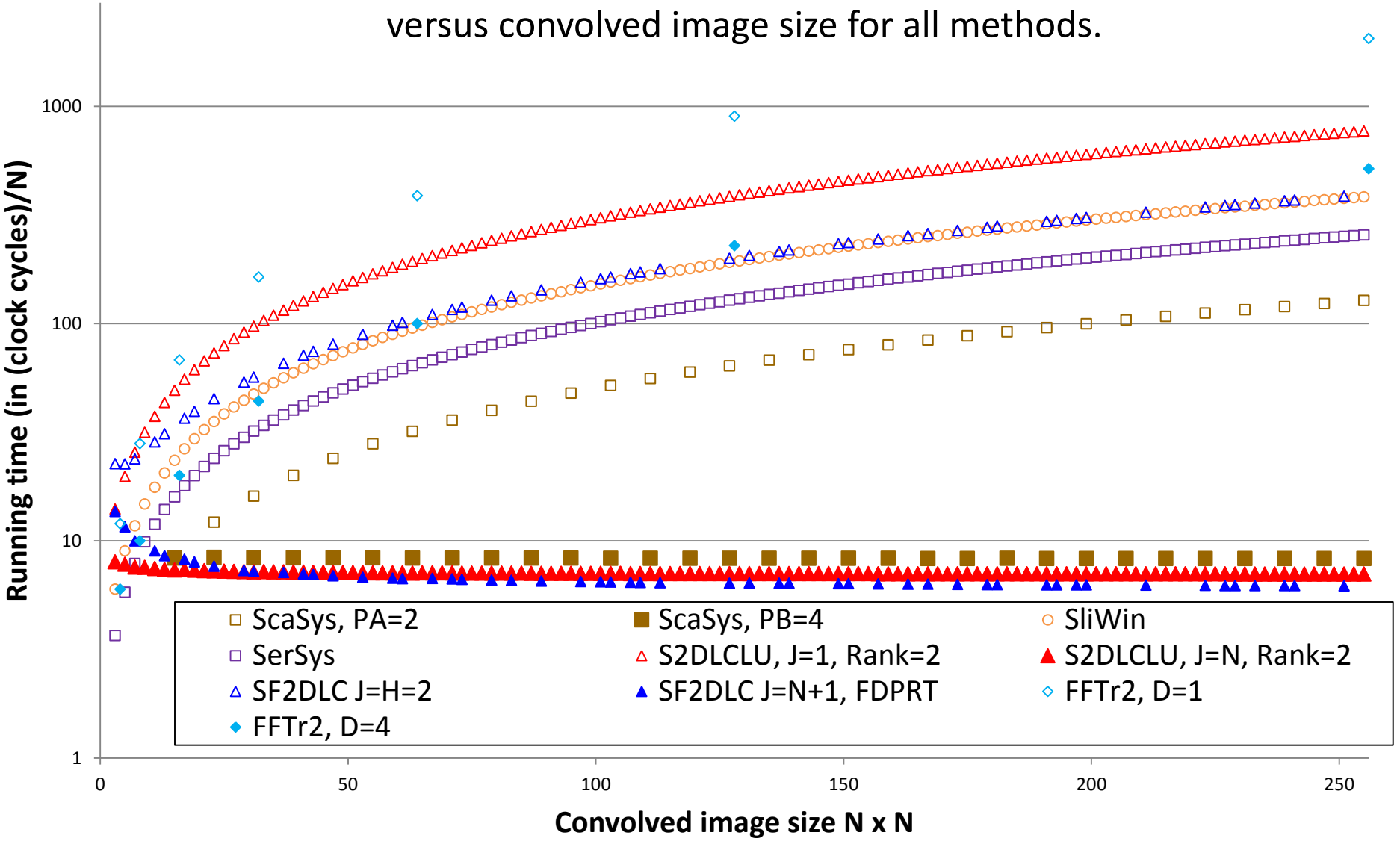
Results - Pareto



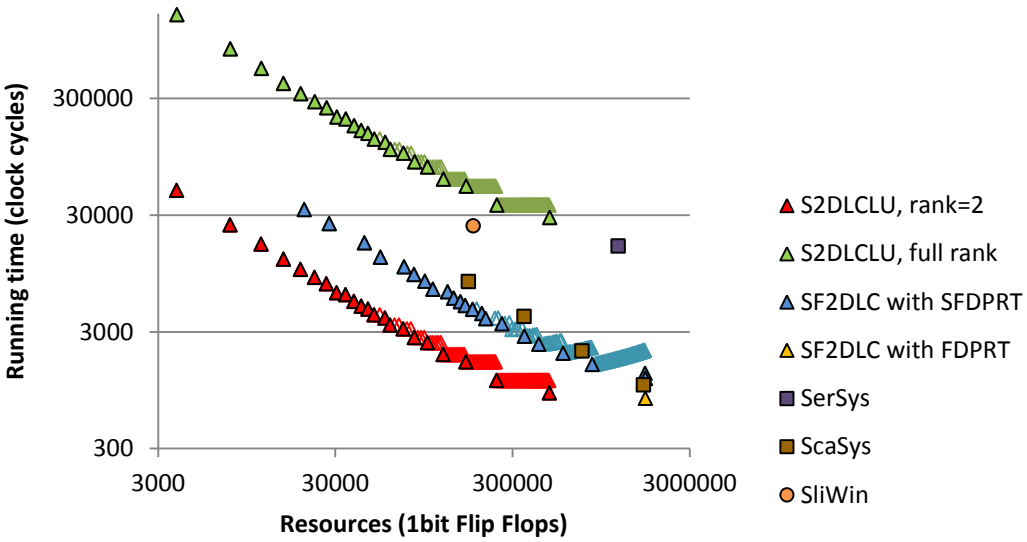
Comparative plot for the different implementations based on the number of cycles and the number of 1-bit additions only. The plot shows the Pareto front (in red) for the proposed SFDPRT for H = 2, …, 251, for an image of size 251x251.

Results 2-D Convolution/X-corr

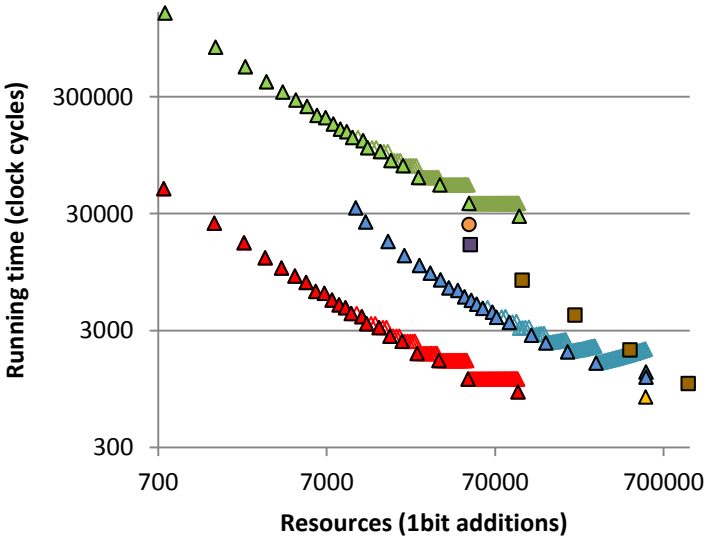
Running time in clock cycles (normalized by image size N) versus convolved image size for all methods.

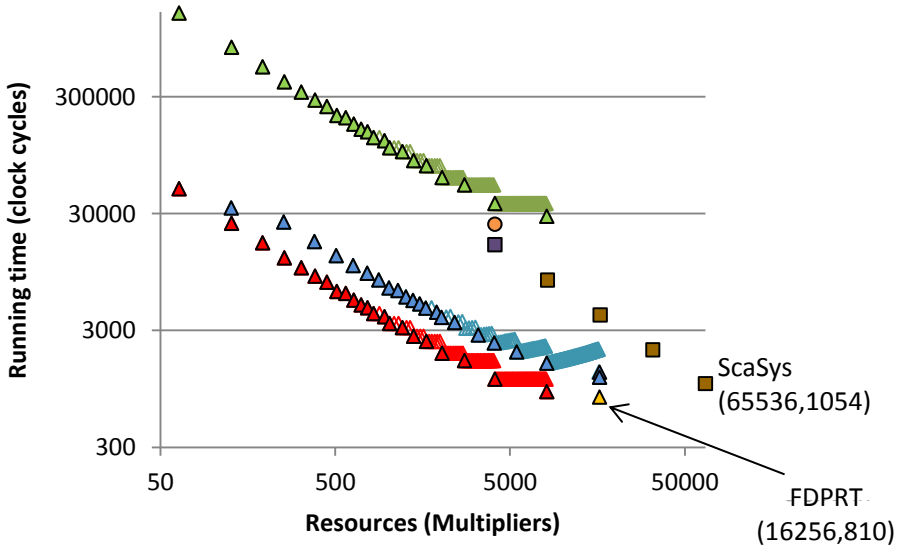


Results 2-D Convolution/X-corr



Pareto front for running time vs different resource usage.
Image and kernel size are 64x64, then N=127





Future work

- Beyond prime N, need $N = 2^m$. Additions reduced to $N^2 \log_2 N$ while prime directions increase to 3N/2 and the inverse needs to be computed iteratively.

- Study multi-objective space defined by the accuracy, performance, and required resources of the DPRT and its applications in 2-D convolutions and cross-correlations.

Concluding remarks

Overall, my dissertation has led to the development of fast and scalable methods for the computation of the DPRT and its inverse.

The fast methods have enabled the application of the DPRT to new areas that were not possible with previous implementations.

Patent and Publications

Patent:

System and Methods for "Scalable and Fast Discrete Periodic Radon Transform", Inventors: Cesar Carranza, Daniel Llamocca, Marios S. Pattichis, Filed Dec. 17, 2013.

Relevant publications:

- **C. Carranza**, D. Llamocca, and M. Pattichis, "Fast and scalable computation of the forward and inverse discrete periodic radon transform," IEEE Transactions on Image Processing, vol. 25, no. 1, pp. 119-133, Jan 2016.
- **C. Carranza**, D. Llamocca, and M. Pattichis , "A scalable architecture for implementing the fast discrete periodic radon transform for prime sized images," in 2014 IEEE International Conference on Image Processing (ICIP), Oct 2014, pp. 1208-1212.
- **C. Carranza**, D. Llamocca, and M. Pattichis, "The fast discrete periodic radon transform for prime sized images: Algorithm, architecture, and vlsi/fpga implementation", in 2014 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI), April 2014, pp. 169-172.

Publications (cont)

- D. Llamocca, M. Pattichis, and **C. Carranza**, “A framework for selfreconfigurable dcts based on multiobjective optimization of the power-performance-accuracy space,” in 2012 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), July 2012, pp. 1–6
- D. Llamocca, **C. Carranza**, and M. Pattichis, “Dynamic multiobjective optimization management of the energy-performance-accuracy space for separable 2-d complex filters,” in 2012 22nd International Conference on Field Programmable Logic and Applications (FPL), Aug 2012, pp. 579–582.
- **C. Carranza**, V. Murray, M. Pattichis, and E. Barriga, “Multiscale am-fm decompositions with gpu acceleration for diabetic retinopathy screening”, in 2012 IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI), April 2012, pp. 121-124.
- D. Llamocca, **C. Carranza**, M. Pattichis, "Separable FIR Filtering in FPGA and GPU Implementations: Energy, Performance, and Accuracy Considerations“, 2011 International Conference on Field Programmable Logic and Applications (FPL), Sept. 2011, pp.363,368,

Future publications

- **Journal submission of Fast 2-D Convolutions and Cross-Correlations Using Scalable Architectures:**
  - FPGA implementations is under development, it will be added to the work presented in this manuscript and submitted to IEEE Transactions on Image Processing.
- **Journal submission of DPRT on Multi-core CPU and GPUs**
  - The new parallel algorithms proposed here to compute the DPRT and its inverse using GPUs will be applied to image filtering with large and non-separable kernels and submitted to: TBD.
- **Journal submission of DRASTIC applied to the forward and inverse DPRT.**
  - Accuracy will be added in the multi-objective optimization to generate scalable, fast and accurate architectures for the computation of the DPRT.
- **Two provisional patents submitted:**
  - Parallel algorithms on the GPU
  - Scalable and Fast architectures and algorithms for 2D Convolutions/Cros-correlations.