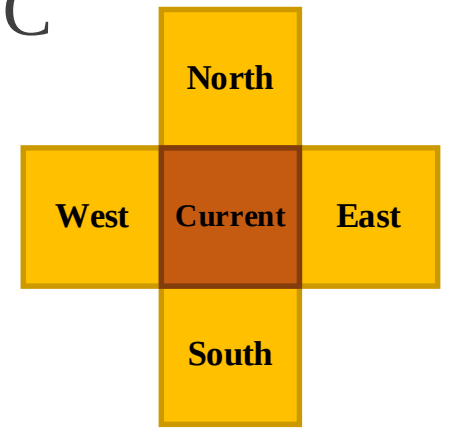
Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL

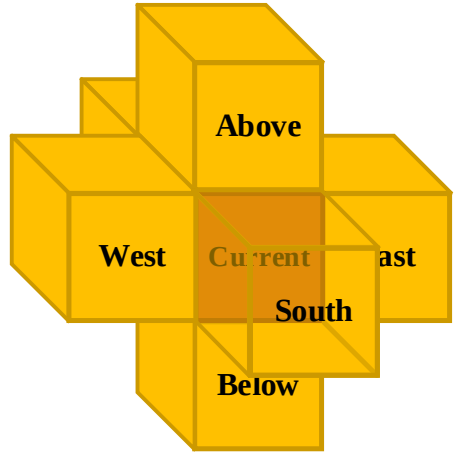
Authors: Hamid Reza Zohouri, Artur Podobas, Satoshi Matsuoka

Introduction

Stencil Computation

- One of the most widely-used computation patterns in HPC
  - Weather, wave and seismic simulations
  - Fluid simulations
  - Image processing
  - Convolutional Neural Networks
- Calculates a weighted sum of *coeff* × *cell_value*
- Exhibits good spatial and temporal locality
  - Spatial and temporal blocking is widely employed to take advantage
- Generally memory-bound for low-order
  - Very high Byte-to-FLOP ratio





Motivation

- FPGAs have very low Byte-to-FLOP ratio among modern accelerators
  - Even high-order stencil computation is memory-bound
- Many recent implementations on FPGAs use temporal blocking to alleviate the bandwidth bottleneck, but they avoid spatial blocking
  - Design simplicity
  - Linear performance scaling with degree of temporal parallelism
  - Row size (2D) or plane size (3D) is limited based on FPGA on-chip memory size → unacceptable for real-world applications, especially in HPC

Contributions

- We combine spatial and temporal blocking in an HLS-based design
  - High performance with no input size restriction

Contributions

- We combine spatial and temporal blocking in an HLS-based design
  - High performance with no input size restriction
- Design complexity increases significantly → Area overhead and low f_{max}
  - We employ multiple HLS-based FPGA-specific optimizations

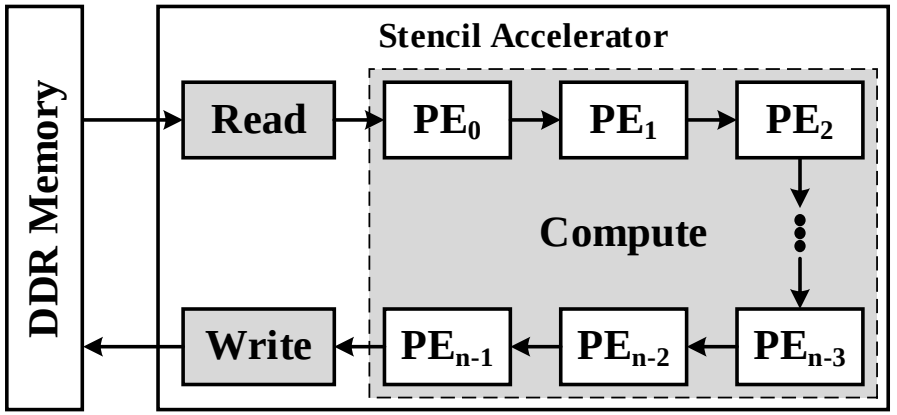
Contributions

- We combine spatial and temporal blocking in an HLS-based design
  - High performance with no input size restriction
- Design complexity increases significantly → Area overhead and low f_{max}
  - We employ multiple HLS-based FPGA-specific optimizations
- Multiple parameters to tune + long place and route time → Long performance tuning time
  - We devise a performance model to minimize parameter search space

Implementation

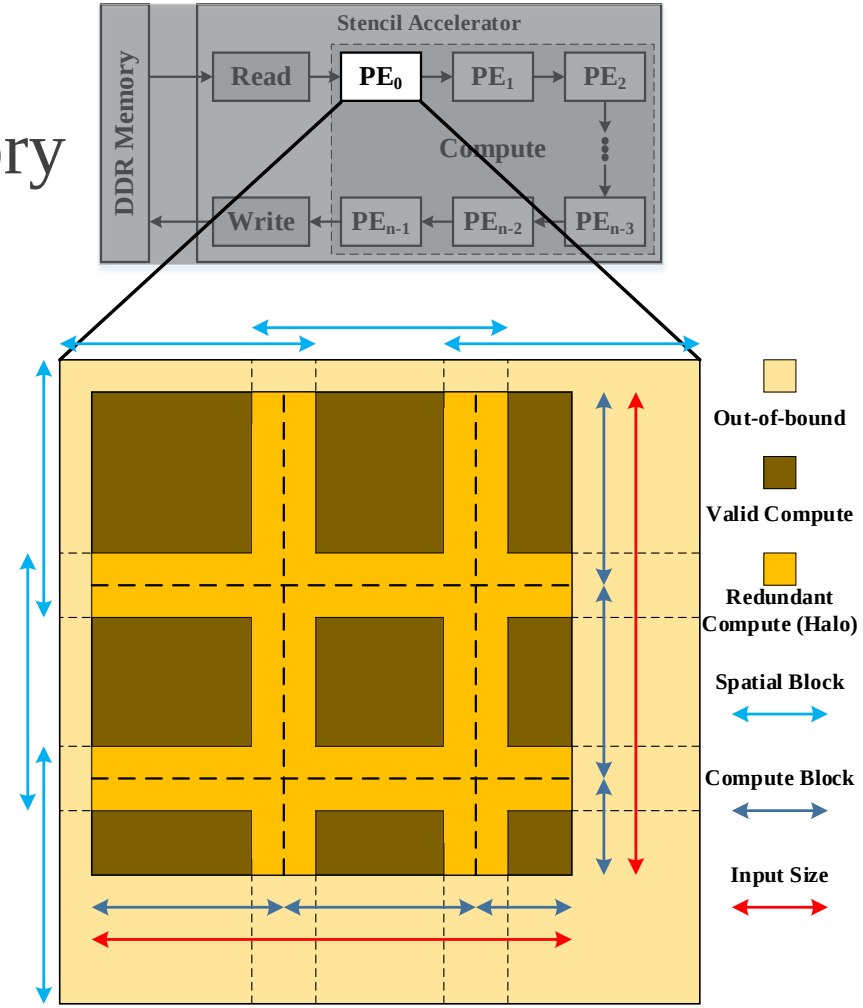
Overview

- OpenCL-based design with Intel FPGA SDK for OpenCL
- Multi-kernel single work-item streaming design
  - Two kernels for memory access
  - One kernel for compute, replicated in form of multiple PEs
  - Streaming via on-chip channels



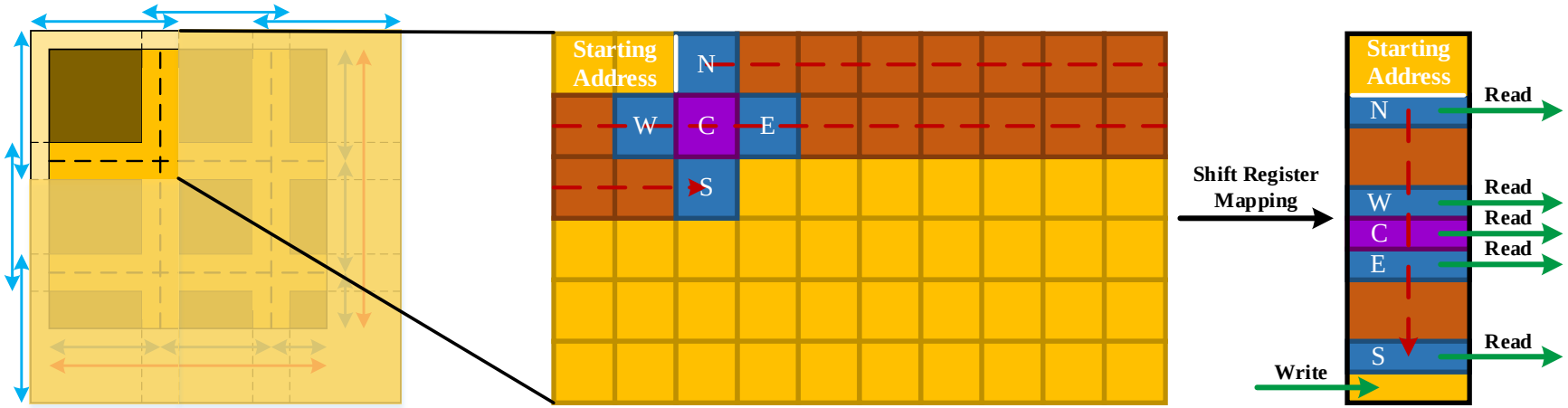
Spatial Blocking

- Neighbor cells are kept on-chip and reused
  - Avoids redundant accesses to external memory
- One spatial dimension is streamed
- Other dimensions are blocked
- Blocks are overlapped
  - Avoid communication/synchronization
- Parameter: **block size**



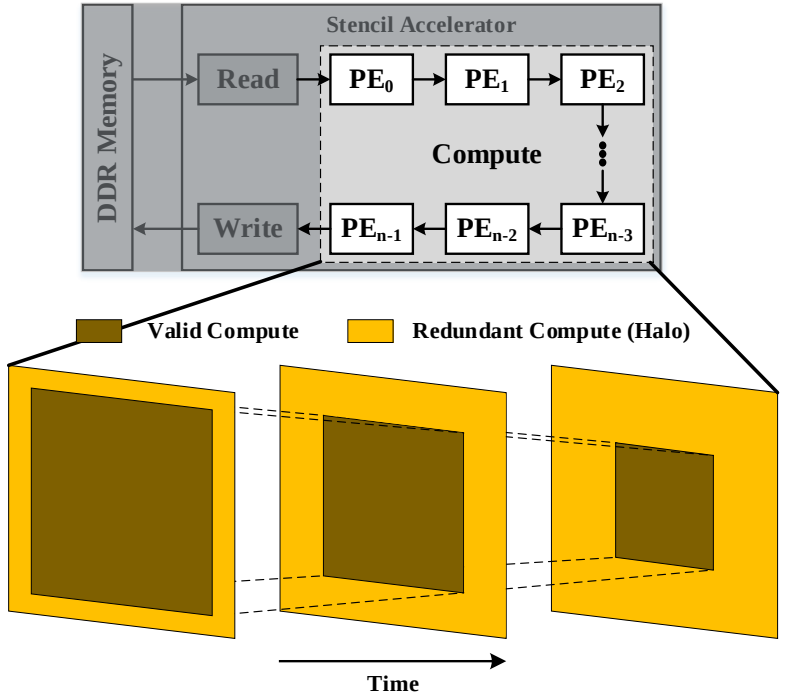
Spatial Blocking (Cont.)

- On-chip buffer is configured as shift register
  - Stencil is shifted forward by incrementing starting address of on-chip buffer
  - Minimum on-chip memory size: 2 block rows for 2D and 2 block planes for 3D
  - Maps well to FPGA architecture
  - Not possible on CPUs/GPUs
- Computation is vectorized in the *x* dimension
  - Parameter: **vector size**



Temporal Blocking

- Multiple time steps (iterations) are combined
  - External memory accesses between them are avoided
- Compute kernel defined as *autorun*
- Replicated into multiple PEs
- Each PE works on a different but consecutive time-step
- Halo size increases with number of PEs
- No thread divergence, no need for warp specialization
- Parameter: **degree of temporal parallelism**
  - Equal to number of PEs



FPGA-Specific Optimizations

FPGA-Specific Optimizations

```
for (y = 0; y < m; y++) {
   for(x = 0; x < n; x++) {
      compute (x,y);
   }
}
```

FPGA-Specific Optimizations

- Loop collapse (1)
  - Avoids area overhead of multiply-nested loops

```
for (y = 0; y < m; y++) {
   for(x = 0; x < n; x++) {
      compute (x,y);
   }
}
```


```
int x = 0, y = 0;
while (y != m) {
   compute (x,y);
   x++;
   if (x == n) {
      x = 0;
      y++;
   }
}
```

FPGA-Specific Optimizations

- Loop collapse (1)
  - Avoids area overhead of multiply-nested loops
- Exit condition optimization (2)
  - Simplifies exit condition
  - Shortens critical path and improves f_{max}

```
for (y = 0; y < m; y++) {
   for(x = 0; x < n; x++) {
      compute (x,y);
   }
}
```

(1)

```
int x = 0, y = 0;
while (y != m) {
   compute (x,y);
   x++;
   if (x == n) {
      x = 0;
      y++;
   }
}
```

(2)

```
int x = 0, y = 0, index = 0;
while (index != m * n) {
   index ++;
   compute (x,y);
   x ++;
   if (x == n) {
      x = 0;
      y ++;
   }
}
```

FPGA-Specific Optimizations

- Loop collapse (1)
  - Avoids area overhead of multiply-nested loops
- Exit condition optimization (2)
  - Simplifies exit condition
  - Shortens critical path and improves f_{max}
- Padding
  - Overlapped blocking results in unaligned accesses
  - Padding helps improve memory bandwidth

```
for (y = 0; y < m; y++) {
   for(x = 0; x < n; x++) {
      compute (x,y);
   }
}
```

(1)

```
int x = 0, y = 0;
while (y != m) {
   compute (x,y);
   x++;
   if (x == n) {
      x = 0;
      y++;
   }
}
```

(2)

```
int x = 0, y = 0, index = 0;
while (index != m * n) {
   index ++;
   compute (x,y);
   x ++;
   if (x == n) {
      x = 0;
      y ++;
   }
}
```

Performance Model

Performance Model

Performance Model

$$run\_time(s) = \text{——————————}$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{}$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) =$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write})$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write}) \times size_{cell}$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write}) \times size_{cell}$$

$$th_{mem}(GB/s) =$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write}) \times size_{cell}$$

$$th_{mem}(GB/s) = \min(\quad)$$

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write}) \times size_{cell}$$

$$th_{mem}(GB/s) = \min(th_{max},$$)

Performance Model

$$run\_time(s) = \frac{t_{transfer}}{10^9 \times th_{mem}}$$

$$t_{transfer}(B) = \left\lceil \frac{iter}{par_{time}} \right\rceil \times (t_{read} + t_{write}) \times size_{cell}$$

$$th_{mem}(GB/s) = \min(th_{max}, \frac{f_{max} \times par_{vec} \times size_{cell} \times num_{acc}}{10^9})$$

Methodology

Software

- FPGA
  - Quartus and AOC v16.1.2; v17.0+ exhibit significant performance regression
- GPU
  - Highly-optimized code from [1] (with temporal blocking)
  - CUDA 8.0/9.0
- Xeon/Xeon Phi
  - State-of-the-art YASK framework [2] (temporal blocking exists but is ineffective)
  - Intel Compiler 2018.1

[1] N. Maruyama and T. Aoki, "Optimizing Stencil Computations for NVIDIA Kepler GPUs," in *Proceedings of the 1st International Workshop on High-Performance Stencil Computations (HiStencils'14)*, Vienna, Austria, 2014, pp. 89-95.

[2] C. Yount, J. Tobin, A. Breuer and A. Duran, "YASK—Yet Another Stencil Kernel: A Framework for HPC Stencil Code-Generation and Tuning," *Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC)*, Salt Lake City, UT, 2016, pp. 30-39.

Benchmarks

Benchmark	FLOP per cell update	Bytes per cell update	$\frac{\text{Bytes}}{\text{FLOP}}$
Diffusion 2D	9	8	0.889
Diffusion 3D	13	8	0.615
Hotspot 2D	15	12	0.800
Hotspot 3D	17	12	0.706

Hardware

Device	Peak Memory Bandwidth (GB/s)	Peak Compute Performance (GFLOP/s)	$\frac{\text{Bytes}}{\text{FLOP}}$	TDP
Stratix V GX A7	25.6	200	0.128	40
Arria 10 GX 1150	**34.1**	**1450**	**0.024**	**70**
Xeon E5-2650 v4	76.8	350	0.219	105
Xeon Phi 7210F	400	5325	0.075	235
Tesla K40c	288.4	4300	0.067	235
GTX 980Ti	336.6	6900	0.049	275
Tesla P100 PCI-E	720.9	9300	0.078	250
Tesla V100 SXM2	900.1	14900	0.060	300

Results

FPGA Results

Device	Kernel	bsize	par_{vec}	par_{time}	Performance (GFLOP/s)	Logic|M20K|DSP	f_{max} (MHz)	Power (Watt)	Model Accuracy
Stratix V	Diffusion 2D	4096	2	24	112.030	69%| 52%| 95%	302.48	29.845	86.8%
	Hotspot 2D	4096	2	20	140.273	55%| 83%| 95%	269.97	33.361	87.0%
	Diffusion 3D	256x256	8	4	101.457	84%| 61%| 64%	301.02	21.135	82.8%
	Hotspot 3D	256x256	8	4	90.104	47%| 94%| 95%	246.18	36.126	68.7%
Arria 10	Diffusion 2D	**4096**	**8**	**36**	**758.204**	**62%| 67%| 91%**	**343.76**	**72.530**	**86.3%**
	Hotspot 2D	4096	4	36	592.865	60%|100%| 89%	322.47	50.129	86.6%
	Diffusion 3D	**256x256**	**16**	**12**	**374.673**	**76%|100%|100%**	**286.61**	**71.628**	**60.8%**
	Hotspot 3D	128x128	16	8	323.211	62%|100%| 96%	296.20	73.398	64.2%

- 2D 2x faster than 3D
- Arria 10 3-4x faster than Stratix V
- 2D: Big block size → low redundancy with temporal blocking → par_{time} scales better than par_{vec}
- 3D: Small block size → high redundancy with temporal blocking → par_{vec} scales better than par_{time}

Model Accuracy

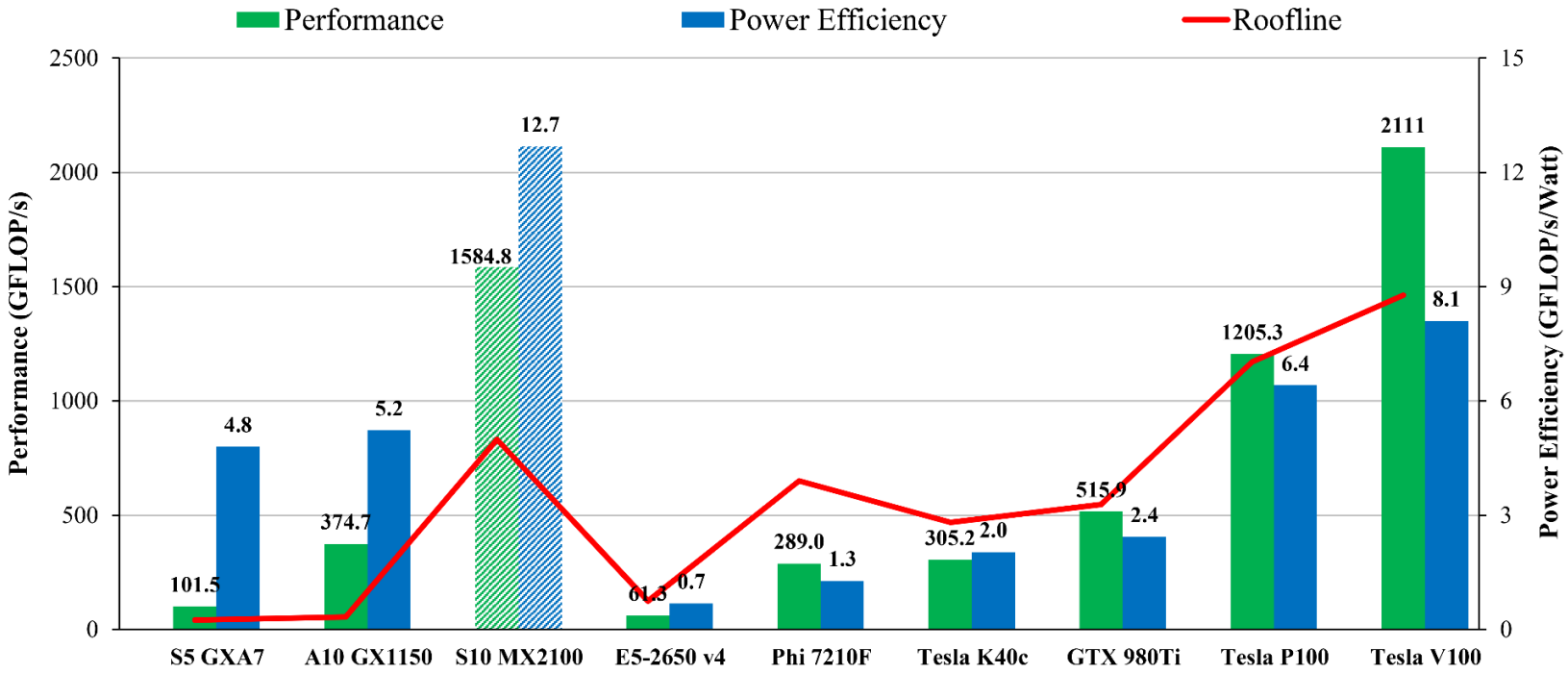
- Model accuracy = pipeline efficiency
- 80-90% for 2D
- 60-80% for 3D
- Sources of discrepancy
  - Non-linear improvement in memory bandwidth when $f_{max} > freq_{mem}$
  - More reads than writes
  - Non-aligned vector accesses split into two smaller accesses
  - All related to memory controller/interface
- Profiling shows average burst size hardly goes beyond 8
  - Lower pipeline efficiency for 3D stencils that rely on larger vector size

Performance Projection for Stratix 10

- DSP and Block RAM utilization is projected based on Arria 10 results
- Operating frequency is assumed to increase by ~100 MHz compared to Arria 10
  - Hyperflex will have limited effectiveness since critical path is in the design
- Model accuracy is used as correction factor

Device	Kernel	bsize	par_{vec}	par_{time}	Estimated Performance (GFLOP/s)	f_{max} (MHz)	Utilized Memory Bandwidth	Resource Bottleneck
GX 2800	**Diffusion 2D**	**8192**	**8**	**140**	**3558.0**	**450**	**38%**	**DSP**
	Hotspot 2D	8192	4	140	2953.5	450	28%	DSP
	Diffusion 3D	256x256	32	24	1490.8	400	100%	DSP
	Hotspot 3D	256x256	16	24	1230.8	400	100%	Block RAM
MX 2100	Diffusion 2D	8192	8	92	2338.5	450	6%	DSP & Block RAM
	Hotspot 2D	8192	4	92	1943.8	450	4%	DSP & Block RAM
	Diffusion 3D	**512x512**	**128**	**4**	**1584.8**	**400**	**80%**	**DSP**
	Hotspot 3D	256x256	32	12	1404.1	400	30%	DSP & Block RAM

Comparison with Other Hardware



- Arria 10 faster than K40c/Xeon Phi 7210F and more power efficient than 980 Ti
  - Despite over 8 times lower memory bandwidth
- Stratix 10 MX 2100 likely faster than P100 and more power efficient than V100
- Temporal blocking has good scaling on FPGAs, moderate scaling on GPUs, and no scaling on Xeon/Xeon Phi

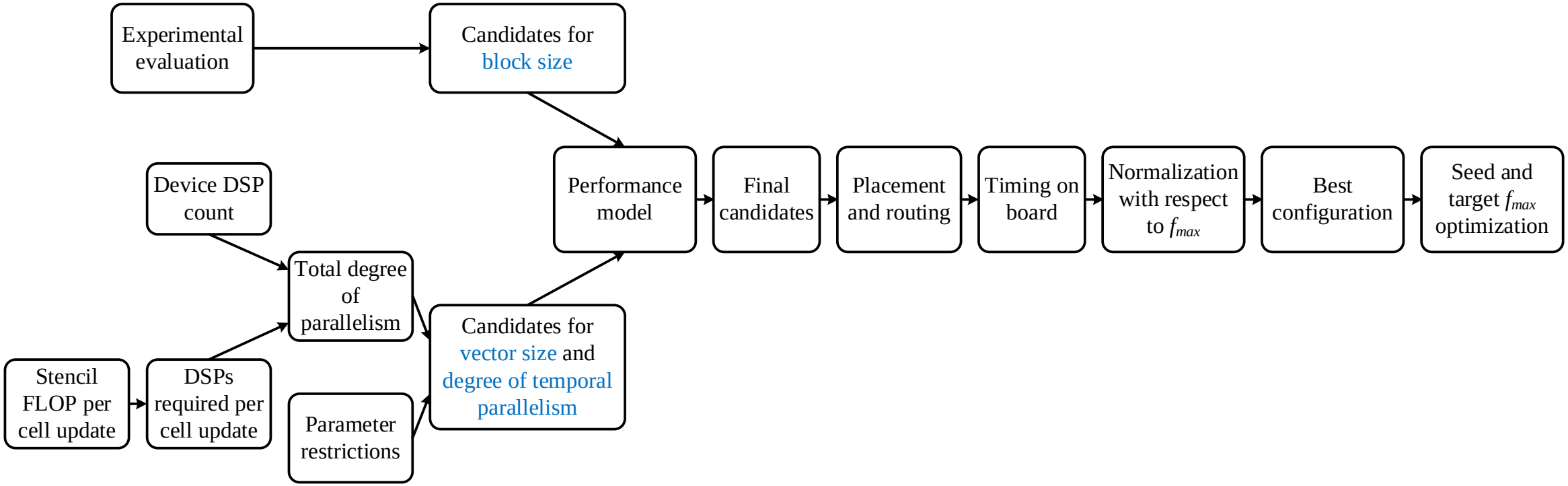
Conclusion

Conclusion

- Combined spatial and temporal blocking
  - High performance and unrestricted input size
- Multiple HLS-based optimizations
  - Lower logic overhead, very high f_{max}, improved bandwidth efficiency
- 3D stencils scale better with larger vector size
  - Increasing memory bandwidth is more effective than increasing FPGA area
- 2D stencils scale better with degree of temporal parallelism
  - Increasing FPGA area is more effective than increasing memory bandwidth
- Temporal blocking scales better on FPGAs compared to other hardware
  - Allows competitive performance despite much lower memory bandwidth
- Highest single-FPGA performance to date: 760 (2D) and 375 (3D) GFLOP/s on Arria 10
- Estimated 3.5 (2D) and 1.5 (3D) TFLOP/s on Stratix 10

Backup

Parameter Tuning for FPGA



Timing and Power Measurement

- Only kernel runtime is measured
  - Initialization and host/device transfers are ignored
- Stratix V: Estimated FPGA power + Maximum power of external memory
- Arria 10: Full board power measured using on-board sensor
- GPUs: Full board power measured using NVML library
- Xeon: CPU-only power measured using MSR library
- Xeon Phi: Processor and on-package memory power measured using MSR library
- Only Diffusion 3D is used for hardware comparison
  - No highly-optimized GPU code available for others

Performance Scaling with Temporal Blocking