Automatic Throughput Optimization of High-Level-Synthesized Accelerators

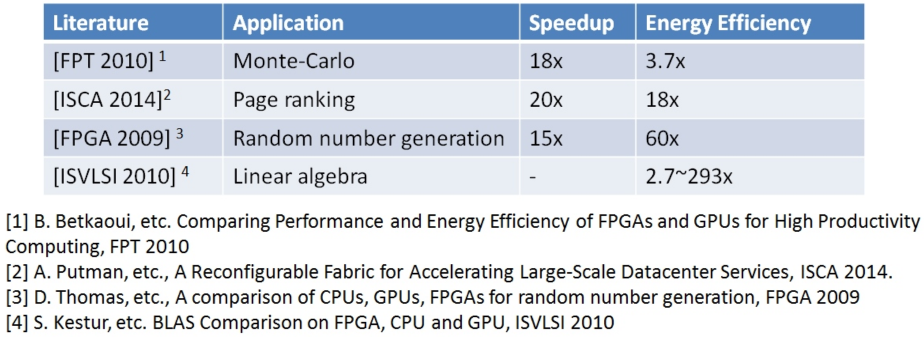
Peng Li[1], Peng Zhang[1], Louis-Noel Pouchet[2,1], Jason Cong[1]

[1]University of California, Los Angeles
[2]The Ohio State University

Feb 24, 2015

Accelerators

- Orders of magnitude energy efficiency
  - Lower frequency
  - Less dynamic behavior

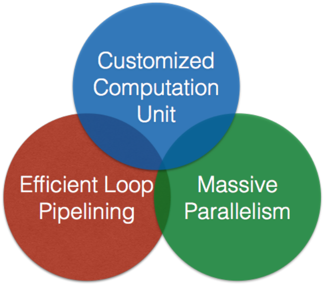


Literature	Application	Speedup	Energy Efficiency
[FPT 2010] 1	Monte-Carlo	18x	3.7x
[ISCA 2014]2	Page ranking	20x	18x
[FPGA 2009] 3	Random number generation	15x	60x
[ISVLSI 2010] 4	Linear algebra	-	2.7~293x

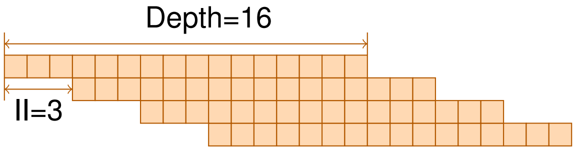
[1] B. Betkaoui, etc. Comparing Performance and Energy Efficiency of FPGAs and GPUs for High Productivity Computing, FPT 2010
[2] A. Putman, etc., A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, ISCA 2014.
[3] D. Thomas, etc., A comparison of CPUs, GPUs, FPGAs for random number generation, FPGA 2009
[4] S. Kestur, etc. BLAS Comparison on FPGA, CPU and GPU, ISVLSI 2010

Accelerators

- Orders of magnitude energy efficiency
  - Lower frequency
  - Less dynamic behavior
- Significant speedup



Efficient Loop Pipelining



- Minimize Loop II can boost the performance of loops
  - Loop transformations to remove dependence
  - Array partitions to remove port conflict

Massive Parallel Architecture

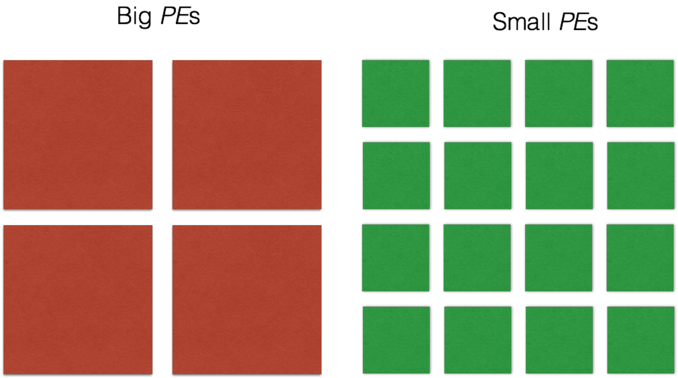
- Extract parallelism from application
- Duplicate hardware PEs

Massive Parallel Architecture

- Extract parallelism from application
- Duplicate hardware PEs
- Traditional design flow: optimize-then-duplicate

Massive Parallel Architecture

- Extract parallelism from application
- Duplicate hardware PEs
- Traditional design flow: optimize-then-duplicate
- Co-optimization of loop pipelining and module duplication

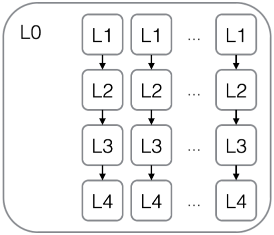


Motivational Example: DWT

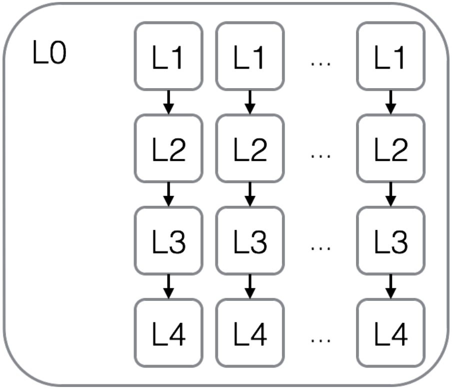
Discrete Wavelet Transformation

```
L0:for i=0 to n-1 do
  L1:for j = 1 to m-3 step 2 do
    tmp[i][j] += a1 * (tmp[i][j-1] + tmp[i][j+1]);
  ... ...
  L2:for j = 2 to m-1 step 2 do
    tmp[i][j] += a2 * (tmp[i][j-1] + tmp[i][j+1]);
  ... ...
  L3:for j = 1 to m-3 step 2 do
    tmp[i][j] += a3 * (tmp[i][j-1] + tmp[i][j+1]);
    img[j/2+m/2][i] = k2 * tmp[i][j];
  ... ...
  L4:for j = 2 to m-1 step 2 do
    tmp[i][j] += a4 * (tmp[i][j-1] + tmp[i][j+1]);
    img[j/2][i] = k1 * tmp[i][j];
  ... ...
```

	+	*
L1	2	1
L2	2	1
L3	2	2
L4	2	2



Dependence Analysis: DWT



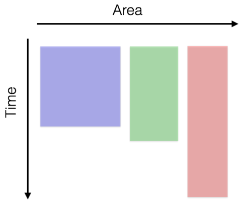
- L0: Fully parallel
- Data dependence between L1, L2, L3 and L4
    - Resource sharing between loops

Performance-Area Tradeoffs in Loop Pipelining

	II_{L1}	II_{L2}	II_{L3}	II_{L4}	LUT	FF	DSP	Cycles
1	1	1	1	1	3379	2236	28	590337
2	1	1	1	2	3261	2364	28	720897
3	1	1	2	1	3260	2363	28	720897
4	1	1	2	2	3003	1903	17	851457
5	1	2	1	1	3325	2239	28	720897
6	1	2	1	2	3207	2367	28	851457
7	1	2	2	1	3206	2366	28	851457
8	1	2	2	2	2885	1939	17	982017
9	2	1	1	1	3324	2238	28	720897
10	2	1	1	2	3206	2366	28	851457
11	2	1	2	1	3222	2365	28	851457
12	2	1	2	2	2901	1938	17	982017
13	2	2	1	1	3270	2241	28	851457
14	2	2	1	2	3088	2369	28	982017
15	2	2	2	1	3104	2368	28	982017
16	2	2	2	2	2066	1496	14	1112577

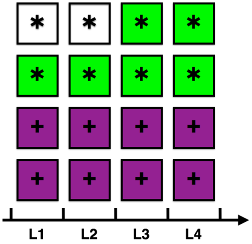
Pareto-Optimal Solutions for DWT

	II_{L1}	II_{L2}	II_{L3}	II_{L4}	LUT	FF	DSP	Cycles
1	1	1	1	1	3379	2236	28	590337
2	1	1	1	2	3261	2364	28	720897
3	1	1	2	1	3260	2363	28	720897
4	1	1	2	2	3003	1903	17	851457
5	1	2	1	1	3325	2239	28	720897
6	1	2	1	2	3207	2367	28	851457
7	1	2	2	1	3206	2366	28	851457
8	1	2	2	2	2885	1939	17	982017
9	2	1	1	1	3324	2238	28	720897
10	2	1	1	2	3206	2366	28	851457
11	2	1	2	1	3222	2365	28	851457
12	2	1	2	2	2901	1938	17	982017
13	2	2	1	1	3270	2241	28	851457
14	2	2	1	2	3088	2369	28	982017
15	2	2	2	1	3104	2368	28	982017
16	2	2	2	2	2066	1496	14	1112577



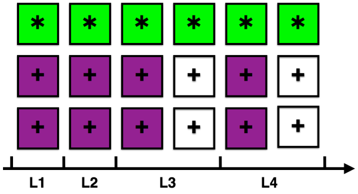
Implementation 1: Fully Pipelined

	L1	L2	L3	L4
Operations	$2\oplus, 1\otimes$	$2\oplus, 1\otimes$	$2\oplus, 2\otimes$	$2\oplus, 2\otimes$
Loop II	1	1	1	1
Operators	$2+ , 1\star$	$2+ , 1\star$	$2+ , 2\star$	$2+ , 2\star$
Cycles	137728	137728	140800	140800



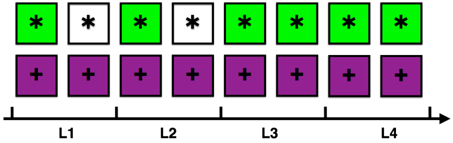
Implementation 4: Non-Fully Pipelined

	L1	L2	L3	L4
Operations	$2\oplus, 1\otimes$	$2\oplus, 1\otimes$	$2\oplus, 2\otimes$	$2\oplus, 2\otimes$
Loop II	1	1	2	2
Operators	$2+$, $1\star$	$2+$, $1\star$	$1+$, $1\star$	$1+$, $1\star$
Cycles	137728	137728	271360	271360



Implementation 16: Non-Fully Pipelined

	L1	L2	L3	L4
Operations	$2\oplus, 1\otimes$	$2\oplus, 1\otimes$	$2\oplus, 2\otimes$	$2\oplus, 2\otimes$
Loop II	2	2	2	2
Operators	$1+, 1\star$	$1+, 1\star$	$1+, 1\star$	$1+, 1\star$
Cycles	268288	268288	271360	271360



Experimental Result for DWT

- Experiment Setup
  - Image size: 512$\star$512
  - EDA tool: Xilinx Vivado Design Suite 2013.4
  - Target Device: Xilinx Virtex-7 XC7V585T
    - DSP: 1260; LUT: 728400; FF: 364200
  - Area and critical path data reported by Vivado after P&R
  - Cycle data reported by Vivado HLS RTL-level simulation

No.	LUT	FF	DSP	Cycles	CP(ns)	#(PE)	Perf. /PE	Total Perf.
1	2855	1985	28	590337	8.800	36	1	36
4	2716	1830	17	851457	8.821	59	0.71	41.9
16	2641	1624	14	1112577	8.909	72	0.53	38.6

Parameters to Influence Optimal Solutions

Platform

- Total available resources in the platform

Components

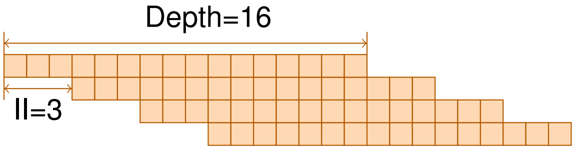
- The area of each shareable components

Loops

- Loop trip counts
- Minimum initiation interval
- Computing loads of each shareable component
- Loop dependence graph

Performance Model

- Single loop



$$Cycle = II * (TripCount - 1) + Depth$$

- Multiple loops

$$\begin{tikzcd} & L_2 \arrow[r] & L_4 \arrow[dr] & \\ L_1 \arrow[ur] \arrow[dr] & & & L_5 \\ & L_3 \arrow[urr] & & \end{tikzcd}$$

- $\forall$Chain $\{L_{t_1}, L_{t_2}, ..., L_{t_L}\} \subseteq \mathcal{P}$, $Cycle \geq \sum_{t=1}^{L} Cycle_{t_k}$

Area Model

- Shareable and non-shareable components

$$Area = \sum_{k=1}^{\#(Loops)} Area_k^{NS} + \sum_{j=1}^{\#(Comp)} (Area_j^S * Alloc_j)$$

- Total number of shareable components:
$\forall$Antichain $\{L_{t_1}, L_{t_2}, ..., L_{t_L}\} \subseteq \mathcal{P}$, $Alloc_j \geq \sum_{t=1}^{L} Alloc_{j,t_k}$

$$\begin{tikzcd}
 & L_2 \arrow[r] & L_4 \arrow[dr] & \\
L_1 \arrow[ur] \arrow[dr] & & & L_5 \\
 & L_3 \arrow[urr] & &
\end{tikzcd}$$

Put it All Together

Minimize

$$\frac{Cycle}{\#(PE)}$$

Subject to

$$\forall 1 \leq k \leq N, Cycle_k = II_k * (TC_k - 1) + Depth_k$$

$$\forall \text{Chain } \{L_{t_1}, L_{t_2}, ..., L_{t_L}\} \subseteq \mathcal{P}, Cycle \geq \sum_{t=1}^{L} Cycle_{t_k}$$

$$\forall \text{Antichain } \{L_{t_1}, L_{t_2}, ..., L_{t_L}\} \subseteq \mathcal{P}, Alloc_j \geq \sum_{t=1}^{L} Alloc_{j,t_k}$$

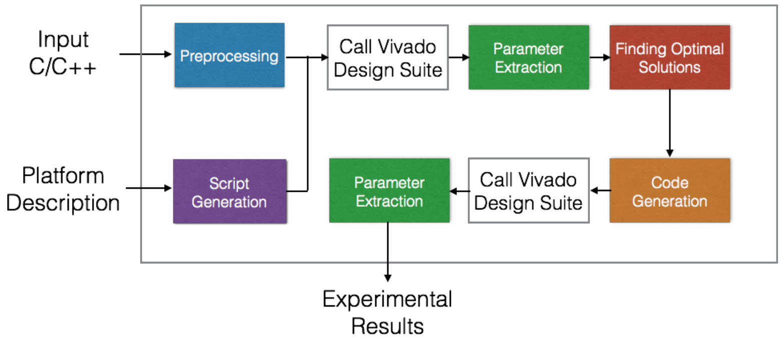
$$Area = \sum_{k=1}^{N} Area_k^{NS} + \sum_{j=1}^{M} (Area_j^{S} * Alloc_j)$$

$$Area * \#(PE) \leq Area_{Available}$$

$$\forall 1 \leq k \leq N, II_k \geq II_k^{Min}$$

$$\forall 1 \leq j \leq M, \forall 1 \leq k \leq N, II_k * Alloc_{j,k} \geq Load_{j,k}$$

Algorithm Implementation



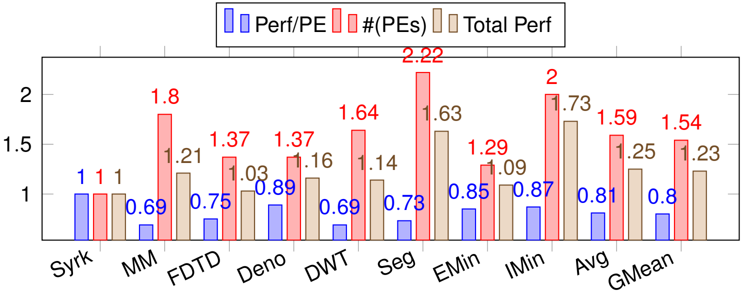
- Source code transformation using ROSE compiler
- Vivado Design Suite 2013.4 is used to perform the HLS, simulation, logic synthesis, place and route

Benchmark Description

Name	Description	Loops	Comp Types	Total Comps	Lines of Code
Syrk	Symmetric Rank-k Operations	2	2	6	8
2MM	Matrix Multiplication	3	2	6	15
FDTD	Finite Diff. Time Domain	4	2	11	16
Deno	Rician Noise Removal	2	6	56	27
DWT	Discrete Wavelet Transform	4	2	10	31
Seg	Image Segmentation	5	6	61	91
EMin	X-Ray Edge Minimization	3	4	37	118
IMin	X-Ray Image Minimization	10	2	47	124

Experimental Results

Bmk .	LOC	#(Total)	#(Pareto)	Loop IIs	Runtime
Syrk	8	2	2	$\langle 1, 1 \rangle$	14.5s
2MM	15	2	2	$\langle 1, 1/2, 1, 1 \rangle$	16.7s
FDTD	16	6	3	$\langle 1, 1, 1, 1/2 \rangle$	16.4s
Deno	27	153	6	$\langle 4, 3/5 \rangle$	15.5s
DWT	31	16	3	$\langle 1, 1, 1/2, 1/2 \rangle$	12.9s
Seg	91	4620	9	$\langle 1, 2, 1/4, 2/3, 5 \rangle$	103s
EMin	118	16	4	$\langle 5/6, 5, 1 \rangle$	14.0s
IMin	124	10	6	$\langle 3/12, 1, 5, 1, 1, 5, 5, 5, 1, 1 \rangle$	23.7s



Conclusions

- Loop pipelining and massive parallelising are key optimization techniques in HLS
    - Co-optimization will improve the performance
- We developed a unified framework
    - Loop pipelining
    - Module selection and duplication
    - Resource sharing
- 25% throughput improvement over a set of selected benchmarks

Discussion and Future Work

- In some benchmarks, the outer-most loop is dependence-free.
- Another approach is to implement the designs with pipelined outer-most loop.
- The proposed inter-module sharing approach may not be beneficial when the task will be executed repeated in a pipeline fashion (which may not be always possible).
- For the DWT case where piplining outer-most loop is possible, the pipelined approach will outputform the proposed resource-sharing approach.