High Level Synthesis of Complex Applications: An H.264 Video Decoder

Xinheng Liu[1], Yao Chen[2;3], Tan Nguyen[3], Swathi Gurumani[3], **Kyle Rupnow[3]**, Deming Chen[1]

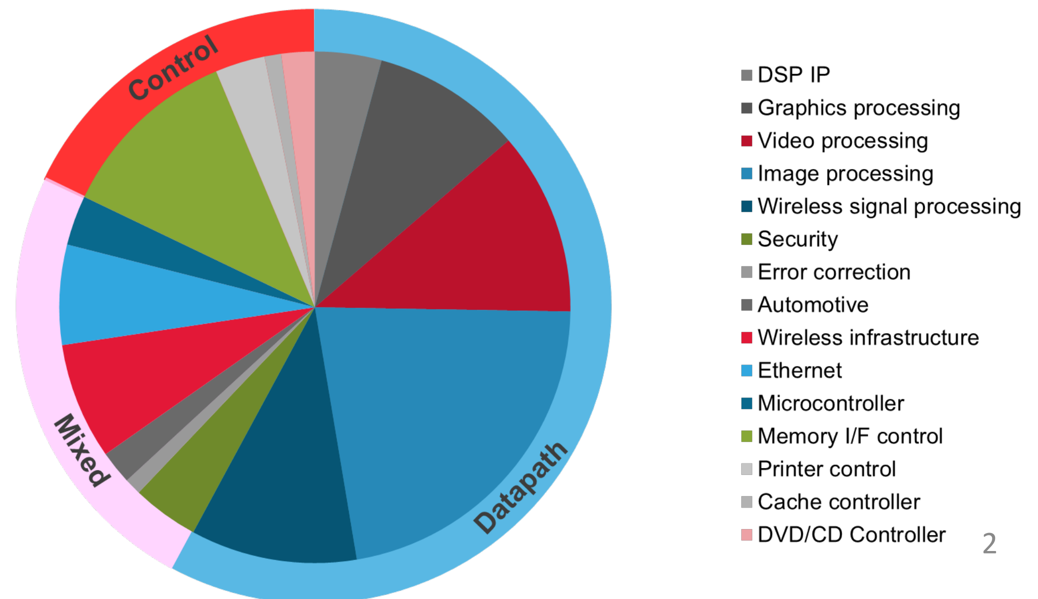
[1] Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, USA

[2] College of Electronic Information and Optical Engineering, Nankai University, China

[3] Advanced Digital Sciences Center, Singapore

High-Level Synthesis Adoption

- HLS tools
  - Significant advancement in design quality
  - Widespread commercial availability
  - Increasingly widely adopted
- Improved design space exploration
- Early debug and verification
- Used for IP design



High-Level Synthesis Evolution

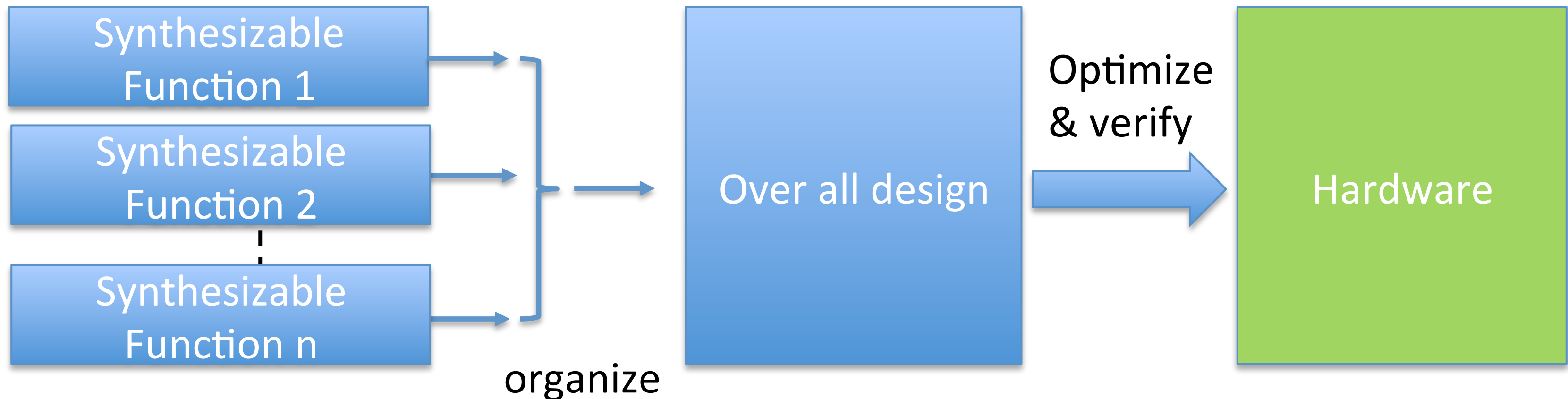
- Increasingly complex applications
  - Existing HLS benchmark size remains small
  - <100 to 1400 lines of codes, < 20 sub-functions
- No sufficient design methodologies
  - Complex designs are treated as simple benchmarks
  - Lack of general study of complexity
  - Many domain or application-specific studies
- Open-source designs remain small and simple
  - Lack of design methodology studies
  - Lack of benchmarks to test HLS complexity handling

Our Contribution

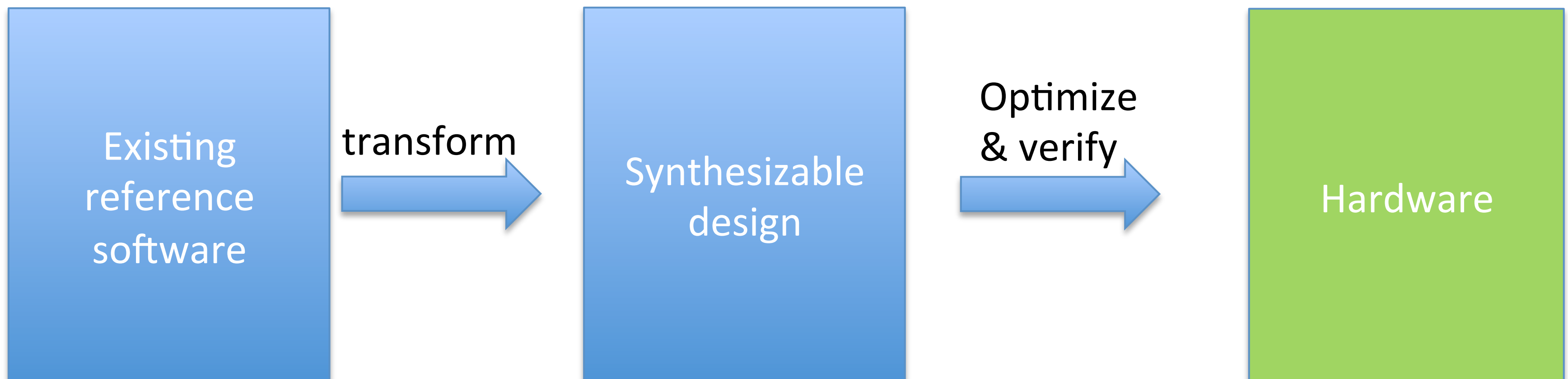
- Complex HLS application case study
  - Complex input code and data structures
  - Optimization strategies for complexity
- Analysis, observations: HLS of complex applications
  - Observations and best-practices
- Open-source benchmark implementation of H.264
  - 34 fps of 640x480 resolution

HLS Design Methodologies

Design from Scratch



Translate from reference



Challenges of Different Design Methods

Design from Scratch

- Start from Specification and Architecture
- Task partitioning
- Manual effort in system integration

Translate from reference

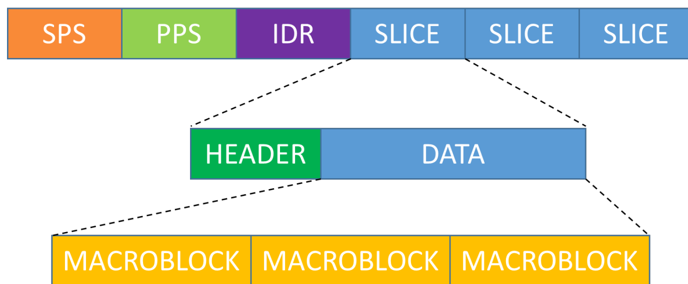
- Lack of goal architecture
- Software organization != hardware organization
- Software redundancy and code reuse
- Efficient, well organized software != good hardware
  - Variable loop bounds
  - Function call/return
  - Functions refactored for reuse
- Merged computation and communication

Why an H.264 Decoder?

- Wide adoption – mobile and high performance app
- Complex input data structure – NAL
- Complex function call – more than 100 functions
- Many function hierarchies – as shown in next page
- Large input code – more than 6000 lines of code
- Demand of higher performance – at least 30 fps
- Constraints of the existing resources – hardware
- Long development cycles – more than half a year

Why H.264 Decoder?

H.264 Insights



Input file structure

-- complex input data structure

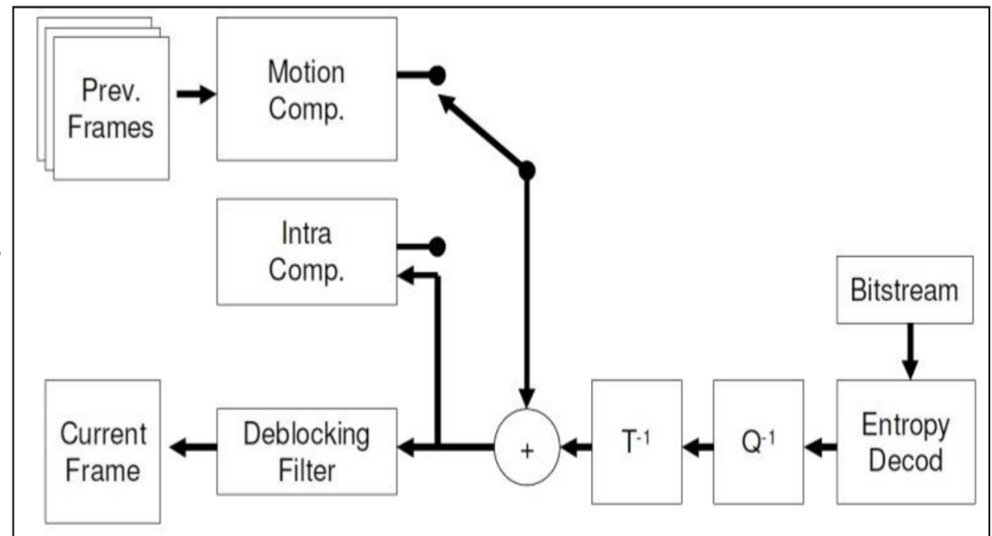
-- quality requirement

H.264 processing

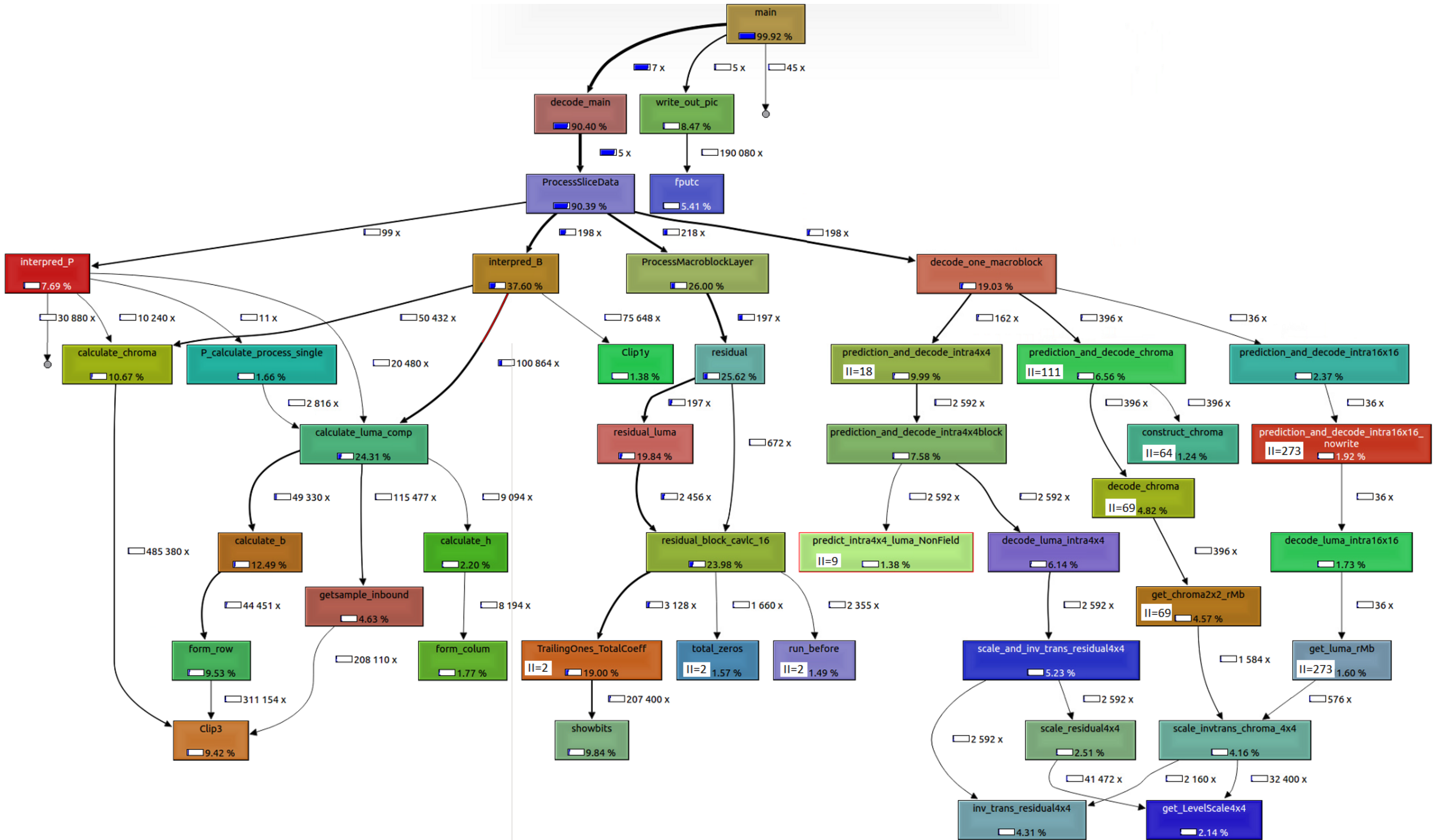
-- Complex internal task functions

-- complex function call hierarchy

-- Demand of high performance



Why H.264 Decoder?



Pre HLS Optimization Design

- Reference code selection – JM H.264 Codec
  - Open-source
  - Ability to partition
  - Generates conformant videos for verification

- Algorithmic code conversion
  - Reference software partitioning
  - Non-synthesizable constructs
  - Preparation for optimization

HLS Based Application Optimization

HLS based optimization goals:

- at least 30 fps for 640x480 resolution videos
- fit into our target platform
  - -- LUT/memory/bandwidth

Optimizations:

- Single function optimization
- Cross-function analysis and optimization
- System Parallelism

HLS Based Application Optimization

Single function optimization – function specialization

```
//Original Clip3() function
int Clip3(int y, int z, int x) {
  return (x<y) ? y : ((x>z) ? z : x); }
```

```
//Clip a number within range (0, 255)
//Clip3(0, 255, x)
int Clip1_1(int x) {
  return (x & 0x80000000) ?
    0 : ((x & 0x7FFFFF00) ? 0xFF : x); }
```

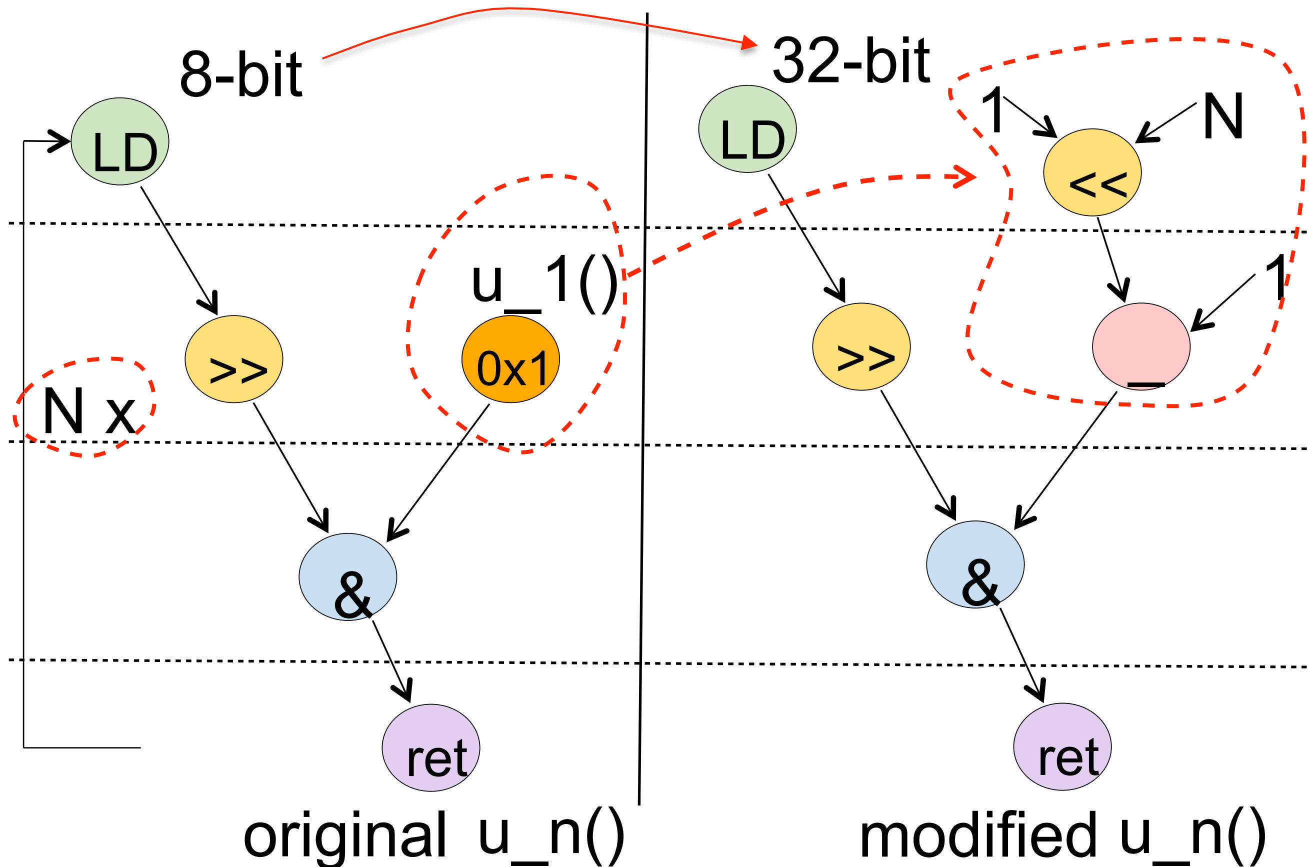
```
// Use case of Clip1_1()
Sluma[startx+i][starty+j] =
    Clip1_1(predL4x4[i][j]+rMbL4x4[i][j]);
```

Single function optimization – pipeline & unroll

- Function rewrite
- pragma insertion

HLS Based Application Optimization

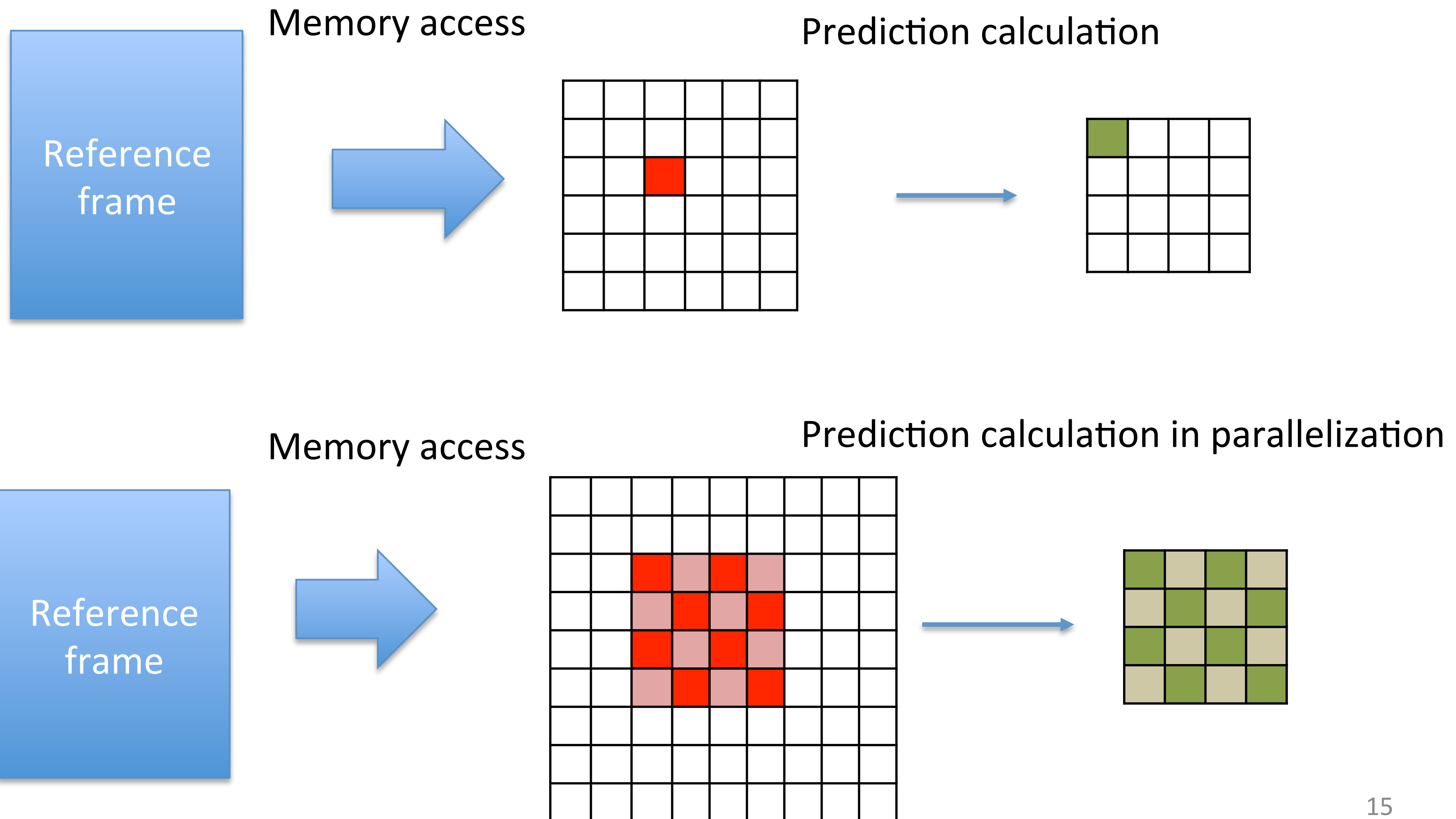
Single function optimization – function call efficiency



HLS Based Application Optimization

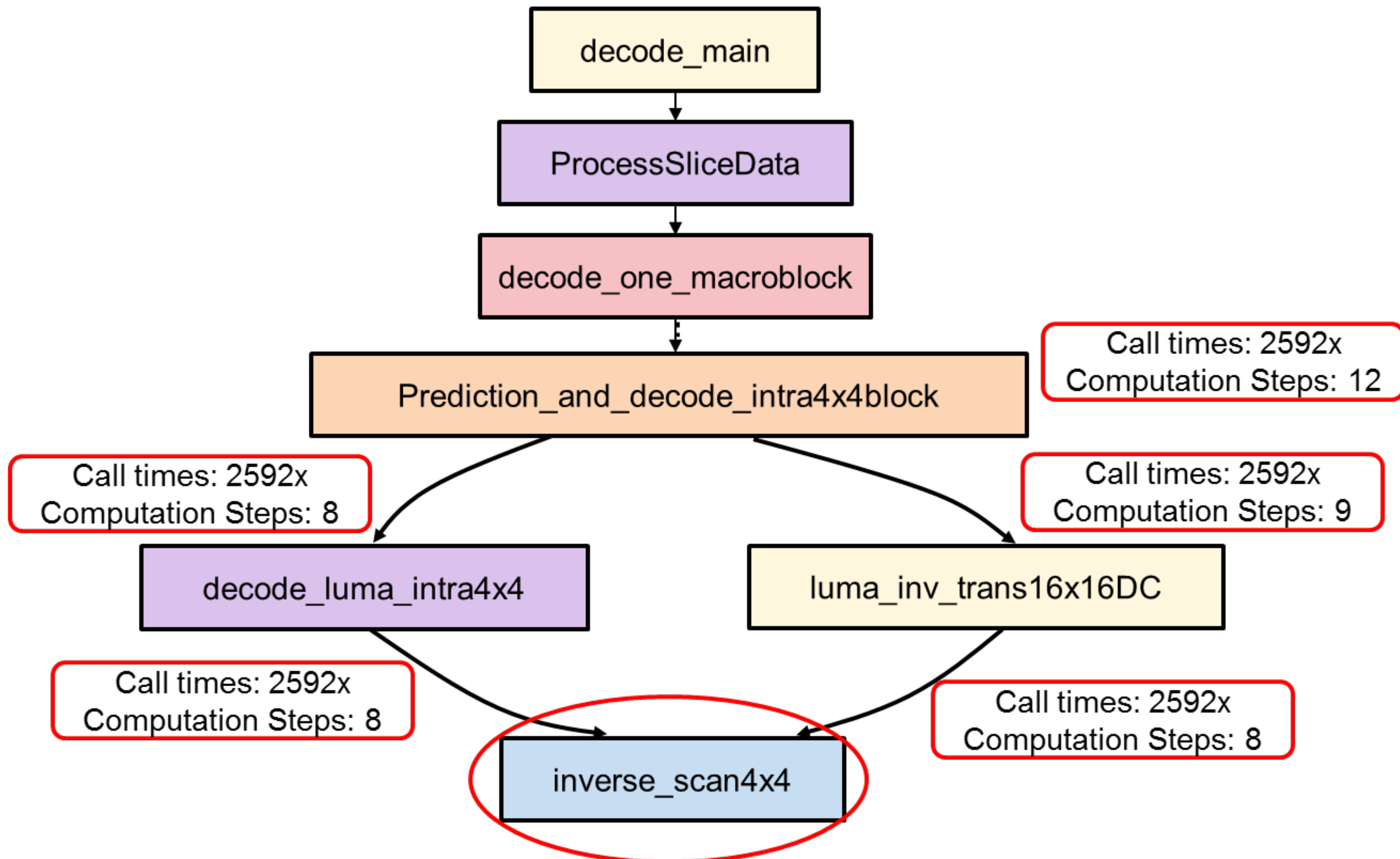
Single function optimization – memory access reuse

Inter-prediction function as an example



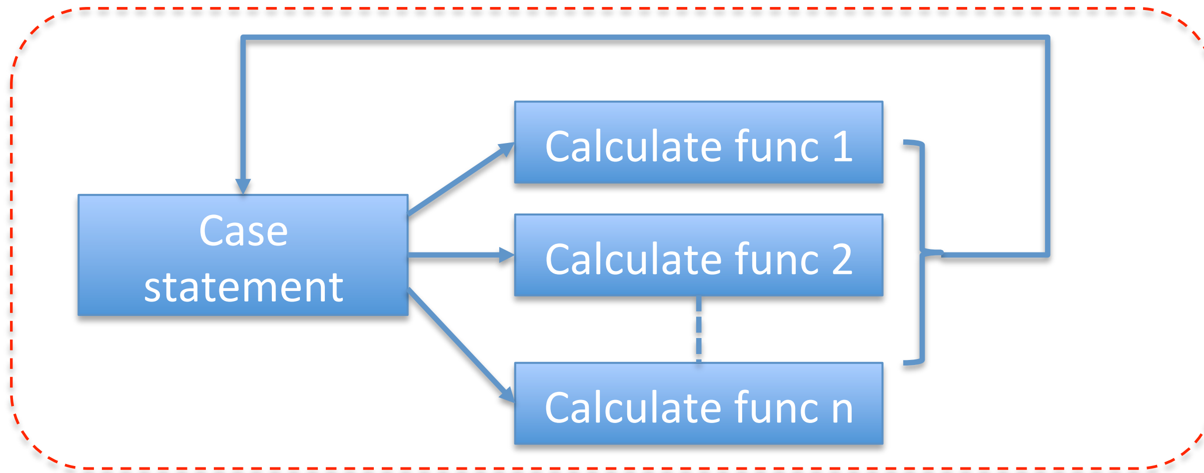
HLS Based Application Optimization

Cross function analysis-- leaf function cost most time

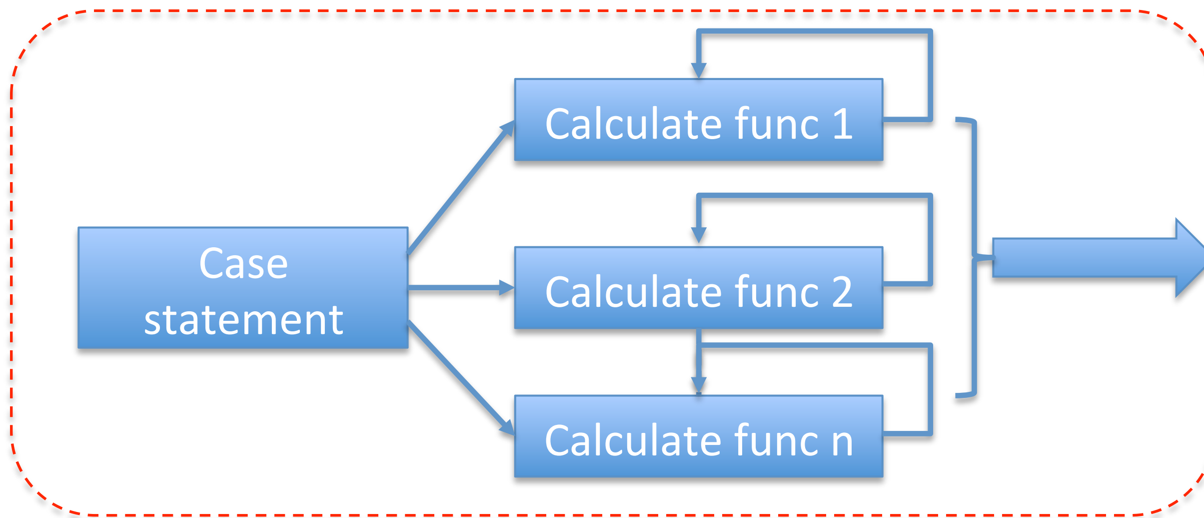


HLS Based Application Optimization

Cross function optimization –loop reorder



Before reorder



After reorder

HLS Based Application Optimization

Cross function optimization – function inlining

Tradeoff:

lowered function call

Increased area

decode_main

ProcessSliceData

decode_one_macroblock

Prediction_and_decode_intra4x4block

decode_luma_intra4x4

luma_inv_trans16x16DC

inverse_scan4x4

inverse_scan4x4

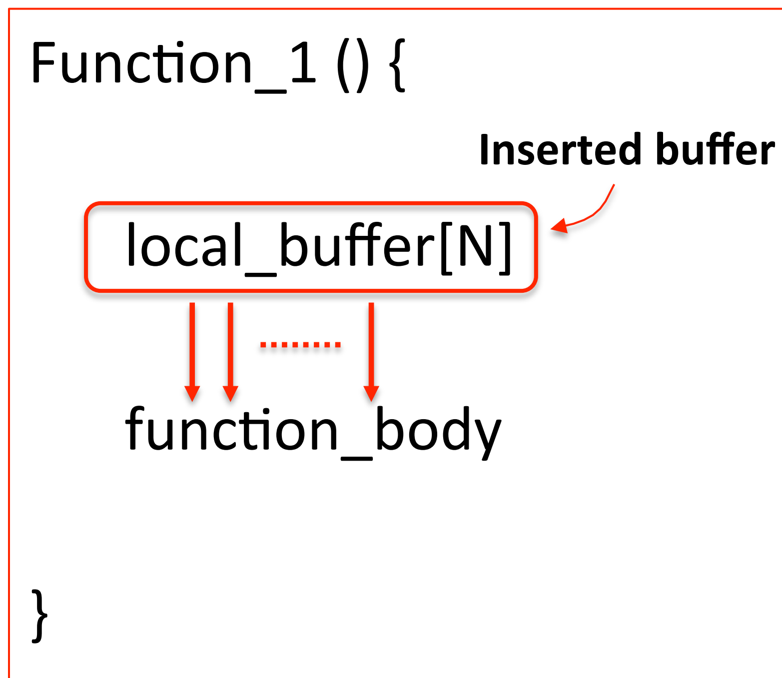
inverse_scan4x4

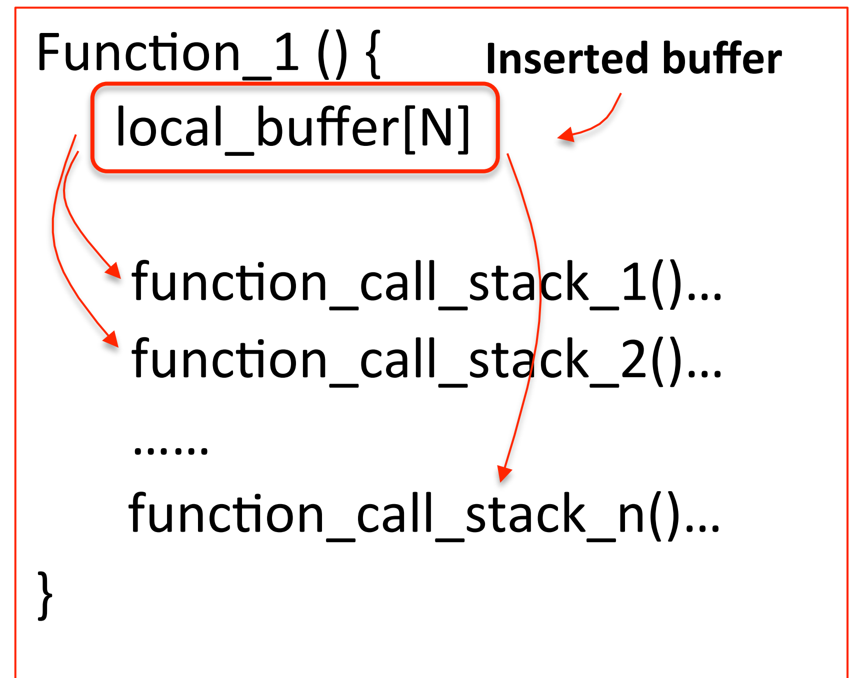
HLS Based Application Optimization

Cross function optimization – buffer optimization

Local buffer insertion – optimize leaf function performance

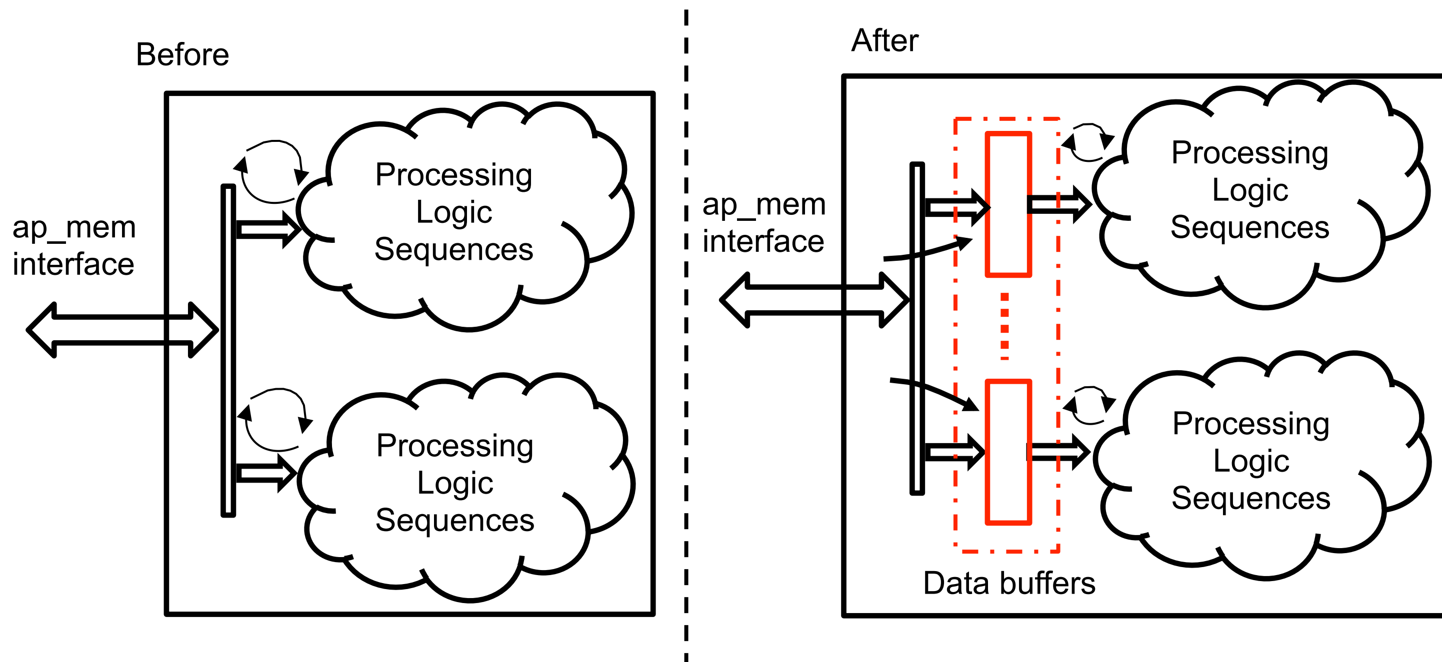
Call stack buffer insertion – improve call stack parallelization





HLS Based Application Optimization

System parallelism optimization – buffer optimization



System parallelism – task parallelism

- Buffer duplication – multiple functions access
- Interface duplication – data input partition

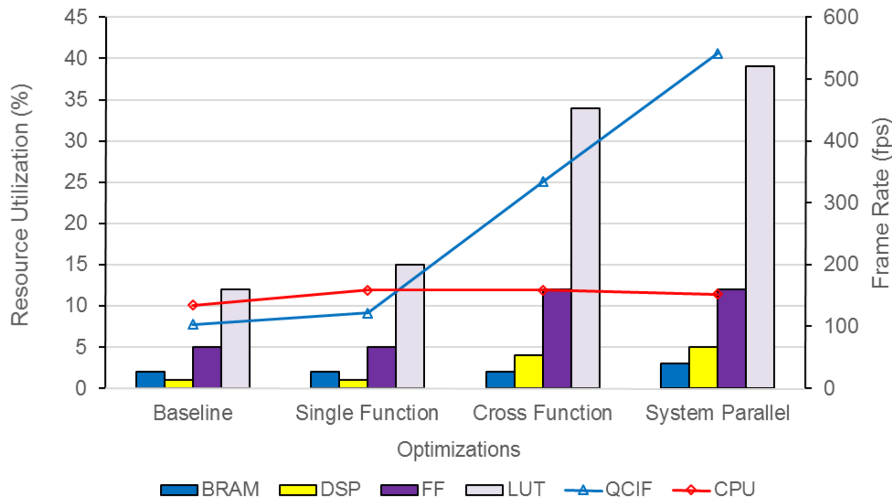
Experiment

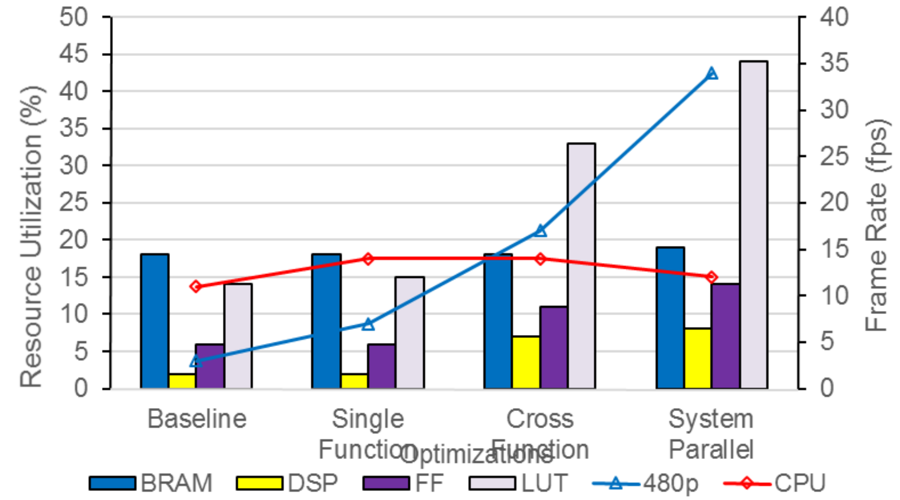
Experiment setup:
Hardware platform – Xilinx VC709 Evaluation Platform
Synthesize and P&R server – Intel Xeon CPU E5-2630 v3
Software version – Vivado_HLS and Vivado 2014.4
On board verification – Omnitek Zynq 7045 board

Three kinds of evaluations:

- Resource usage
- Performance
- Optimization difference

Performance of Low & High Resolution

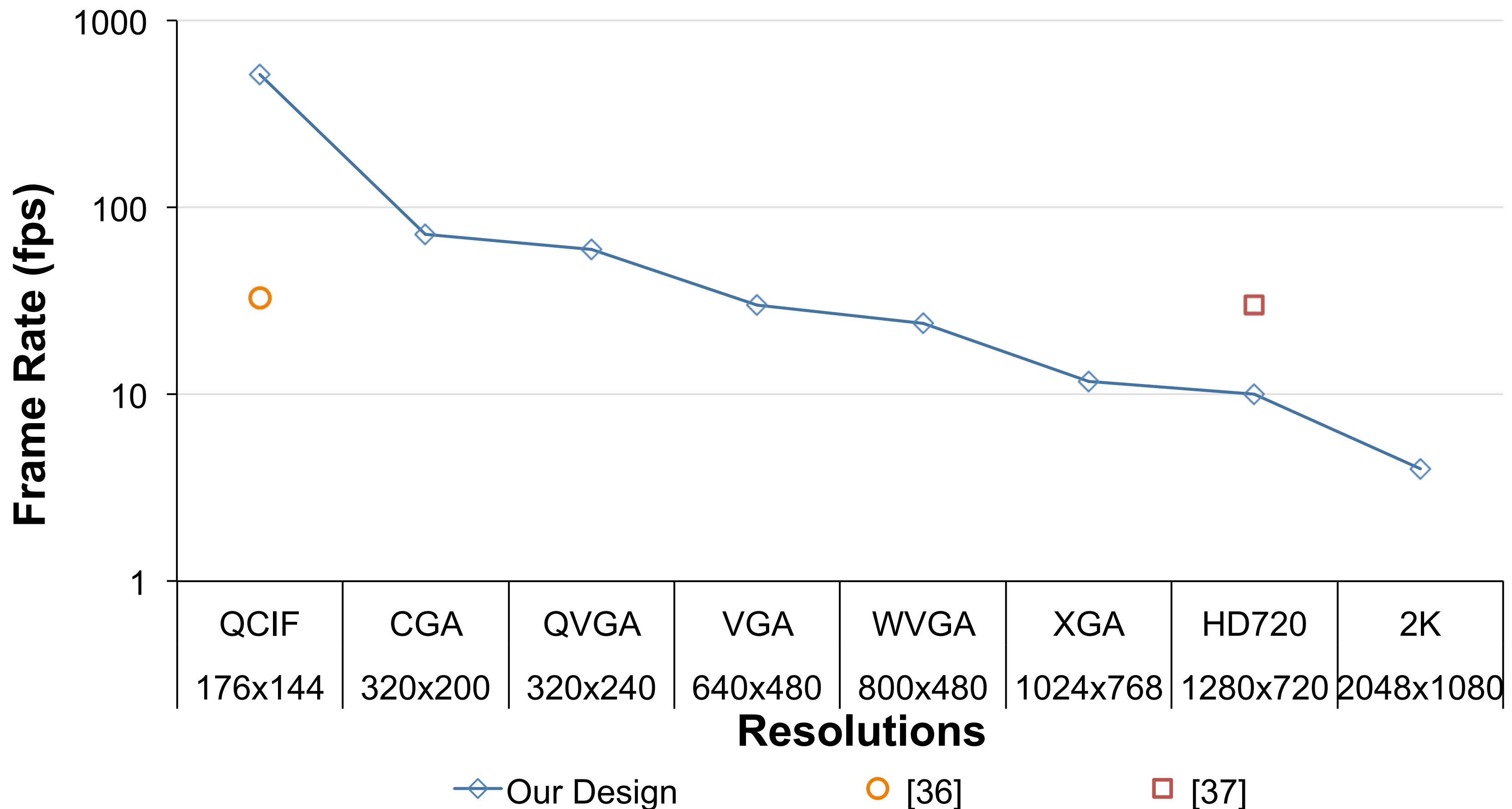




- Both CPU and HLS design have better performance on single function step
- HLS design have better performance on the following step while CPU design have worse performance
- The usage of DSP, FF and LUT present an increasing trend while BRAM remain basically same.

Comparison with Existing Designs

Average time consuming in units of macroblock



Observations & Proposed HLS extension

Best practice observations:

- Single, critical function optimizations remind important but not effective enough
- Cross-function optimizations provide the majority of the gains
- Buffer insertion creates optimization opportunities

Proposed HLS extension:

- Integrated Call-graph and function call frequency profiling
- Illustrate OpenMP-style pragmas to define local structures
- Improve loop of function call optimization

Conclusion

- Top-down HLS of a complex application
- C to HW synthesis and reach 34 fps at 480p resolution
- Design method of best practices for HLS optimizations of complex applications
- Future tool improvements for supporting complex applications
- Open source of pragma-inserted C code of the H.264 decoder as a complex HLS benchmark is available for use under the MIT license at http://dchen.ece.illinois.edu/tools.html.

Open-source Updates

Release updates 01/2016:

- Better parameterization of the input resolution
- Further improved performance
- On board verification demo
- Full simulation/synthesis/implementation scripts
- Less pragmas illustrated (generic for other HLS tools)