Software optimization for the multi-core architecture with State of the art Design Flow

July 11th 2016

RICOH COMPANY, LTD.
Institute of ICT

Sadahiro Kimura

Agenda

- Motivation
- Traditional Design Flow Issue
- Our Vision
- Outline
- State-of-the-art Design Flow
- Case Study - First Target
- Analyze and Profile
- Parallelize
- Architecture Mapping
- Result
- Conclusion
- Next Step

Motivation

- **Keywords of the issues**
  - Commoditization of hardware platform
    - It is most important to make an excellent feature of the product
  - Difficult software development of the multi-core platform
    - Optimization and debugging is very difficult
  - The time shortening of the development
    - Time to Market is very important
  - Quality of the development
    - We have limitations of manual partitioning and mapping

- **We need to make the new design flow to solve them !**

- **We need to catch up state-of-the-art design flow !**

Traditional Design Flow Issues

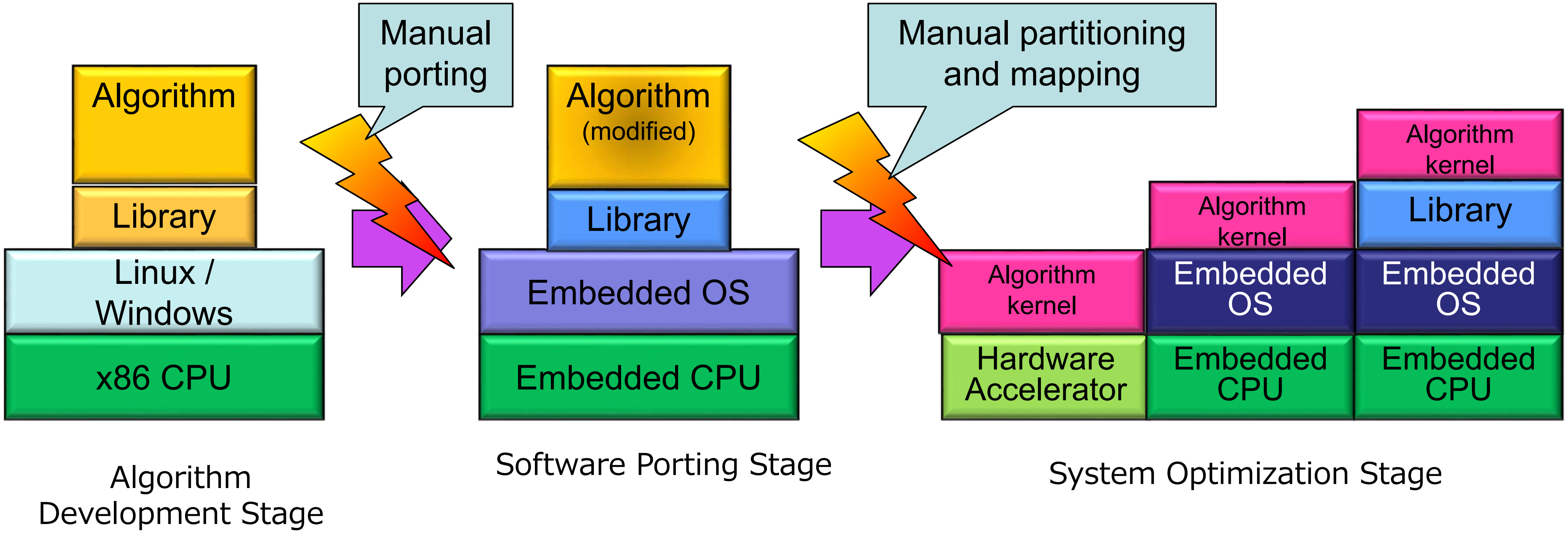
RICOH imagine. change.

Algorithm Development issues

- Algorithm design is separated from the system design
- Algorithm engineers do not consider system implementation

Implementation and Optimization issues

- Manual porting of algorithms to target operating system
- Manual partitioning and mapping



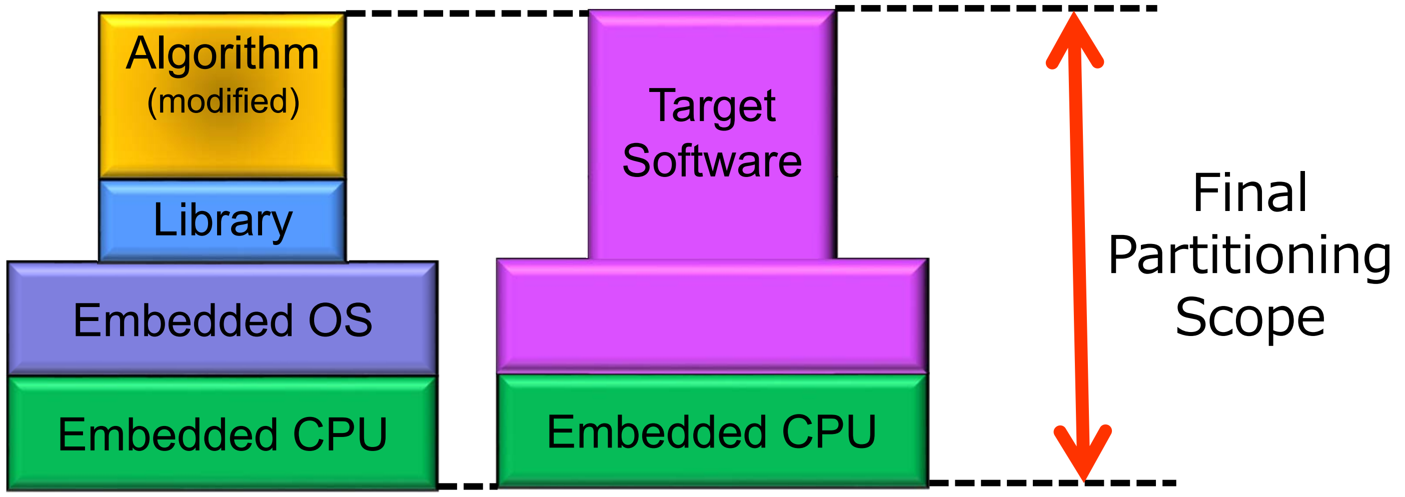
Our Vision

Our Vision

- To make the seamless design flow from Algorithm design to system implementation for multi-core architecture

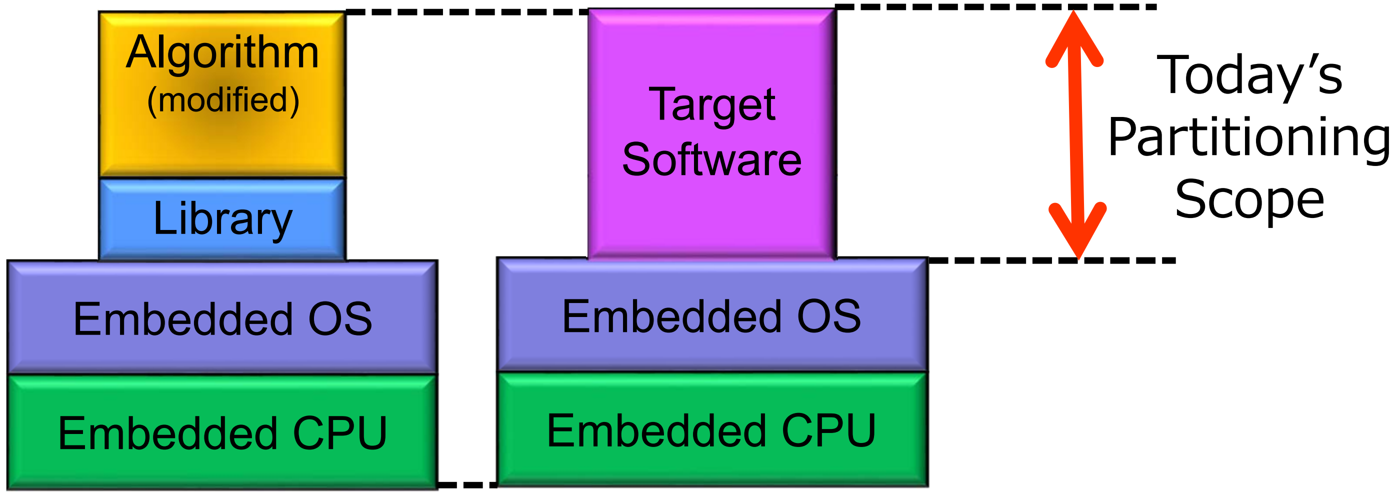
Our Scope

- Our scope is the entire hardware and software
  - Now I'm trying to make software partitioning design flow for multi-core
  - It is important to divide the whole of software for multi-core
    - It should be including Operating System portion – This is the next step !



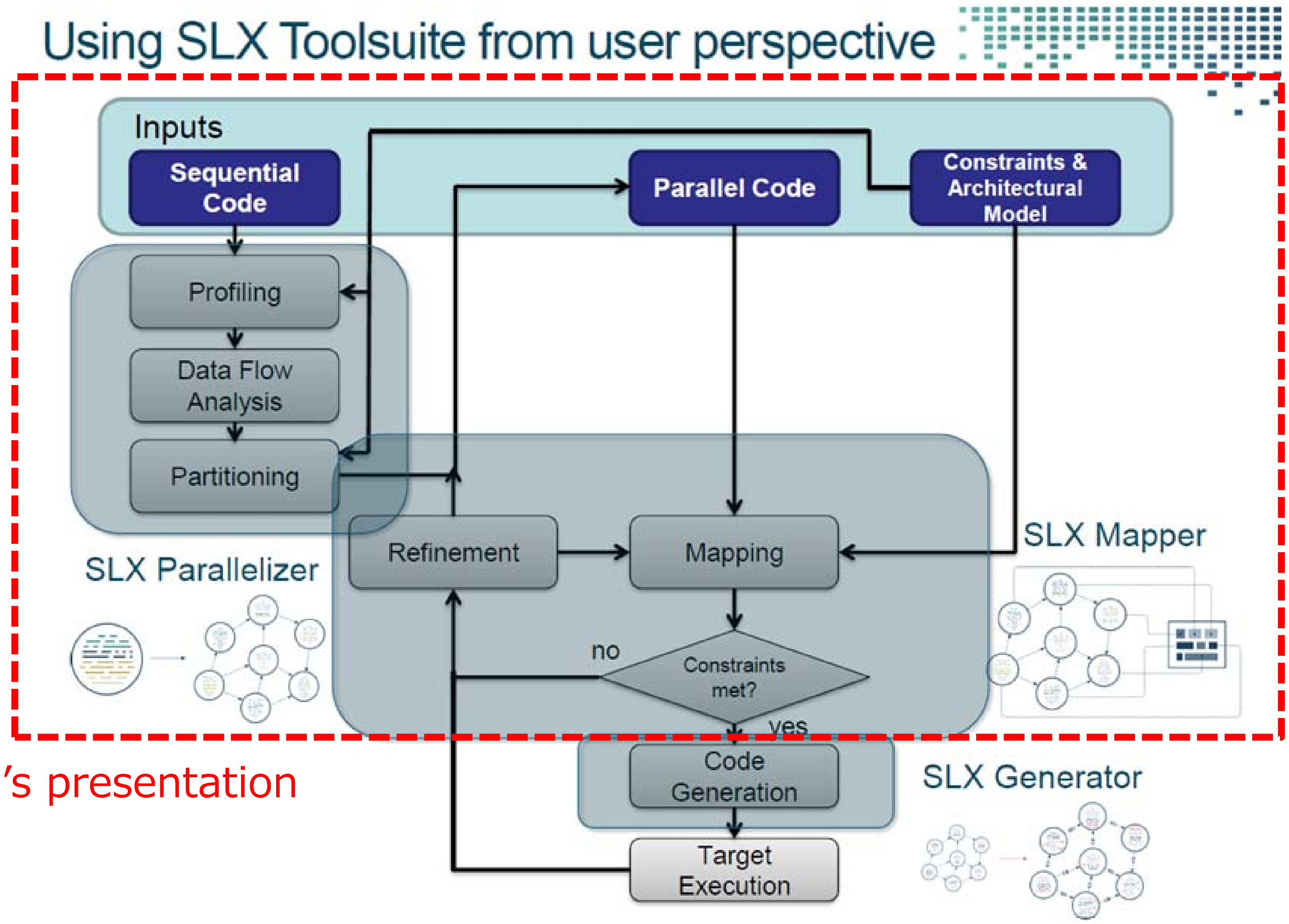
Outline

- We tried to optimize the only software part at the first step
- We tried to evaluate SILEXICA to optimize software part
- We tried to use the JPEG Encoder as the first case study
- We tried to divide JPEG Encoder into some processes
- We got a result that was <u>not expected</u>
- We were inspired by this result
- And, we are considering the next step



State-of-the-Art Design Flow

- **SILEXICA Design Flow**



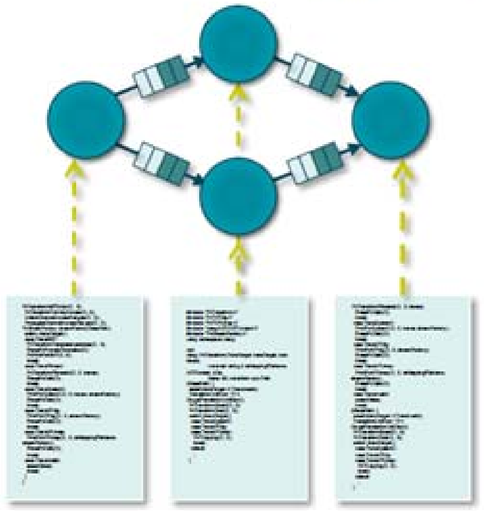
Today's presentation

State-of-the-Art Design Flow

CPN

Parallel Specification: CPN

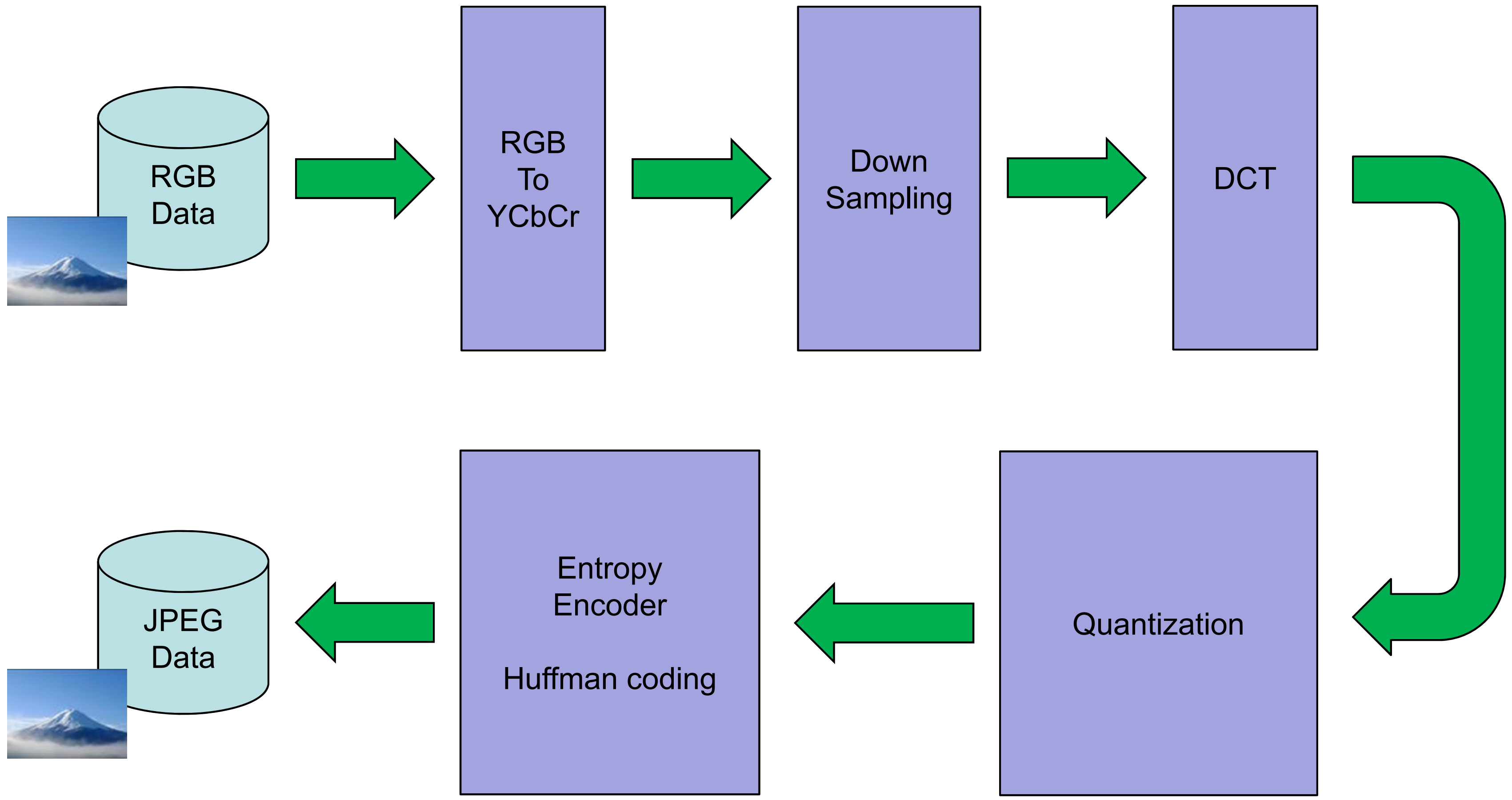
- CPN: C for Process Networks
  - *Intuitive parallel programming*
- Based on C
  - Processes in C
  - Channels support C types, structs, typedefs, …



```
__PNkpn Amp __PNin(short A[2]) __PNout(short B[2]) __PNparam(int boost)
{
  while (1)
    __PNin(A) __PNout(B) {
      for (int i = 0; i < 2; i++)
        B[i] = A[i]*boost;
    }
}

__PNprocess AudioAmp1 = Amp __PNin(C) __PNout(F) __PNparam(3);
__PNprocess AudioAmp2 = Amp __PNin(D) __PNout(G) __PNparam(10);
```

Case Study

Generic JPEG Encoder algorithm



Case Study

JPEG Encoder condition

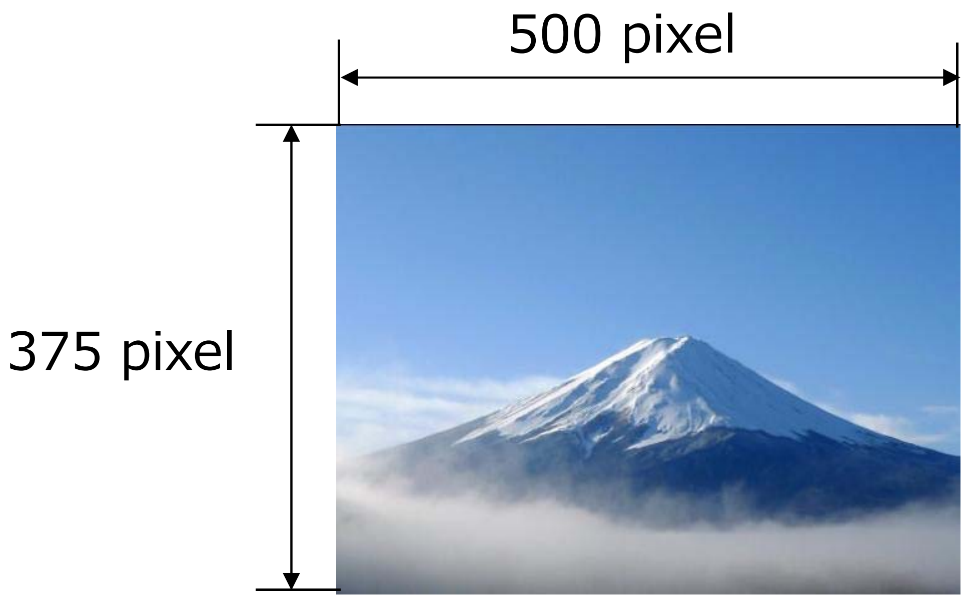
Condition	Value
Number of line at C source code	1121 lines (without comment line)
Number of File	6 files (including header file)
Size of the executable file (Binary size)	494.5K Bytes (Result of SLX compiler)

Input picture condition

- 500pixel x 375pixcel
- RGB format image data

SILEXICA version

- SLX Tool Suite 2016.1
- 64bit Linux CentOS7

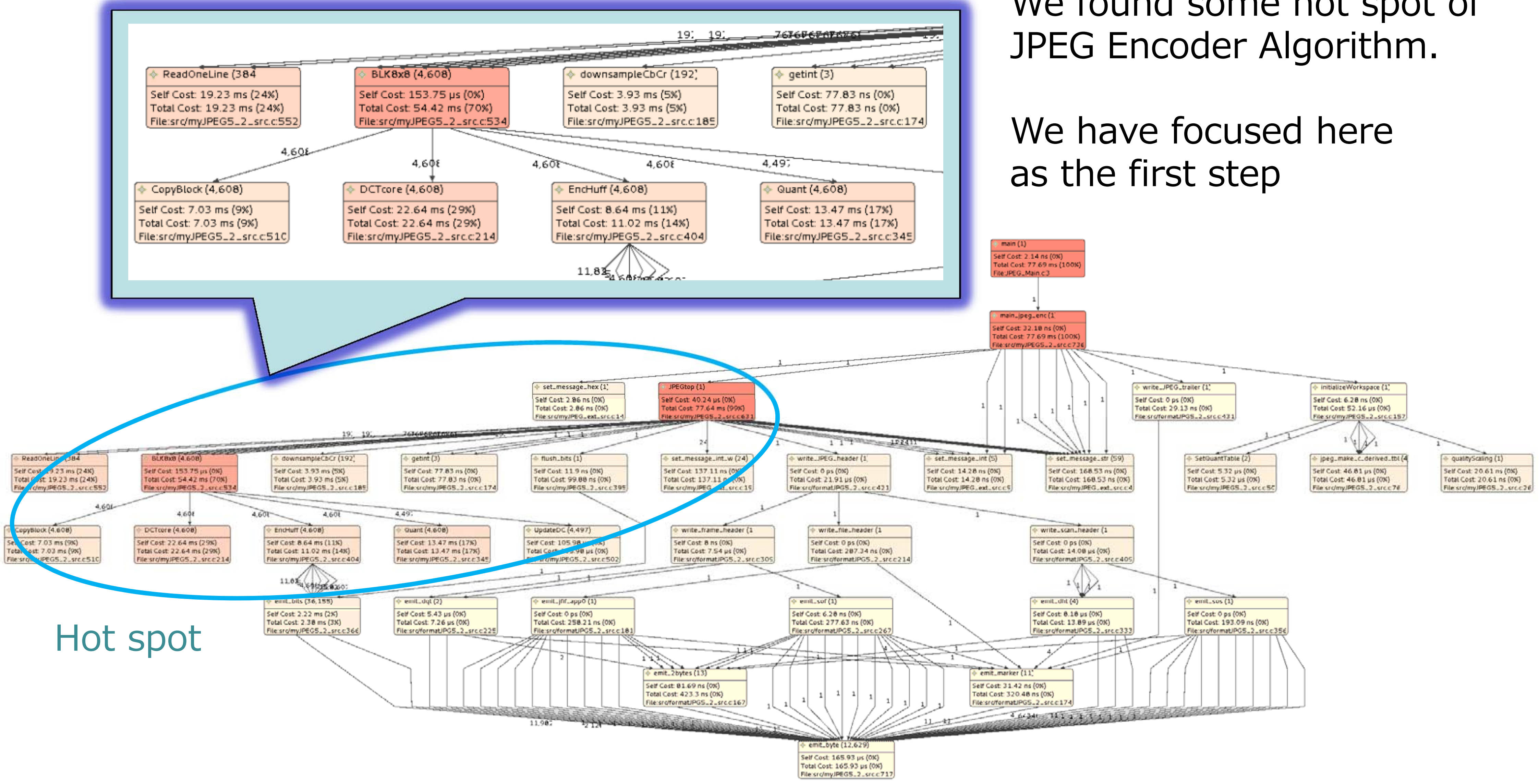


Analyze and Profile

Call Graph of JPEG Encoder

We found some hot spot of JPEG Encoder Algorithm.

We have focused here as the first step



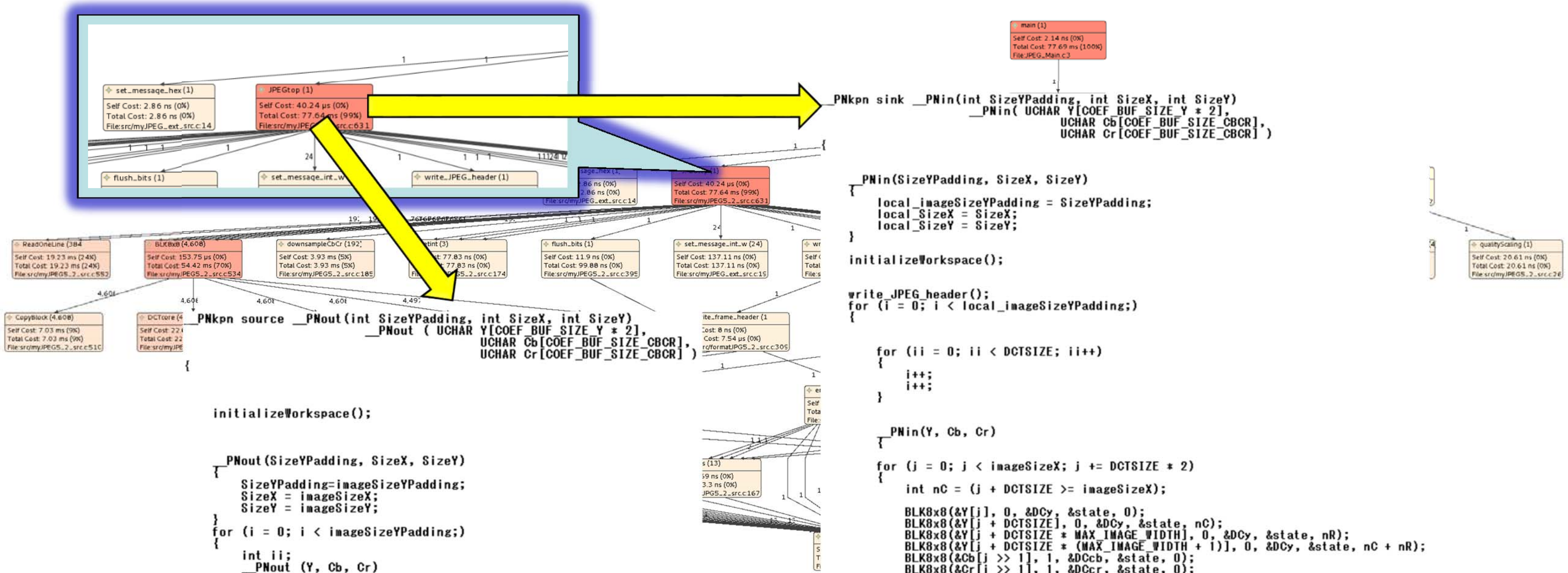
ReadOneLine (384
Self Cost: 19.23 ms (24%)
Total Cost: 19.23 ms (24%)
File:src/myJPEG5_2_src.c:552
BLK8x8 (4,608)
Self Cost: 153.75 μs (0%)
Total Cost: 54.42 ms (70%)
File:src/myJPEG5_2_src.c:534
downsampleCbCr (192)
Self Cost: 3.93 ms (5%)
Total Cost: 3.93 ms (5%)
File:src/myJPEG5_2_src.c:185
getint (3)
Self Cost: 77.83 ns (0%)
Total Cost: 77.83 ns (0%)
File:src/myJPEG5_2_src.c:174
CopyBlock (4,608)
Self Cost: 7.03 ms (9%)
Total Cost: 7.03 ms (9%)
File:src/myJPEG5_2_src.c:510
DCTcore (4,608)
Self Cost: 22.64 ms (29%)
Total Cost: 22.64 ms (29%)
File:src/myJPEG5_2_src.c:214
EncHuff (4,608)
Self Cost: 8.64 ms (11%)
Total Cost: 11.02 ms (14%)
File:src/myJPEG5_2_src.c:404
Quant (4,608)
Self Cost: 13.47 ms (17%)
Total Cost: 13.47 ms (17%)
File:src/myJPEG5_2_src.c:345
main (1)
main_jpeg_enc (1)
JPEGtop (1)
emit_byte (12,629)

Hot spot

Parallelize

First Trial – CPN code

- Divide JPEG Encoder into 2 processes



main (1)
Self Cost: 2.14 ns (0%)
Total Cost: 77.69 ms (100%)
File:JPEG_Main.c:3
set_message_hex (1)
Self Cost: 2.86 ns (0%)
Total Cost: 2.86 ns (0%)
File:src/myJPEG_ext_src.c:14
JPEGtop (1)
Self Cost: 40.24 µs (0%)
Total Cost: 77.64 ms (99%)
File:src/myJPEG5_2_src.c:631
flush_bits (1)
set_message_int_w
write_JPEG_header (1)
ReadOneLine (384
Self Cost: 19.23 ms (24%)
Total Cost: 19.23 ms (24%)
File:src/myJPEG5_2_src.c:552
BLK8x8 (4,608)
Self Cost: 153.75 µs (0%)
Total Cost: 54.42 ms (70%)
File:src/myJPEG5_2_src.c:534
downsampleCbCr (192)
Self Cost: 3.93 ms (5%)
Total Cost: 3.93 ms (5%)
File:src/myJPEG5_2_src.c:185
Self Cost: 11.9 ns (0%)
Total Cost: 99.88 ns (0%)
File:src/myJPEG5_2_src.c:395
set_message_int_w (24)
Self Cost: 137.11 ns (0%)
Total Cost: 137.11 ns (0%)
File:src/myJPEG_ext_src.c:15
qualityScaling (1)
Self Cost: 20.61 ns (0%)
Total Cost: 20.61 ns (0%)
CopyBlock (4,608)
Self Cost: 7.03 ms (9%)
Total Cost: 7.03 ms (9%)
File:src/myJPEG5_2_src.c:510
DCTcore (4

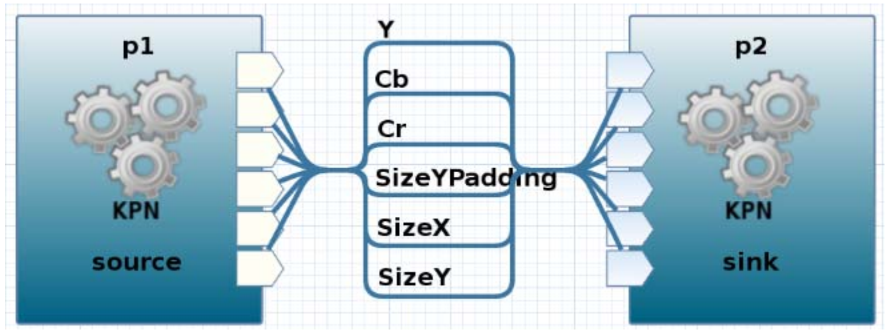
```
__PNkpn source __PNout(int SizeYPadding, int SizeX, int SizeY)
                 __PNout ( UCHAR Y[COEF_BUF_SIZE_Y * 2],
                           UCHAR Cb[COEF_BUF_SIZE_CBCR],
                           UCHAR Cr[COEF_BUF_SIZE_CBCR] )
{


    initializeWorkspace();


    __PNout(SizeYPadding, SizeX, SizeY)
    {
        SizeYPadding=imageSizeYPadding;
        SizeX = imageSizeX;
        SizeY = imageSizeY;
    }
    for (i = 0; i < imageSizeYPadding;)
    {
        int ii;
        __PNout (Y, Cb, Cr)
        {
        ii = 0;
        ReadOneLine(fpImage, i++, &Y[0], 0);
        ReadOneLine(fpImage, i++, &Y[0], 1);
        downsampleCbCr(i, &Cb[0], &Cr[0], 0);
       for (ii = 1; ii < DCTSIZE; ii++)
        {
            ReadOneLine(fpImage, i++, &Y[0], 1);
            ReadOneLine(fpImage, i++, &Y[0], 1);
            downsampleCbCr(i, &Cb[0], &Cr[0], 1);
        }
        }
    }


}
```

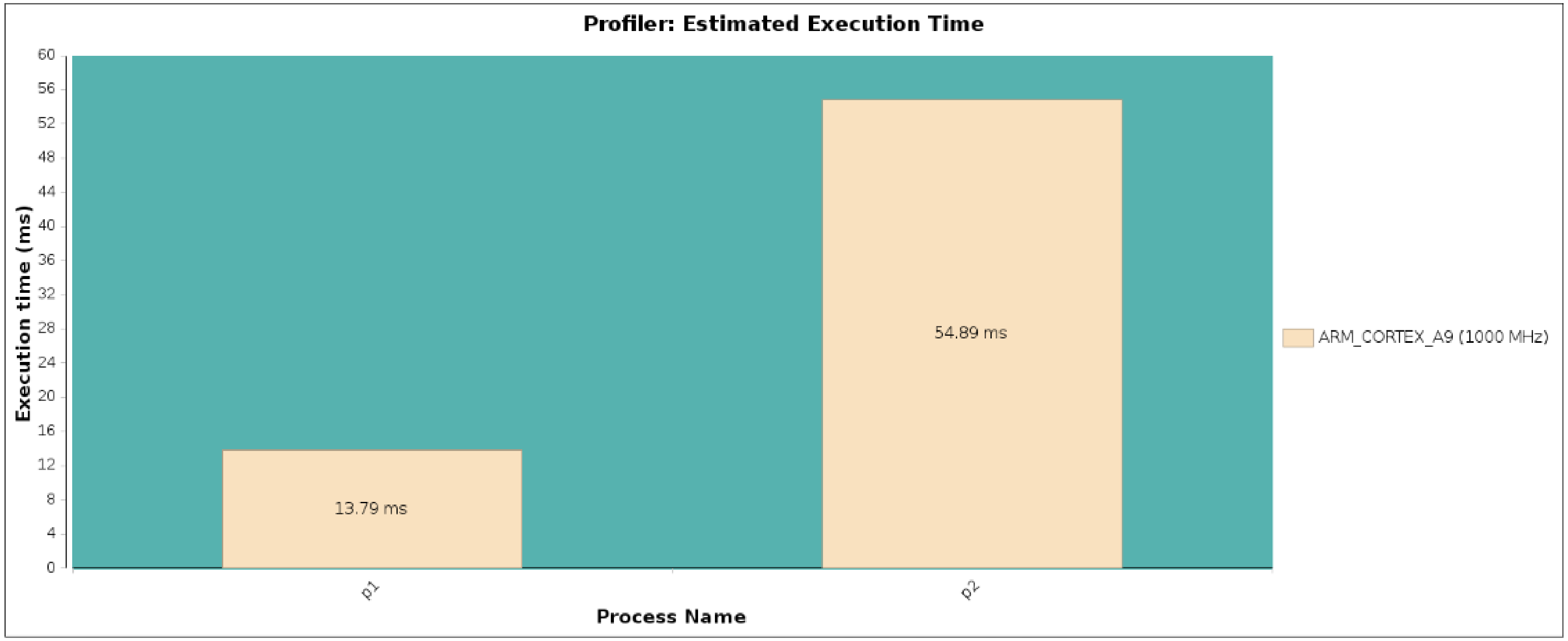
```
__PNkpn sink __PNin(int SizeYPadding, int SizeX, int SizeY)
               __PNin( UCHAR Y[COEF_BUF_SIZE_Y * 2],
                       UCHAR Cb[COEF_BUF_SIZE_CBCR],
                       UCHAR Cr[COEF_BUF_SIZE_CBCR] )
{

    __PNin(SizeYPadding, SizeX, SizeY)
    {
        local_imageSizeYPadding = SizeYPadding;
        local_SizeX = SizeX;
        local_SizeY = SizeY;
    }

    initializeWorkspace();


    write_JPEG_header();
    for (i = 0; i < local_imageSizeYPadding;)
    {


        for (ii = 0; ii < DCTSIZE; ii++)
        {
            i++;
            i++;
        }


        __PNin(Y, Cb, Cr)
        {

        for (j = 0; j < imageSizeX; j += DCTSIZE * 2)
        {
            int nC = (j + DCTSIZE >= imageSizeX);

            BLK8x8(&Y[j], 0, &DCy, &state, 0);
            BLK8x8(&Y[j + DCTSIZE], 0, &DCy, &state, nC);
            BLK8x8(&Y[j + DCTSIZE * MAX_IMAGE_WIDTH], 0, &DCy, &state, nR);
            BLK8x8(&Y[j + DCTSIZE * (MAX_IMAGE_WIDTH + 1)], 0, &DCy, &state, nC + nR);
            BLK8x8(&Cb[j >> 1], 1, &DCcb, &state, 0);
            BLK8x8(&Cr[j >> 1], 1, &DCcr, &state, 0);
        }
        }

    }

    flush_bits(&state);
    write_JPEG_trailer();

    fclose(fpJPG);

}
```

Parallelize

First Trial – Execution time

- Estimate execution time by ARM CA9 architecture model

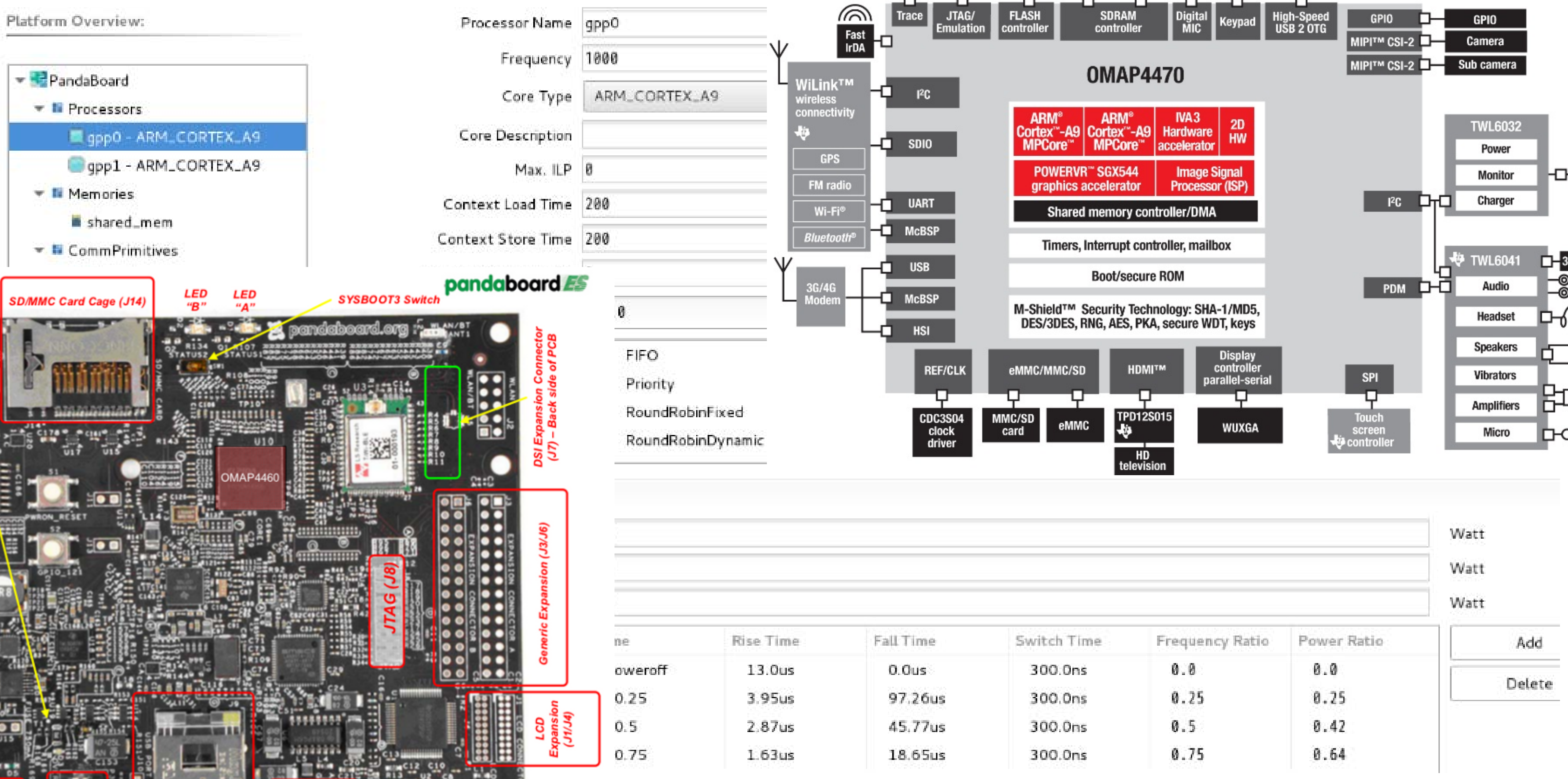




Architecture Mapping

RICOH
imagine. change.

Mapping to Pandaboard



Platform Overview:

- PandaBoard
  - Processors
    - gpp0 - ARM_CORTEX_A9
    - gpp1 - ARM_CORTEX_A9
  - Memories
    - shared_mem
  - CommPrimitives

Processor Name	gpp0
Frequency	1000
Core Type	ARM_CORTEX_A9
Core Description	
Max. ILP	0
Context Load Time	200
Context Store Time	200

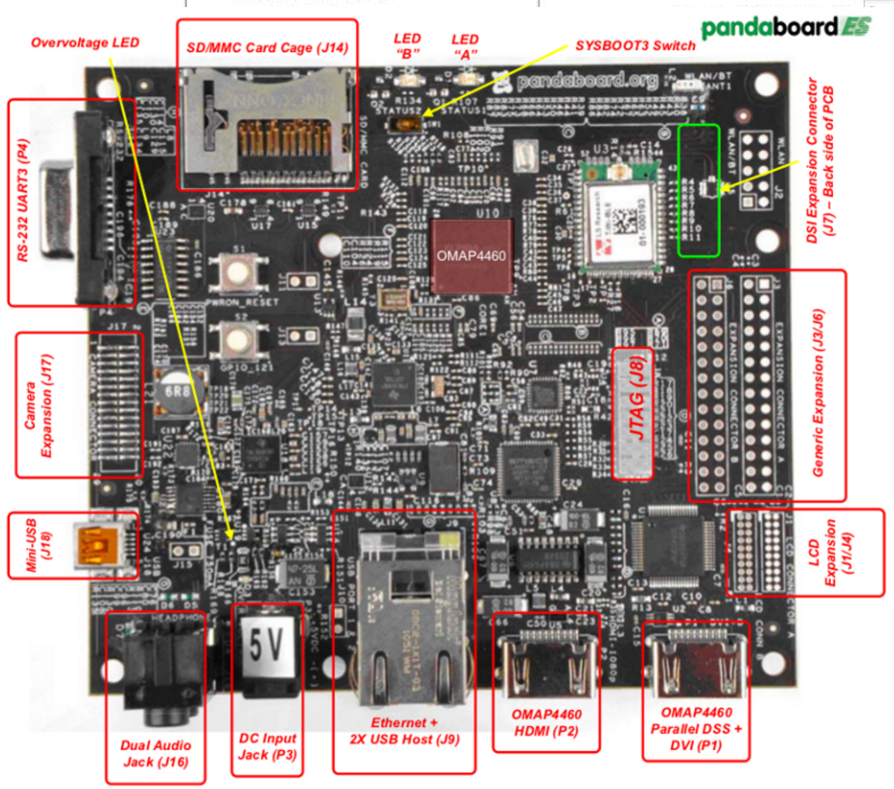
0

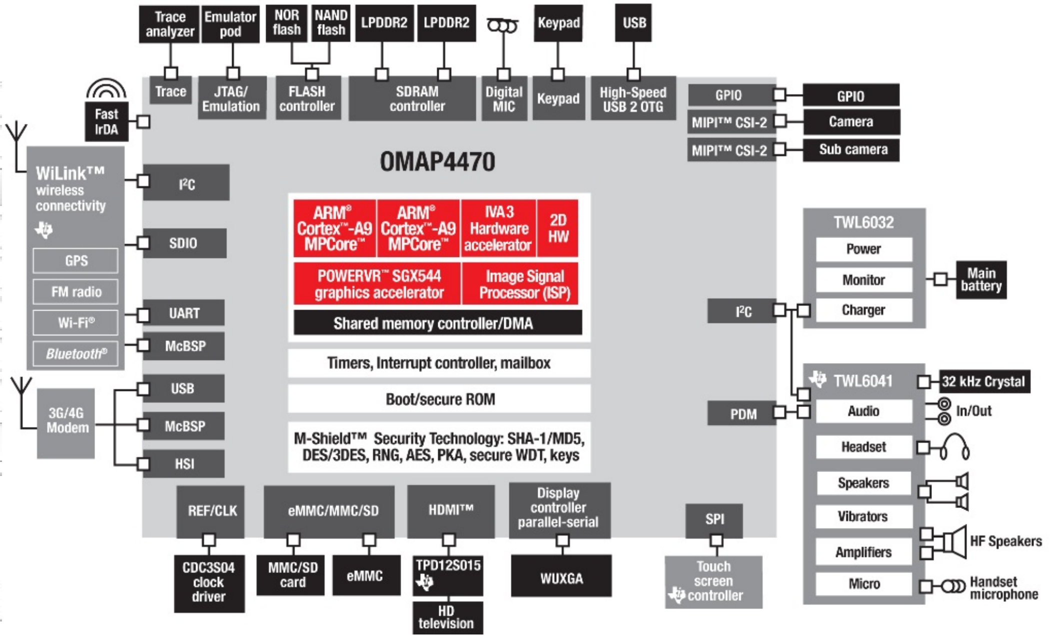
FIFO
Priority
RoundRobinFixed
RoundRobinDynamic

Watt
Watt
Watt

ne	Rise Time	Fall Time	Switch Time	Frequency Ratio	Power Ratio
oweroff	13.0us	0.0us	300.0ns	0.0	0.0
0.25	3.95us	97.26us	300.0ns	0.25	0.25
0.5	2.87us	45.77us	300.0ns	0.5	0.42
0.75	1.63us	18.65us	300.0ns	0.75	0.64

Add
Delete

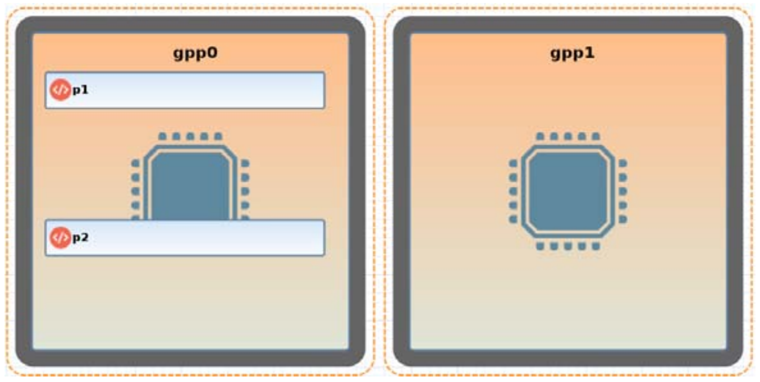




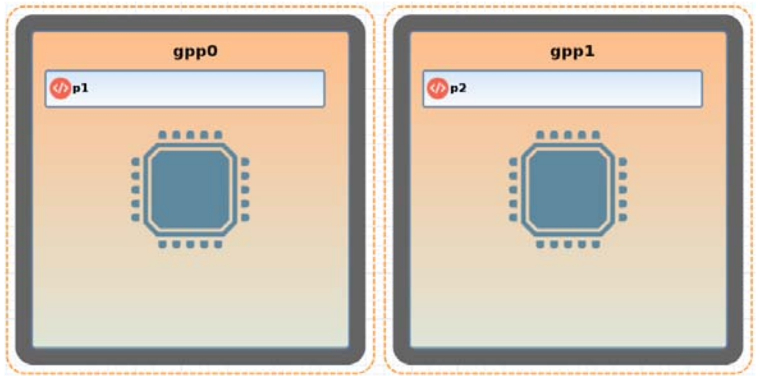
Architecture Mapping

Mapping condition

- Condition A - typical
  - Mapping the two processes into one CPU

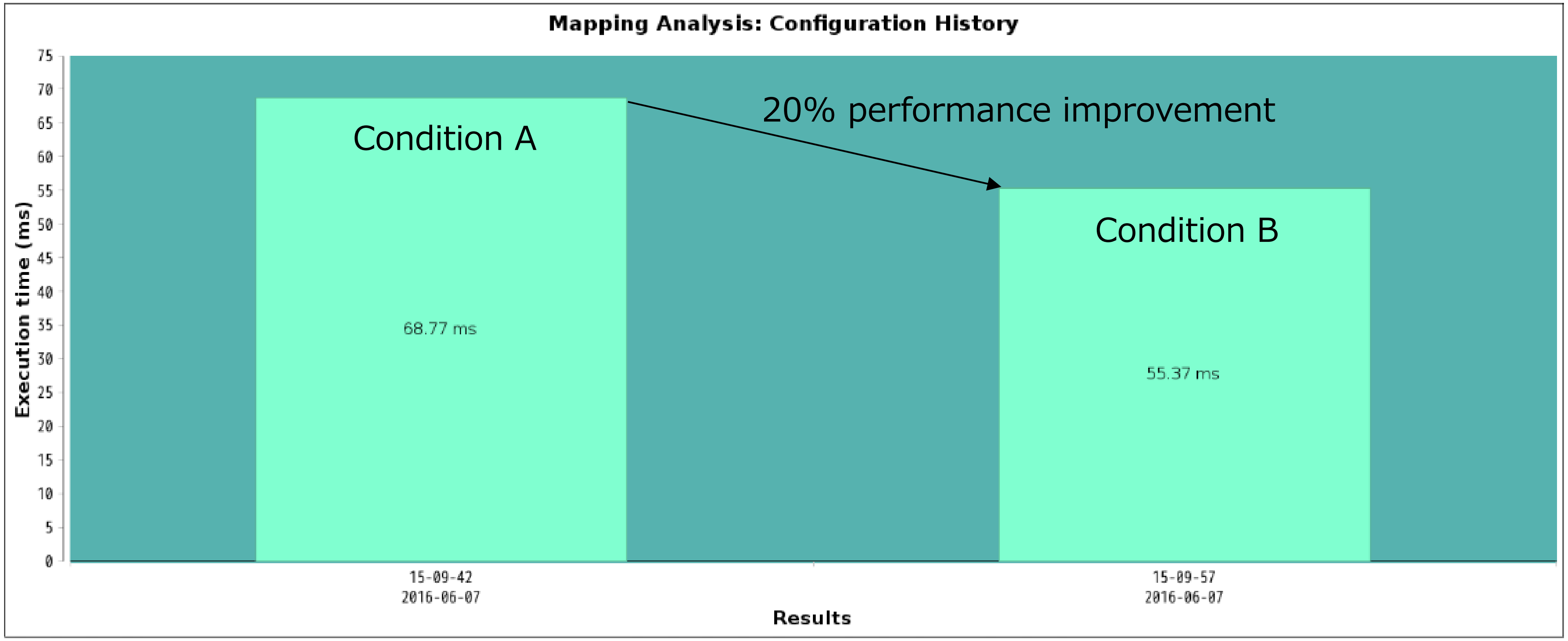


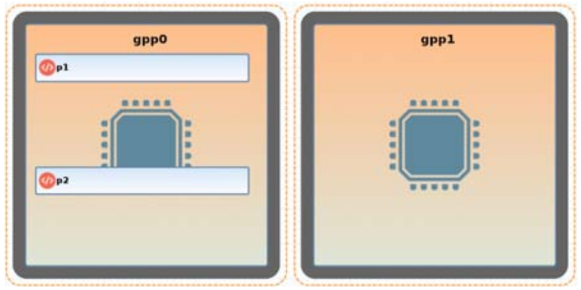
- Condition B
  - Mapping the two processes into two CPU

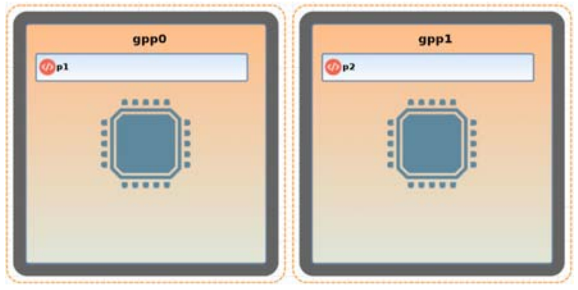


Result

First condition





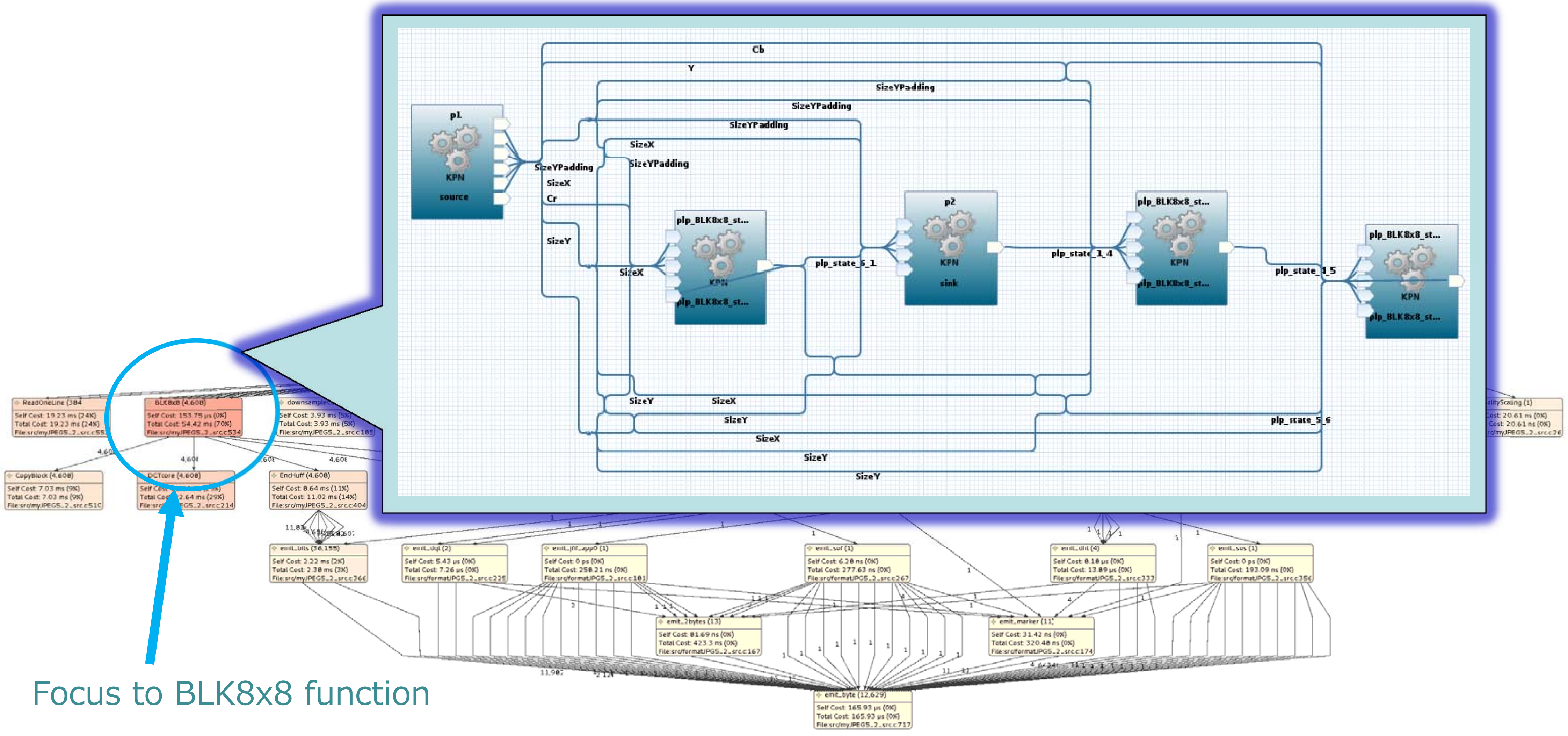


Performance improvement by parallelization was 20%

Parallelize

Second Trial – CPN

- Divide JPEG Encoder into 5 processes



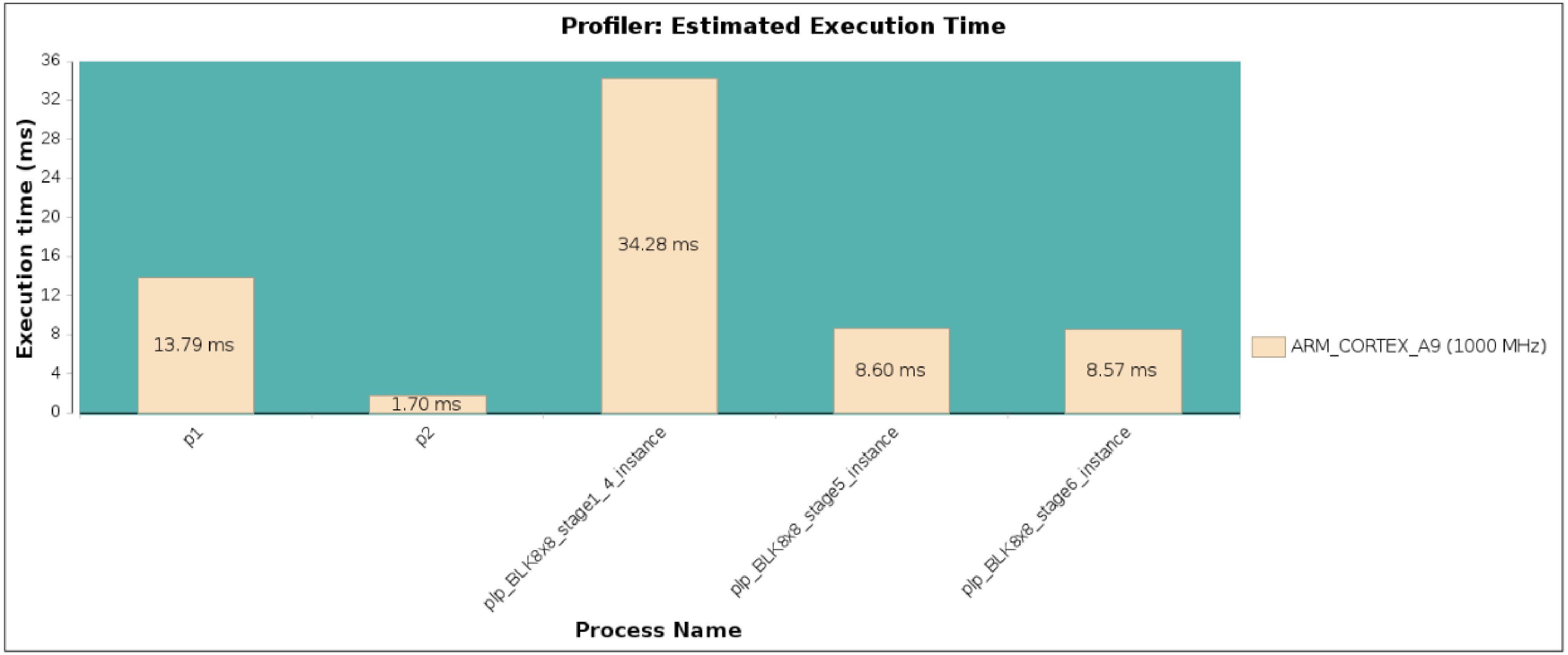
Focus to BLK8x8 function

Parallelize

RICOH
imagine. change.

Second Trial - Execution time

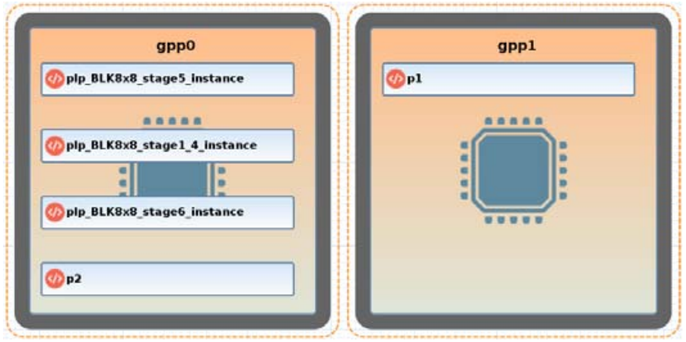
- Estimate execution time by ARM CA9 architecture model



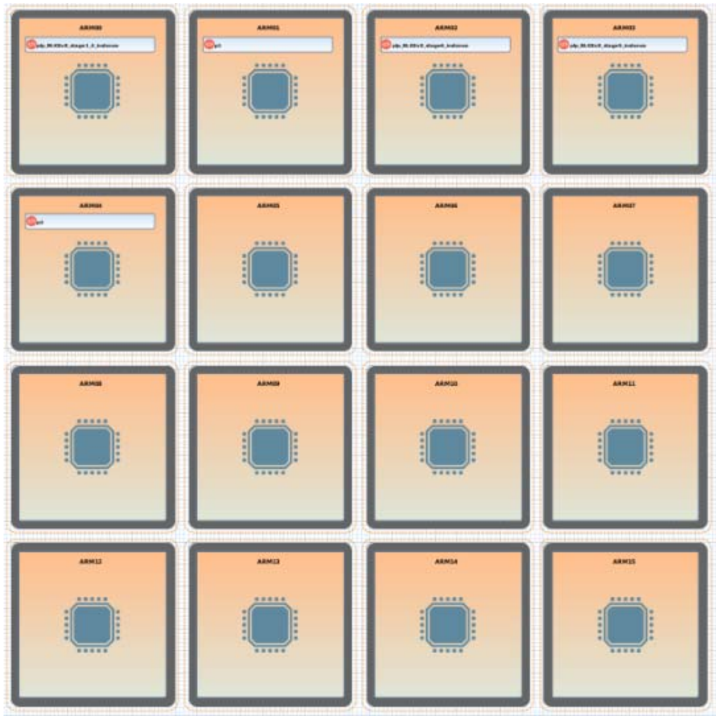
Architecture Mapping

Mapping condition

- Condition C – Mapping to Pandaboard
  - Mapping the two processes
    into two CPU by SLX automatically

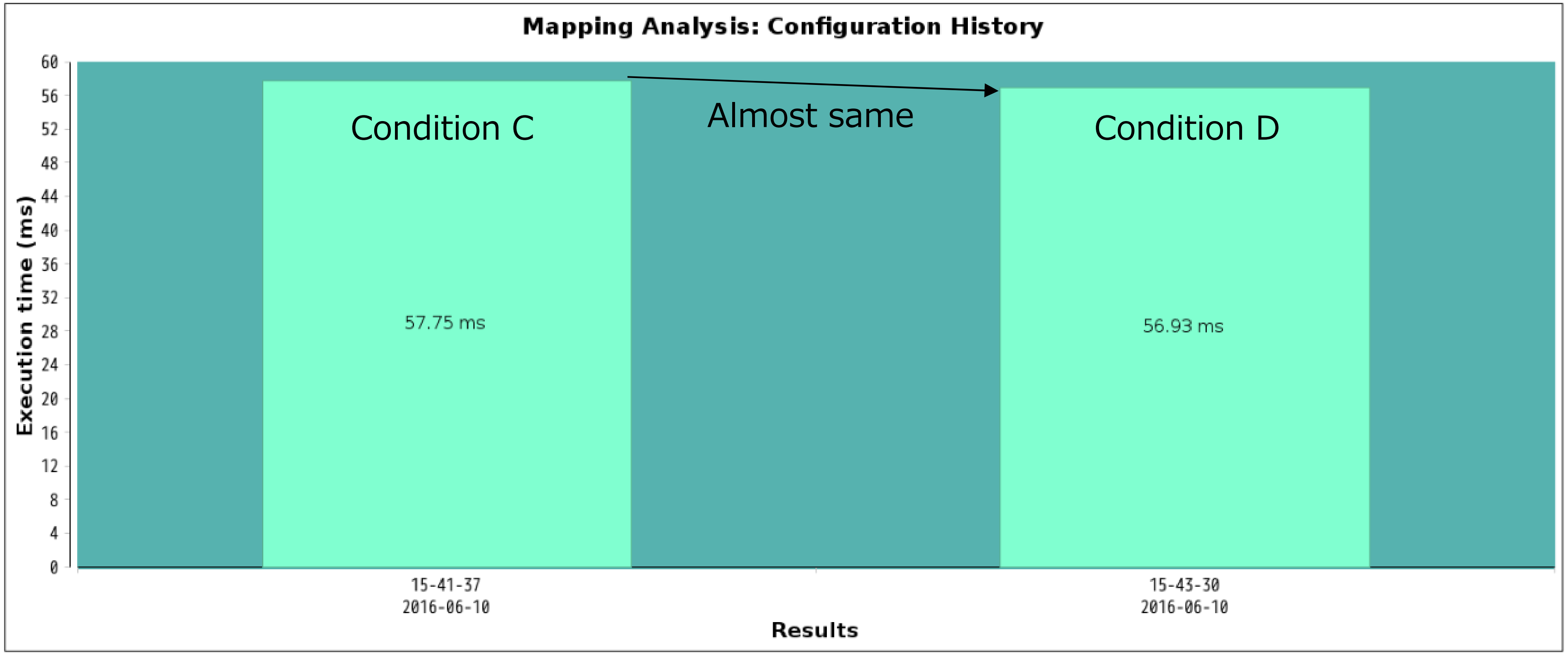

gpp0
plp_BLK8x8_stage5_instance
plp_BLK8x8_stage1_4_instance
plp_BLK8x8_stage6_instance
p2
gpp1
p1

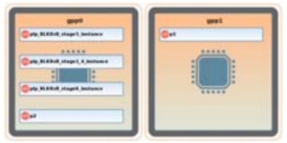
- Condition D – Mapping to 16 core ARM architecture
  - Mapping each processes into a CPU by manual



Result

Second Trial



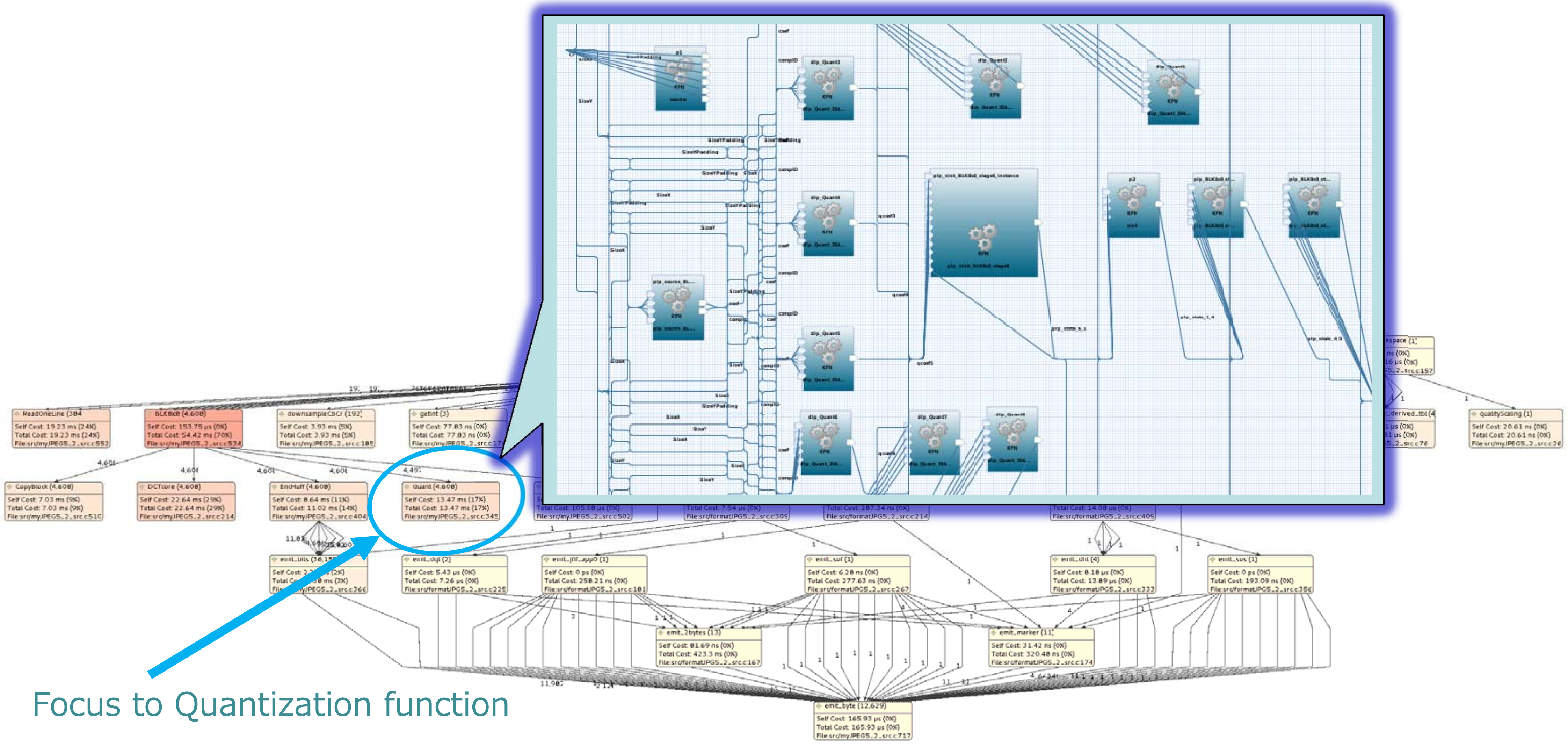


There is no effect of parallelization

Parallelize

Third Trial – CPN

- Divide JPEG Encoder into 14 processes

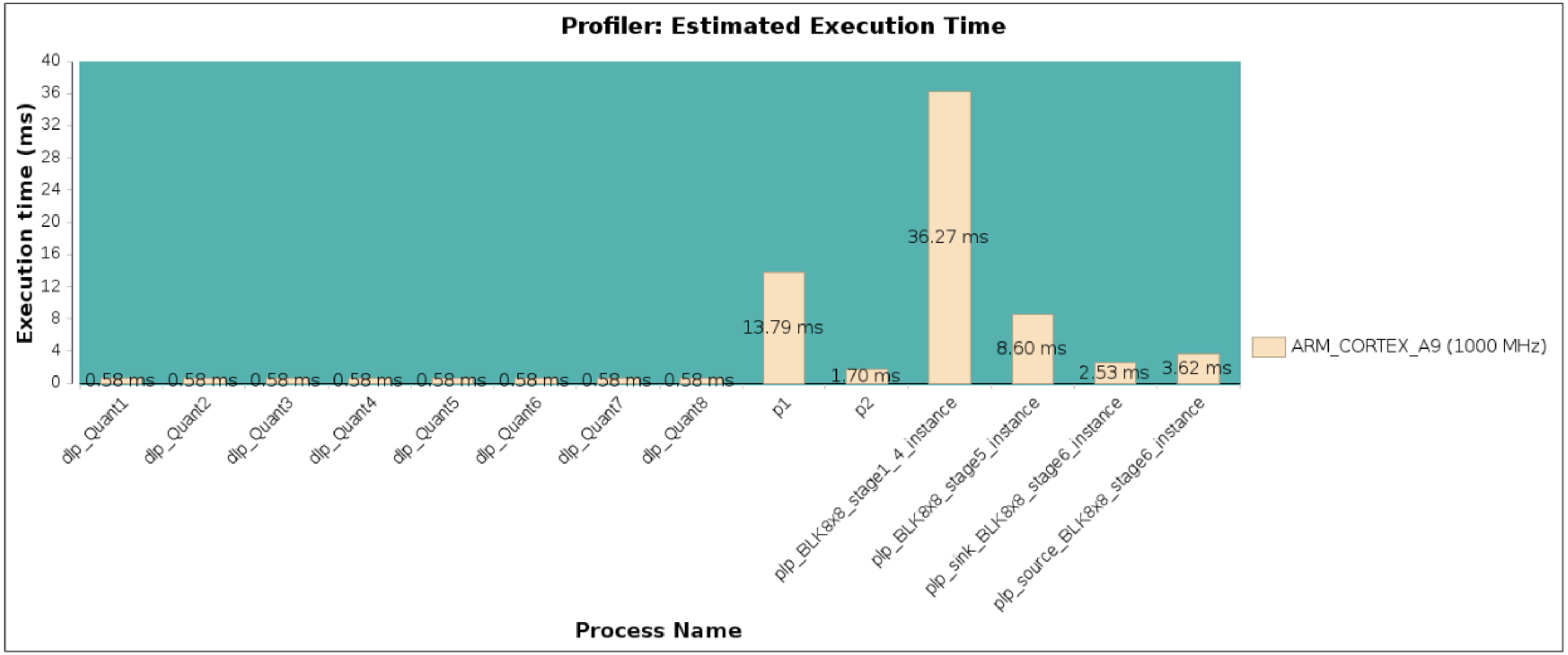


Focus to Quantization function

Parallelize

Third Trial - Execution time

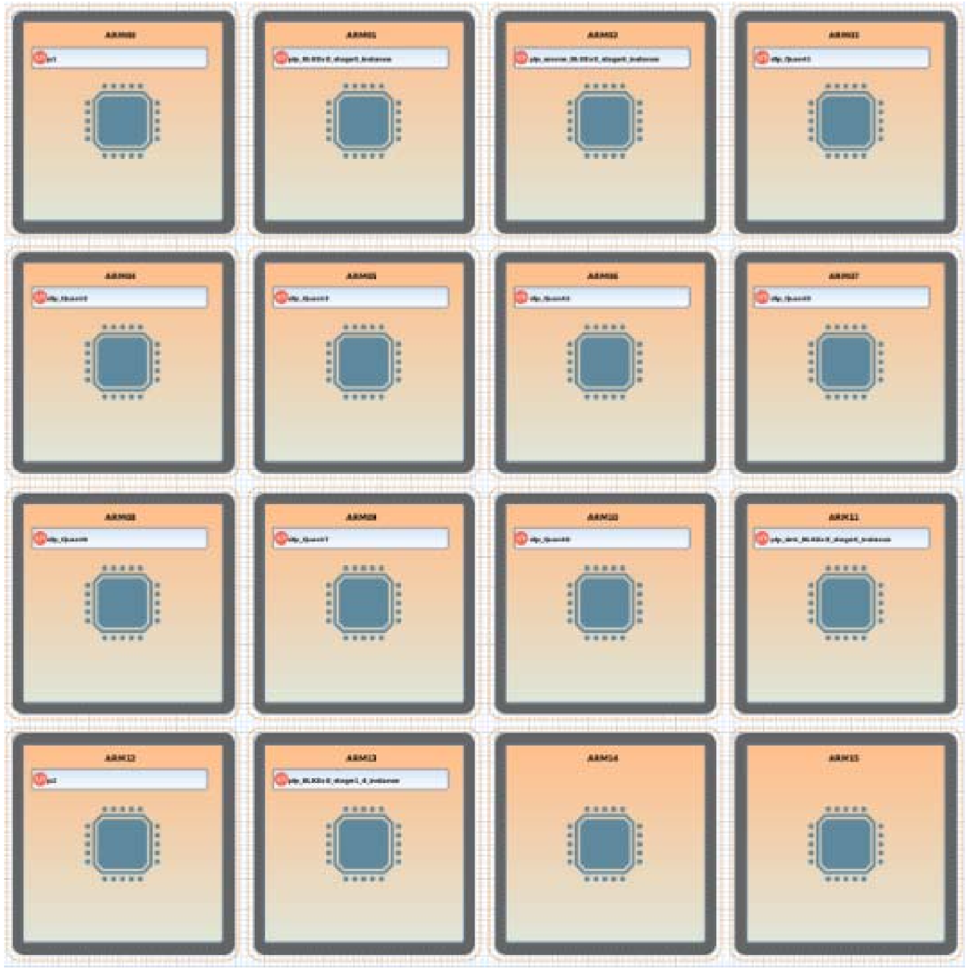
- Estimate execution time by ARM CA9 architecture model



Architecture Mapping

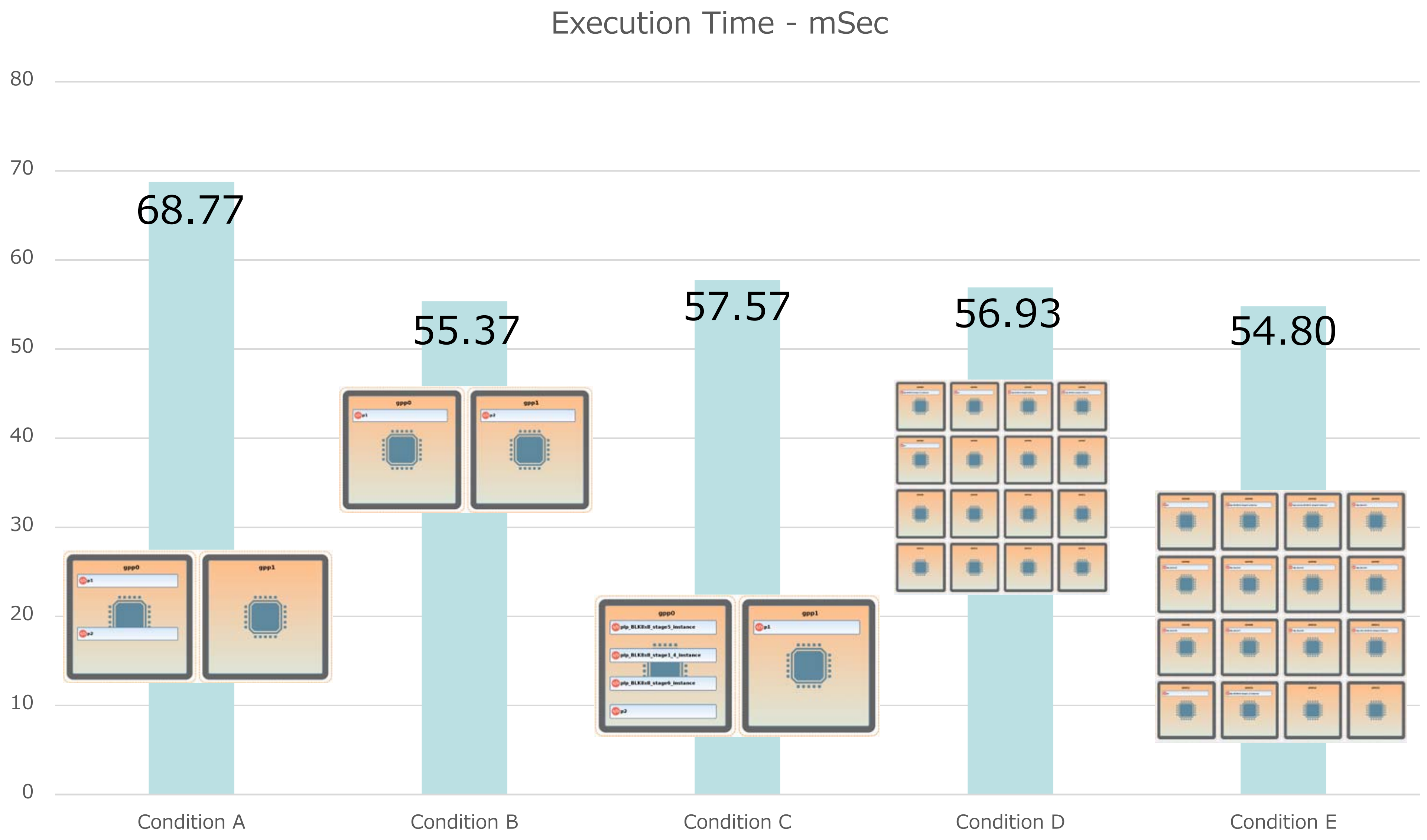
Mapping condition

- Condition E – Mapping to 16 core ARM architecture
  - Mapping each processes into a CPU by SLX automatically



Result

Comparison in all conditions



Conclusion

- **We tried to optimize the JPEG Encoder at the first step**
  - We use SILEXICA tool
  - To make the CPN is little difficult
  - SILEXICA is still semi-automatic flow
    - Need the automatic generation of the CPN

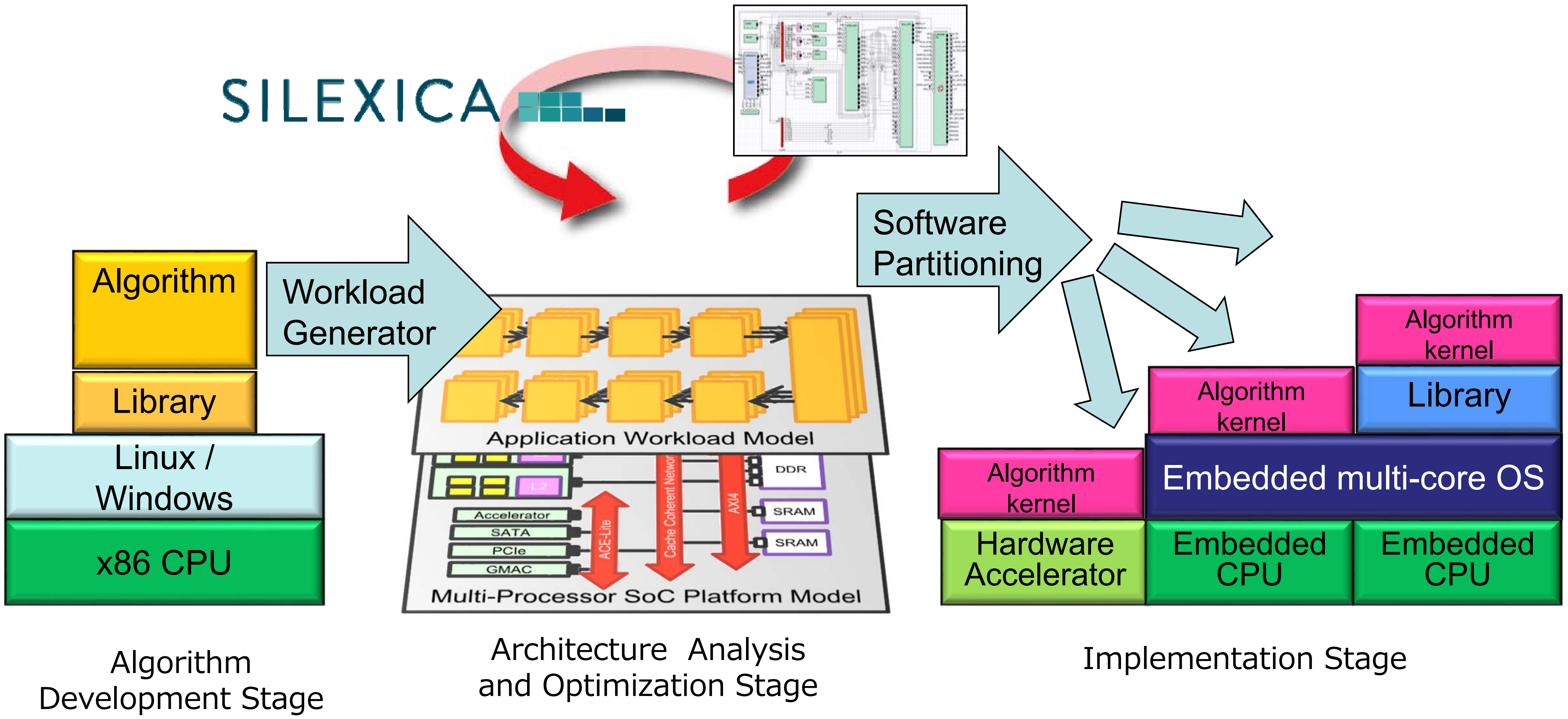
- **We need some division strategy for optimization**
  - There is a trade-off of division and communication between processes
  - We need the early architecture analysis with using virtual platform
    - We can get the image of strategy for optimization

- **We can feel the limitation of commoditization**
  - Hardware platform also important

Next Step

Tool-based Architecture Analysis and Optimization

- Fusion of tools – SILEXCIA and Virtual platform technology
  - Analysis architecture for performance by HW and SW co-design



Thank You !

RICOH

imagine. change.