High level synthesis for sequential nested loops: ~ Fully hardwired SLAM local bundle adjustment~

Atsushi Hatabu,[1] Toshihiko Nakamura,[1] Kenichi Miyazaki,[2]
Kazutoshi Wakabayashi[3]

[1]NEC Corporation, [2]MIRISE Technologies Corporation,
[3]The University of Tokyo

Agenda

1. Introduction
2. Local Bundle Adjustment
3. Why fully hardwired Local BA circuit?
4. Design with HLS (CyberWorkBench)
   - 4.1 Process Pipeline for data feedback loop
   - 4.2 Applicable conditions for loop optimizations
   - 4.3 Loop folding with carried variables
5. Results
   - 5.1 Results: loop folding DII exploration
   - 5.2 Results: entire circuits
6. Summary

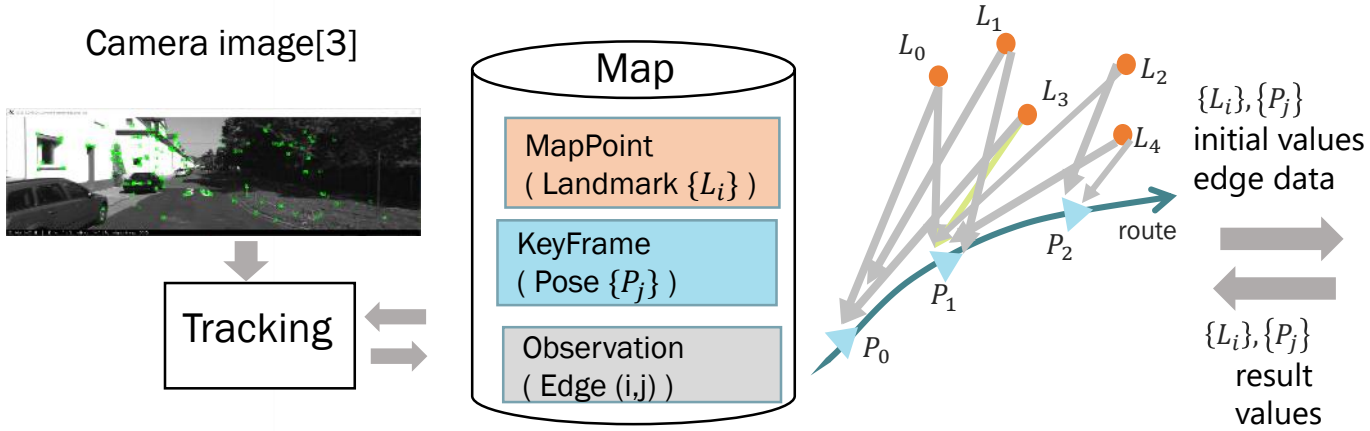
1. Introduction

- Local Bundle Adjustment (Local BA) is a key function to maintain the accuracy of mapping in Visual Simultaneous Localization And Mapping (SLAM) for AD/ADAS.
- Local BA processing speed: CPU 533ms*1, but our target is 15ms for AD/ADAS.

[Delay Budget analysis] : AD/ADAS needs 100ms[1]., **AD/ADAS: =** Recognition->Judge->control
Recognition:= CMOS sensor image -> Visual SLAM –> Camera Info. GNSS, other sensor fusion -> 3D map/pose

20~30ms for Visual SLAM (Tracking+LocalMapping) , therefore**, 10-15ms for Local BA.**

Visual SLAM system[2]



Local Bundle Adjustment

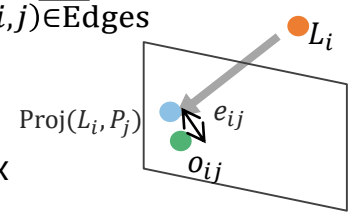
- refining 3D landmark variables $\{L_i\}$ and 6DF pose variables $\{P_j\}$

$$(\{L_i\},\{P_j\}) = \operatorname{argmin} \sum_{(i,j)\in \mathrm{Edges}} \rho(e_{ij}^{\mathrm{T}} \Omega_{ij} e_{ij})$$

$\rho(\cdot)$: robust kernel

e_{ij}: reprojection error

Ω_{ij}: information matrix



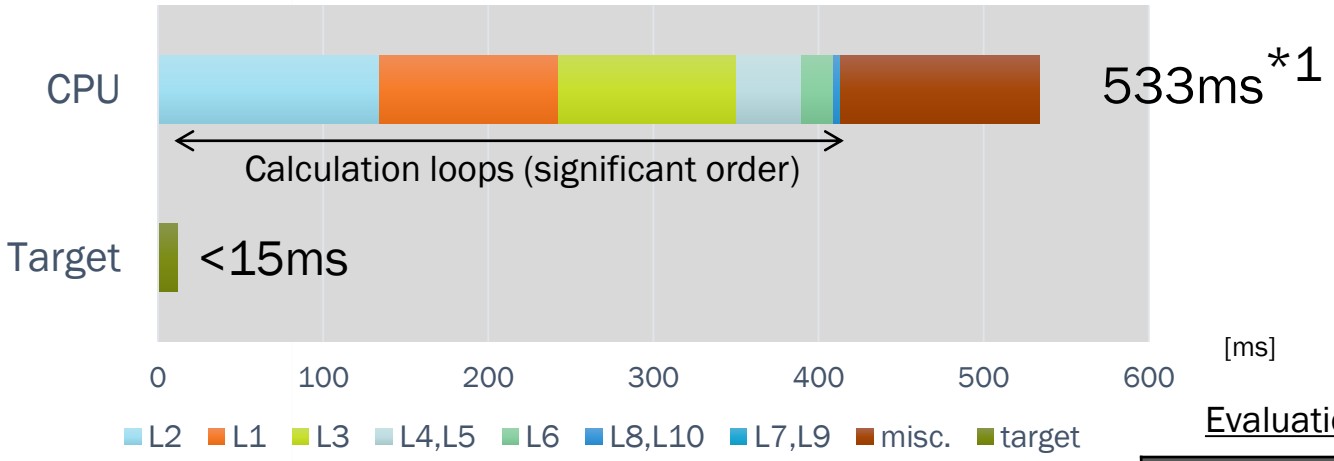
[1] S. Thrun et al., "Stanley: The Robot that Won the DARPA Grand Challenge," in Journal of Field Robotics, vol. 23, Issue 9, pp. 661-692, 2006.
[2] https://github.com/raulmur/ORB_SLAM2
[3] https://www.cvlibs.net/datasets/kitti/eval_odometry.php

*1 g2o library on ARM Cortex A72 1.5GHz, single thread, FP32 precision. https://github.com/RainerKuemmerle/g2o

2. Local Bundle Adjustment

- It contains 10 nested loops and data feedback. Therefore, simple pipelining cannot speed up.
- The internal loop latency should be reduced.

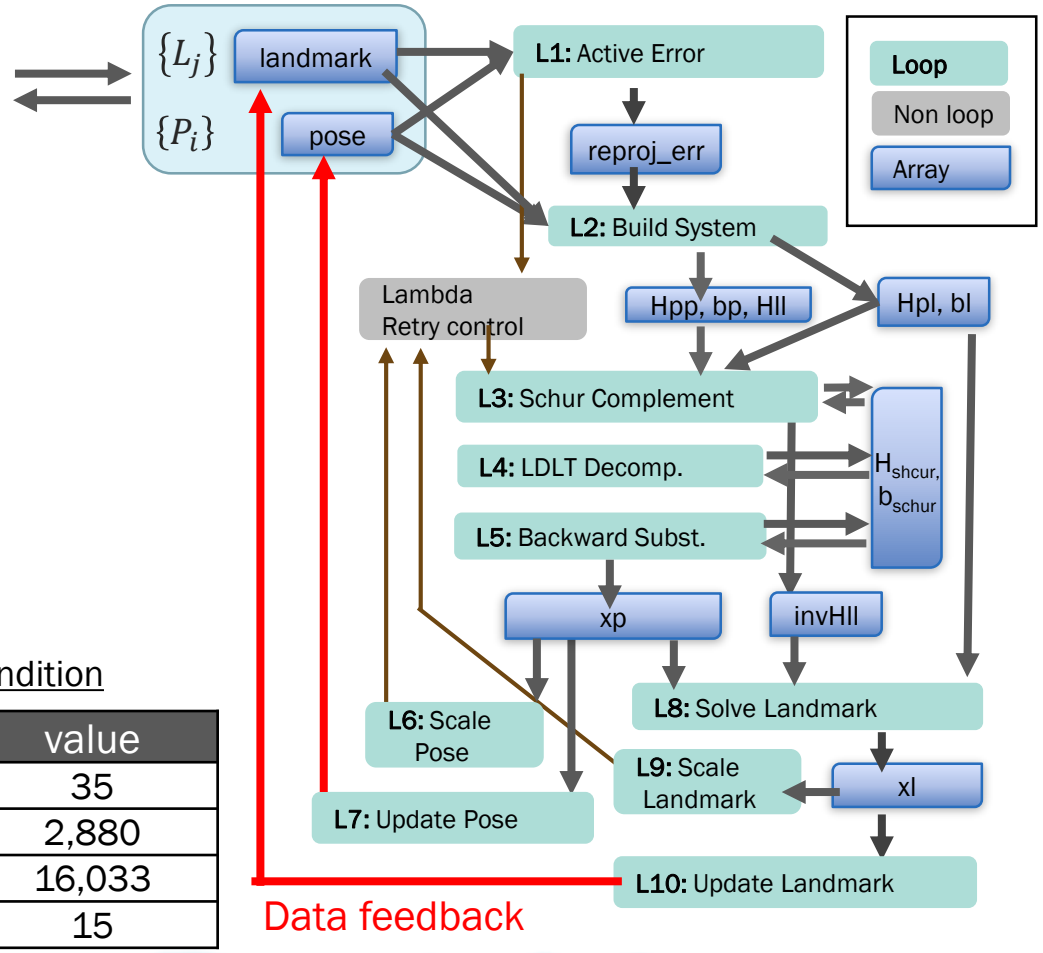
processing time for each loops(L1-L10)



Evaluation condition

Parameter	value
#pose	35
#landmark	2,880
#edge	16,033
#iteration	15

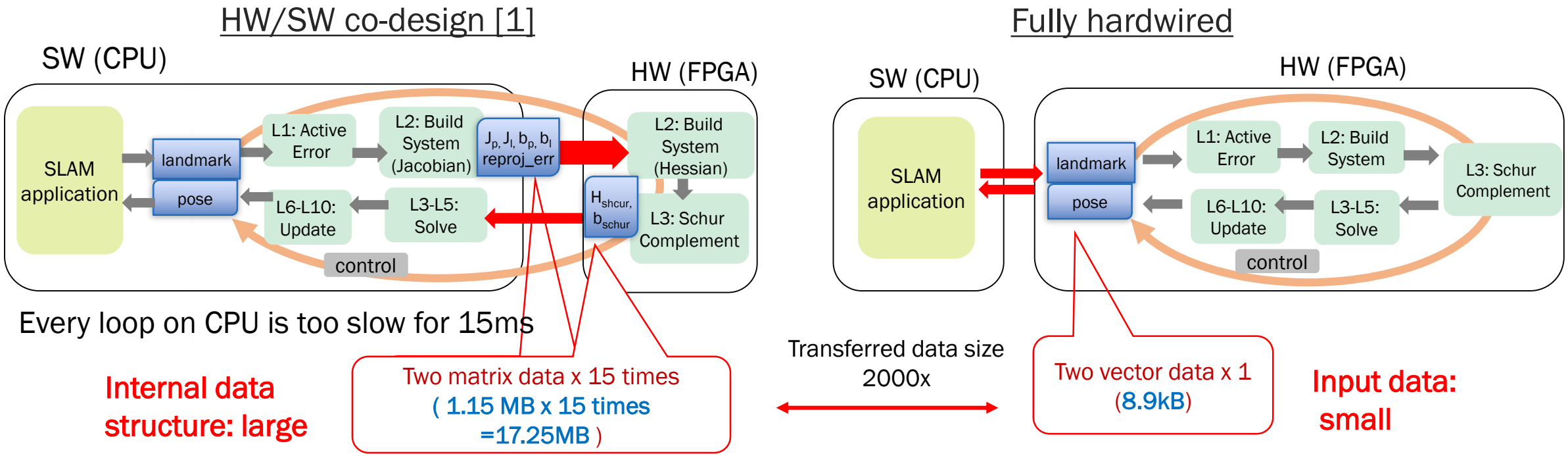
Data flow of Local BA processing



*1 ARM Cortex A72, 1.5GHz, single core used.
Since it only supports double precision, we have modified g2o code.
Time of 15 Levenberg-Marquardt algorithm iterations.

3. Why fully hardwired Local BA circuit?

- We developed a “fully hardwired” circuit for the entire Local BA with HLS
  - Partial hardware accelerator is not enough to achieve 15ms due to CPU-HW communication overhead and every loop is too slow if processed in CPU



[1] S. Qin, Q. Liu, B. Yu and S. Liu, “PI-BA Bundle Adjustment Acceleration on Embedded FPGAs with Co-observation Optimization,” in Proc. of IEEE FCCM 2019, 2019.

4.1 Process Pipeline for data feedback loop

Random access

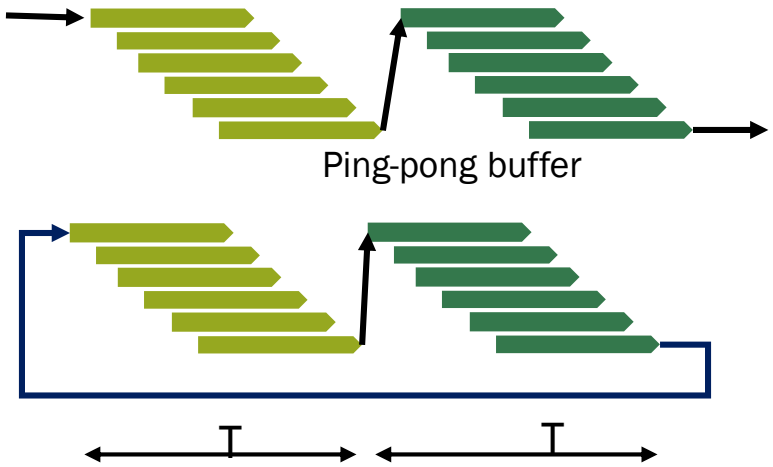
```
float a [1024];
for ( i=0; i<N; i++ ) {
    a[boo(i)] = f(i,…);
}
for ( i=0; i<N; i++ ) {
   … = a[i];
}
```

Sequential access

```
float a [1024];  sum=0;
While(c1){
for ( i=0; i<N; i++ ) {
    sum = sum + f(i,…);
   if (c1) a[k++] = sum;
}
for ( i=0; i<N; i++ ) {
   if (c2) sum= a[k++];
   … = g(i,sum,…);
}
```

Condition c1 and c1 become true the same number of times.
write data and read data is the same

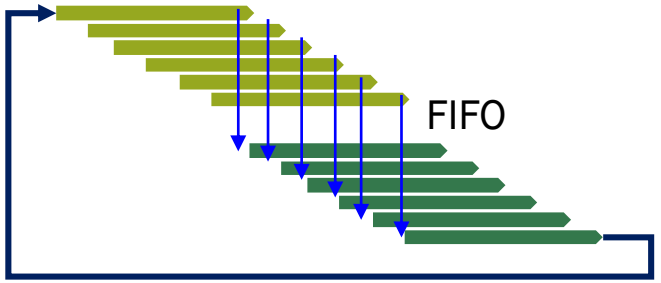
1) Simple Process pipeline with random access arrays



No Feedback
processes: parallel exec.
with DII=1 (throughput)

Data Feedback
processes: sequential exec.
with DII=1 (throughput)
Latency=2T

2) Process pipeline with sequential access arrays, and for the case loops cannot be merged



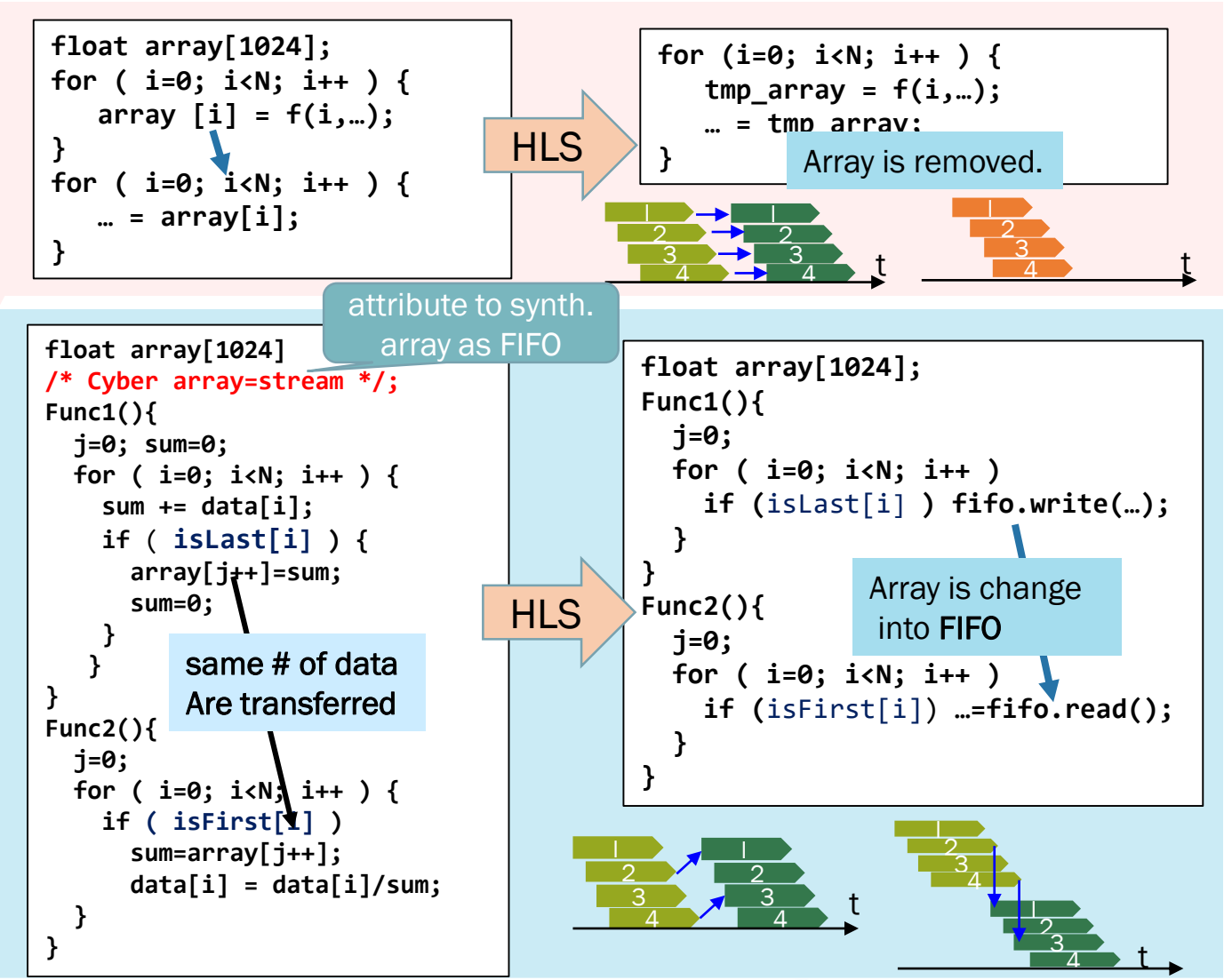
Data Feedback
Two processes Parallel exec.
with DII=1 (throughput)
Latency can be reduced
(latency < 2T)

4.2 applicable conditions for loop optimization

Loop merge: apply always if applicable
Process pipeline: if L.M. cannot be applied, it is applied, if applicable.

Applicable conditions are as follows

Data dependencies btw. loops	Applicable optimization
No dependency	Loop merge
1:Arrays: loop counter access, or synchronous to loop counter	Loop merge
2:Arrays: sequential access, asynchronous to loop counter, and same # of data transferred	Process pipelining with FIFO
3:Arrays: random access *1	Process pipelining with Ping-Pong
4: scalar variables	Sequential Circuits



*1 generally applicable, but not for loops with data feedback .

4.3 Loop folding with carried variables

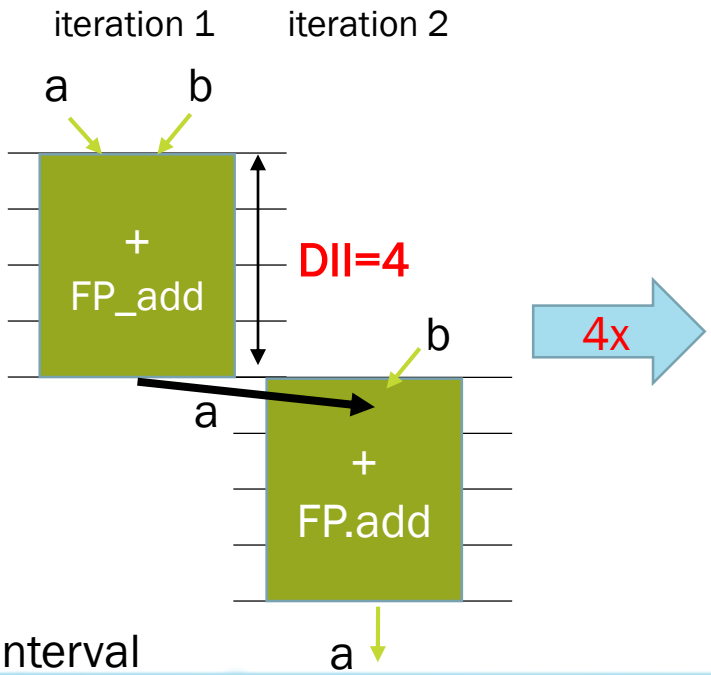
- A carried variable among iterations has data dependency, and it determines DII of pipeline.

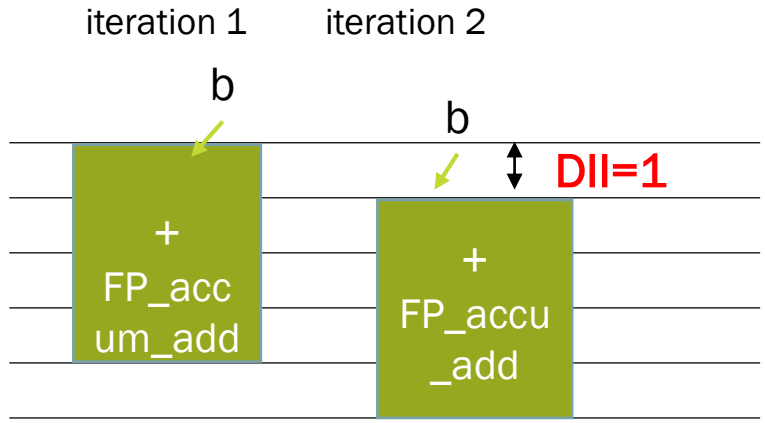
1: Pipeline scheduling with a 4 cycle floating adder generates DII=**4** pipeline circuit.

2: Pipeline scheduling with a 4 cycle floating accumulated adder generates DII=**1** pipeline circuits, since data dependency of "a" is deleted by changing FP_add to FP_accum_add. .

attribute of loop folding

```
float a, b;
a=0f;
//cyber folding=4, or 1
while(cond){
    ...;
    a=a+b;
    c=a;
     ...;
}
```





No data dependency for variable "a"

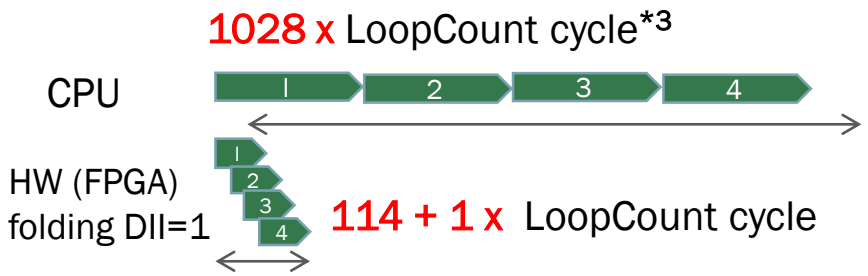
DII: Data Initiation Interval

5.1 Results: loop folding DII exploration

DII: variation exploration for L1-L2 (activeError and buildSystem)

device	folding		Area*1			Cycle*2		Time*2	
	DII	states	LUTs	REGs	DSPs	k cycle	speed up	ms	Speed Up
HW (FPGA)	1	115	86.7	145.0	441	242	1021.0	1.29	127.6
HW (FPGA)	2	118	54.5	48.8	231	483	512.3	2.57	64.0
HW (FPGA)	4	126	35.6	21.2	127	964	256.3	5.14	32.1
CPU*3			-	-	-	247,300	1.0	164.87	1.0

LoopCount=16033



CPU cycle : 1028 x Loop Count vs FPGA cycle: 114 + loop count => 1021x with 332 Fus.

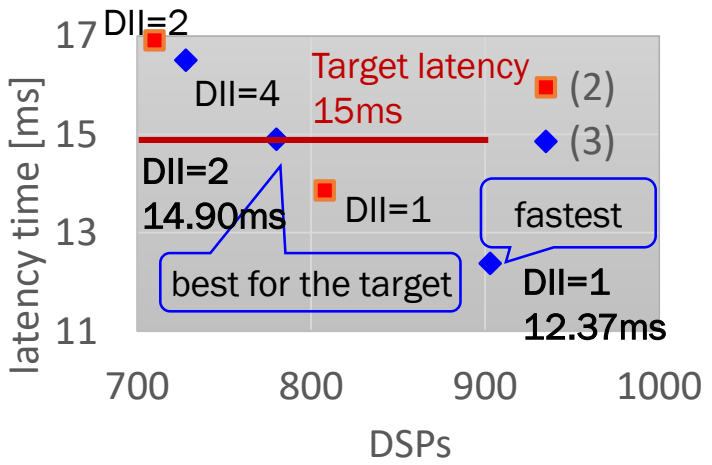
FU for DII=1

OP KIND	#FU
ADD	114
ACC_ADD	10
MUL	193
DIV	13
SQRT	2
Total	332

Calculation of reprojection error, robust kernel, Jacobian, Hessian etc.

Architecture exploration result for entire circuit for DII=1,2,4

LoopOptimization	(2) Loop merge and Loop folding		(3) +Process pipelining		
DII of L1-2,L7-9	1	2	1	2	4
time msec @ 187.5MHz	13.85	15.85	12.37	14.90	16.51
Slice LUTs[k]	305.5	200.0	367.7	255.0	206.8
Registers[k]	208.9	162.3	274.3	203.8	170.3
DSPs	808	710	903	780	738
BlockMem [bits]	264.2	264.2	264.2	264.2	264.2



DII=1, and DII=2 circuits meet the 15ms requirement.
DII=2 is smaller than DII=1.

If we use ASIC for this Local BA, DII=4 can be good candidate.
This results shows how HLS architecture exploration is effective

*1 estimated the by HLS tool, device: Zynq UltraScale+ MPSoC XCZU19EG, Frequency: 187.5MHz, accuracy: FP32
*2 processing cycle or time in 15 iterations of loop L1-L2. where the loop count is 16033.
*3 ARM Cortex-A72 1.5GHz, single core. accuracy FP32, processing tme of function g2o::SparseOptimizer::computeActiveErrors(), SparseOptimizer::activeRobustChi2() and BlockSolver::buildSystem()

5.2 Results: entire circuits

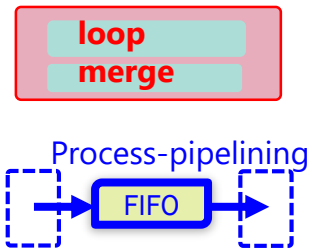
Effects of Loop optimization for DII=1

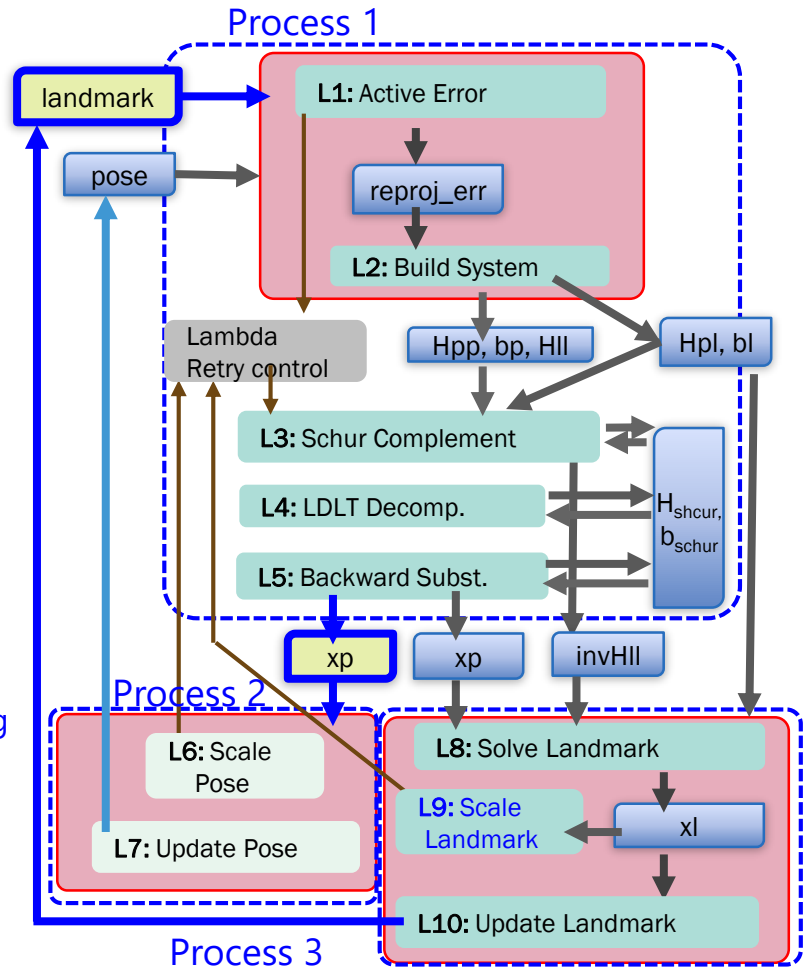
	Latency time*1 [ms]	Speed Up (total)	Speed Up (single)	Slice LUTs*1 [k]	Registers *1 [k]	DSPs*1
(0) **Sequential circuit**	660.9	1.00	-	104.7	35.2	168
(1) **+Loop merge**	561.5	1.18	1.18	99.8	35.0	168
(2) **+Loop folding**	13.9	47.7	40.5	305.5	213.9	808
(3) **+Process pipelining**	12.4	53.4	1.12	367.7	274.3	903

Latency comparison with CPUs

Device	Freq. [MHz]	Time [ms]	Speed up in time	Speed up (cycle)
CPU (ARM)*2	1500	533.3	1.00	1.00
FPGA	187.5	12.4	43.1	344.69

Optimization result





*1 Zynq UltraScale+ MPSoC XCZU19EG, 187.5MHz, estimated the by HLS tool, accuracy: FP32
*2 ARM Cortex A72, 1.5GHz, single core used. g2o FP32 precision

7. Summary

- A fully hardwired local Bundle Adjustment for Visual SLAM is designed with our HLS (CyberWorkBench)
  - Achieve 15ms to realize safe AD/ADAS.
  - Speed up: 43 times (FPGA 187.5MHz vs ARM 1.5GHz), 345x faster in cycle count
- Three types of loop optimization is applied with HLS
  - Shows applicable conditions for these.
  - Effects analysis; speed up ratio from sequential circuit.
    Loop merge: 1.18x, Loop folding 40.5x, Process Pipeline 1.12x, overall 53.4x
- HLS advantages is demonstrated with this design
  - Easy Architecture exploration only with changing synthesis attribute
  - Algorithm designers design this chip with 2men x 3 months. Algorithm optimization is much more effective than RTL optimization.

This presentation is based on results obtained from a project, JPNP23015,
commissioned by the New Energy and Industrial Technology Development Organization (NEDO).