T2S: Programming Spatial Architectures for Productive Performance

Hongbo Rong, Xiaochen Hao, Mingzhe Zhang, Yun Liang, Wenguang Chen

Intel Labs, Peking Univ., Tsinghua Univ., Univ. of Science & Technology of China

FCCM, May 12th, 2021

Thanks to the contributions

- Prof. Zhiru Zhang (Cornell)
  - ✔ Yi-Hsiang Lai
  - ✔ Nitish Srivastava
  - ✔ Shaojie Xiang
  - ✔ Brendan Sullivan
- Prof. Yun Liang (PKU)
  - ✔ Xiaochen Hao
  - ✔ Lianwei Cui
  - ✔ Size Zheng
  - ✔ Yunshan Jia
  - ✔ Xiuping Cui
- Prof. Wenguang Chen (TSU)
  - ✔ Mingzhe Zhang
  - ✔ Guanyu Feng
  - ✔ Huanqi Cao
- Prof. Youhui Zhang (TSU)
  - ✔ Weihao Zhang
- Prof. Vivek Sarkar (GaTech)
  - ✔ Prithayan Barua
- Prof. Jason Cong (UCLA)
  - ✔ Jie Wang

And thanks to the support and help of many people at Intel PCL, SSG, PSG, VTT, DevCloud, Legal, et al.

Disclaimer

The software, tutorial and any accompanying documentation (“Materials”) are provided “as is” with no warranties of any kind, whether written, oral, implied or statutory, including warranties of merchantability or fitness for a particular purpose, non-infringement or arising from course of dealing or usage in trade.

These Materials contain the general insights and opinions of Intel Corporation (“Intel”). The information in these Materials are provided for information only and are not to be relied upon for any other purpose than educational. Intel makes no representations or warranties regarding the accuracy or completeness of the information in this Material. Intel accepts no duty to update this Material based on more current information. Intel is not liable for any damages, direct or indirect, consequential or otherwise, that may arise, directly or indirectly, from the use or misuse of the information in this Material.

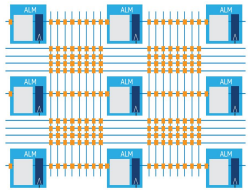
Agenda

- Concept of T2S
- Access to the tool and tutorials
- A deep dive with matrix multiply as an example
- Summary

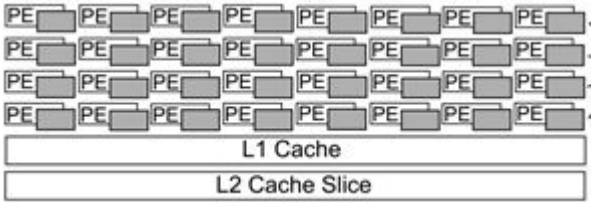
Spatial architectures: All about power efficiency & performance

- Massive compute resources, plus memory, distributed over a 2-D plane
- Application defines custom pipelines
  - Exploit massive parallelism, and minimize data movement
  - Potential for big boost to power efficiency and thus performance

FPGA

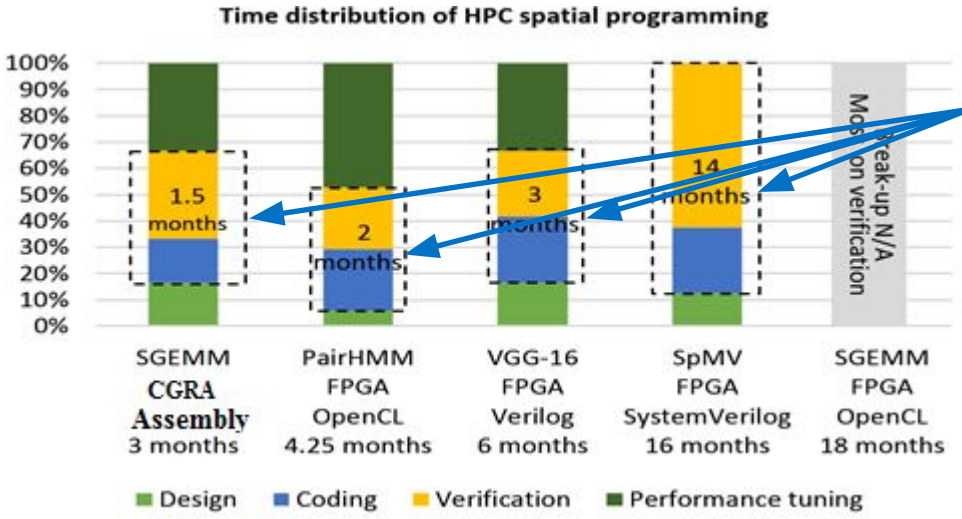


CGR



1. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01220-hyperflex-architecture-fpga-socs.pdf
2. A. Parashar *et al.*, "Efficient Spatial Processing Element Control via Triggered Instructions," in *IEEE Micro*, vol. 34, no. 3, pp. 120-137, May-June 2014.

HPC spatial programming is hard



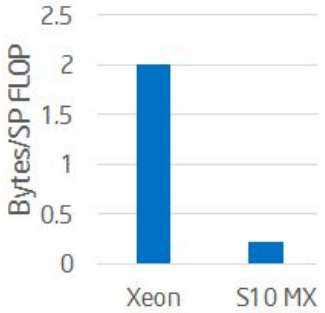
Most time on coding, especially verification

Source of data: Daya Khudia (Intel, SSG), Gorge Powley (Intel, DCG), Yufei Ma (ASU), Jeremy Fowers (Microsoft), Davor Capalija and Tomasz Czajkowski (Intel, PSG)

What to do:

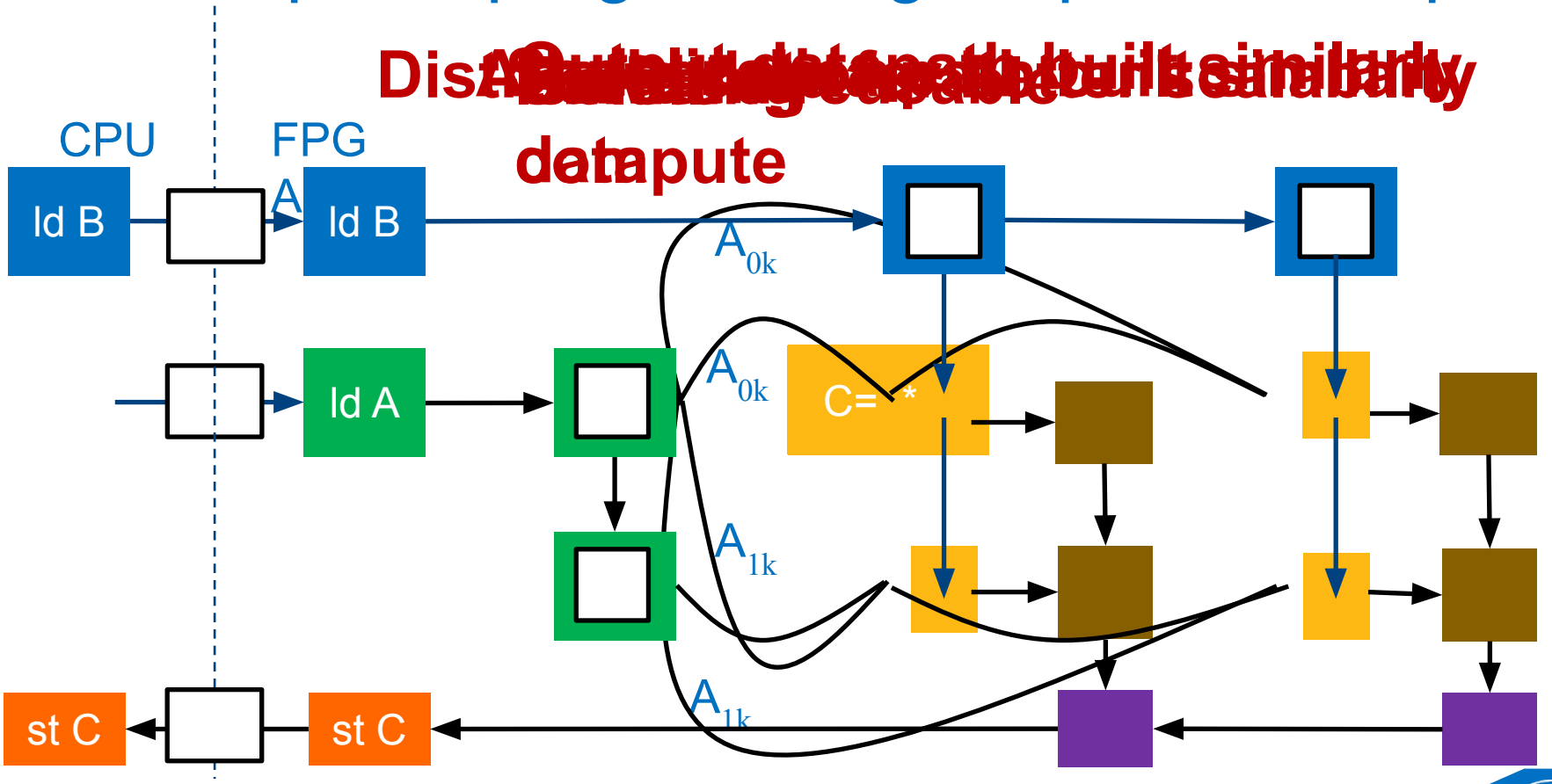
- Reduce coding and verification efforts dramatically

1. Very deep pipeline parallelism: key for high-performance *only* on spatial archs
2. Dramatically lower bandwidth/compute ratio than CPU
3. Prohibitively expensive design space exploration
4. Poor debuggability



Source of data: Christopher J. Hughes (Intel, PCL)

T2S: spatial programming for productive perf



Hypotheses and Validation

- Hypothesis: as long as the same set of optimizations are applied, compiler-generated perf should match ninja perf, but with 1-2 orders of magnitudes of higher productivity.
- Validation: the hypothesis holds at least for dense tensor kernels. 82-92% ninja perf with 9% engineering time across FPGA and CGRA

A10 FPGA

	T2S	Ninja
LOC	20	750
Systolic Array Size	10×8	10×8
Vector Length	16×float	16×float
# Logic Elements	214K (50%)	230K (54%)
# DSPs	1,282 (84%)	1,280 (84%)
# RAMs	1,384 (51%)	1,069 (39%)
Frequency (MHz)	215	245
Throughput (GFLOPs)	549	626

A10 FPGA

Benchmarks	LOC	Frequency (MHz)	Throughput (GFLOPs)
MTTKRP	28	204	700
TTM	30	201	562
TTMc	37	205	738

CGRA

	LOC	Throughput w.r.t Ninja GEMM	FMA usage
GEMM	40	92%	100%
MTKRP	32	99%	100%
TTM	47	104%	100%
TTMc	38	103%	95%

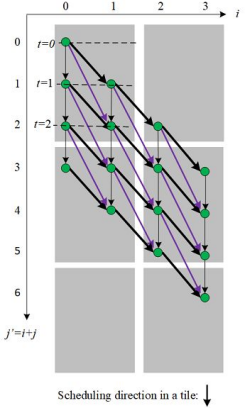
New early results on FPGAs

Capsule	S10	338 GFLOPS (71% peak)
PairHMM	S10	39 GCUPS for fixed-sized inputs
LU	A10	24 GFLOPS for 8x8 matrices

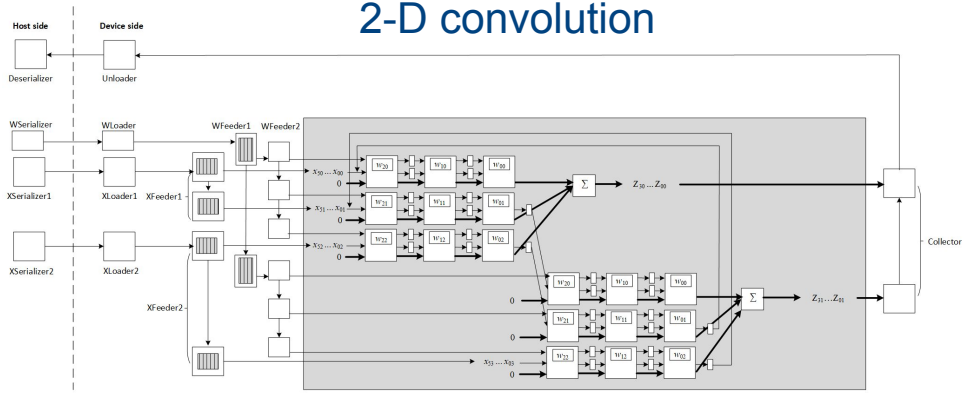
Hypotheses and validation (cont.)

- Hypothesis: A wide range of compute patterns and systolic arrays can be expressed based on UREs (Uniform Recurrence Equations) and space-time transforms.
- Validation: the hypothesis holds at least for dense tensors, dynamic programming, and stencils, and for 1-D, 2-D rectangular or triangular systolic arrays.

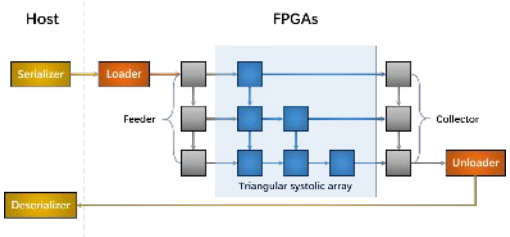
Parallel MM



2-D convolution

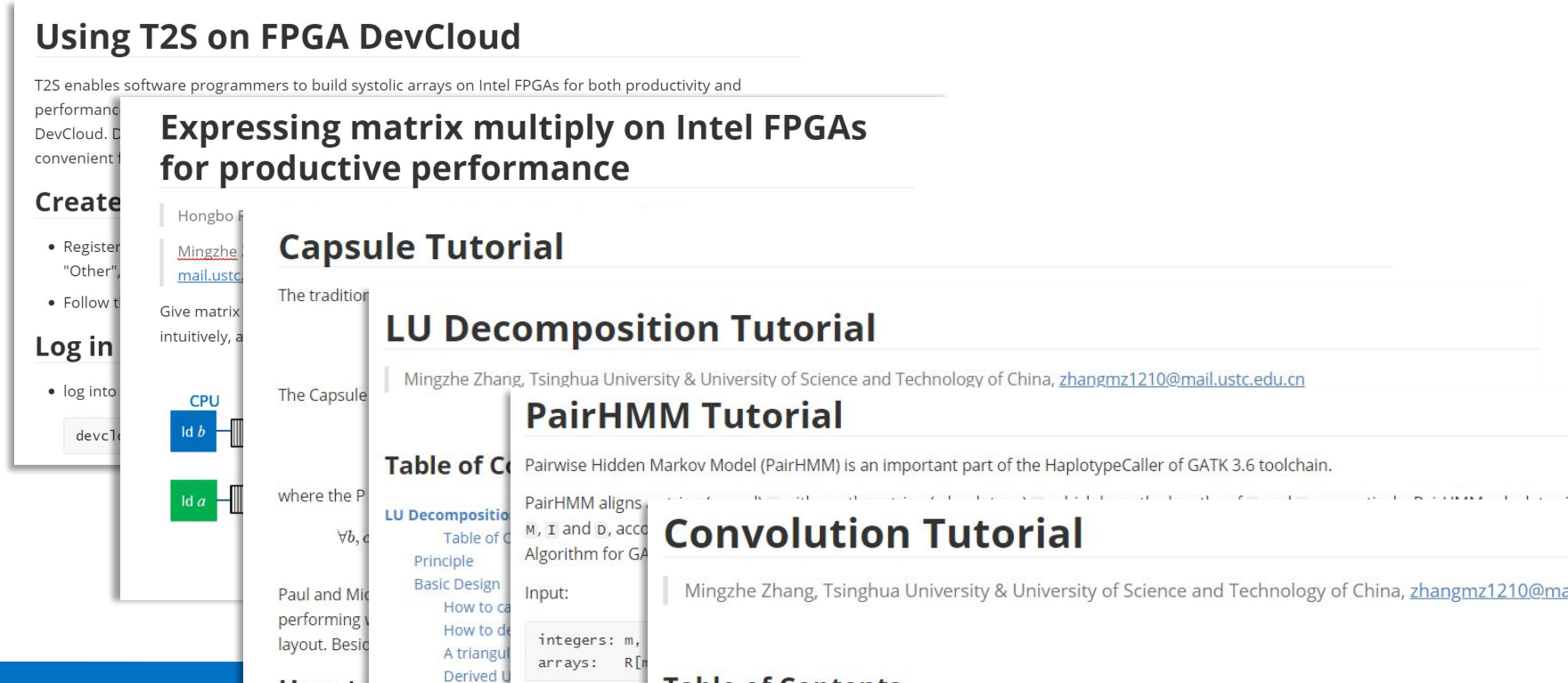


LU



Access to the tool

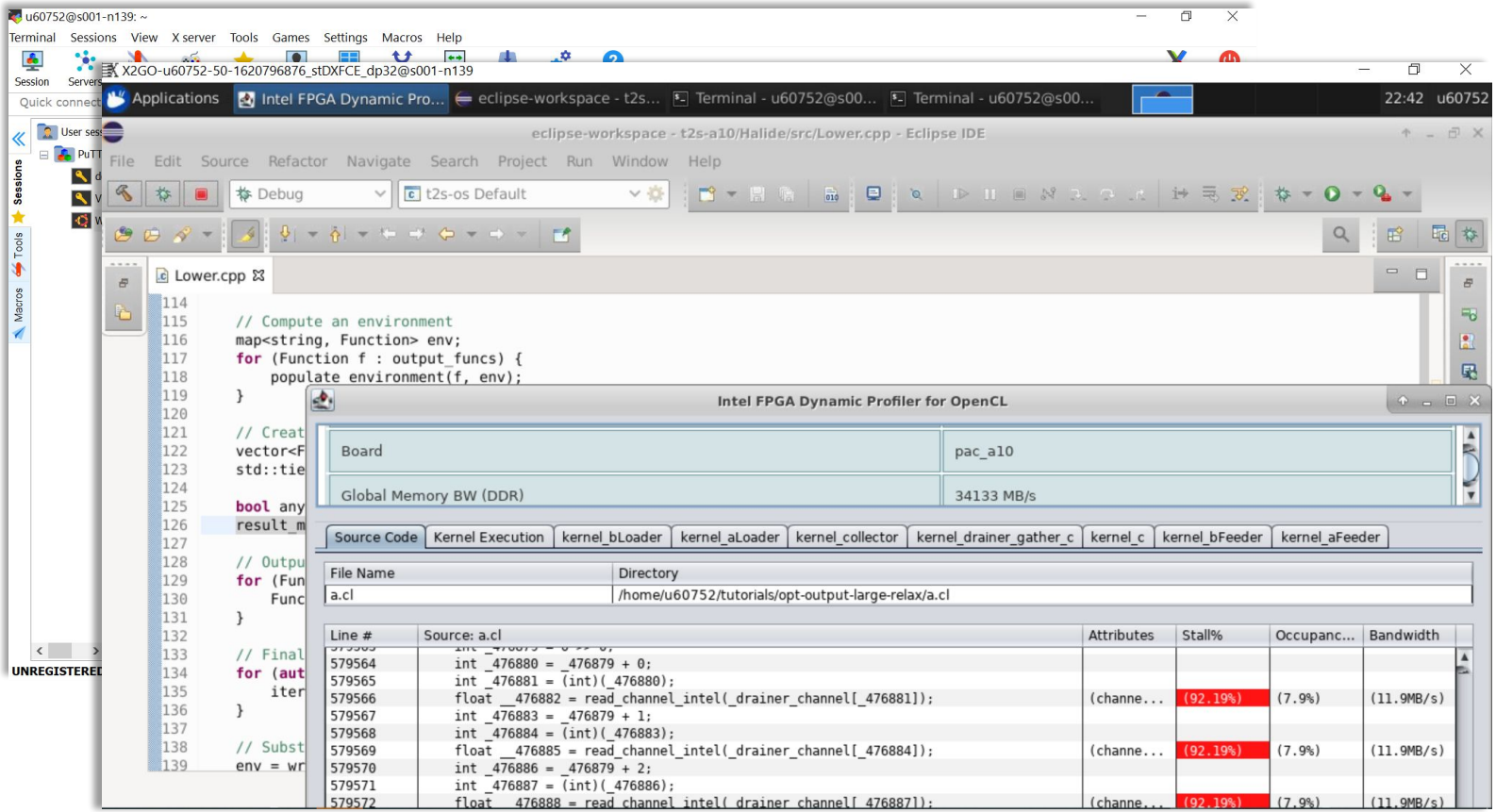
- A tool binary, and a set of tutorials, are freely available at Intel DevCloud
  https://github.com/intel/FPGA-Devcloud/tree/master/main/QuickStartGuides/T



Features

- Dataflow representation in UREs
- Loop transforms
  - Space-time transform, vectorization, unrolling, flattening, infinitization
- Isolation
- Double buffering
- Data scattering and gathering

DevCloud environment

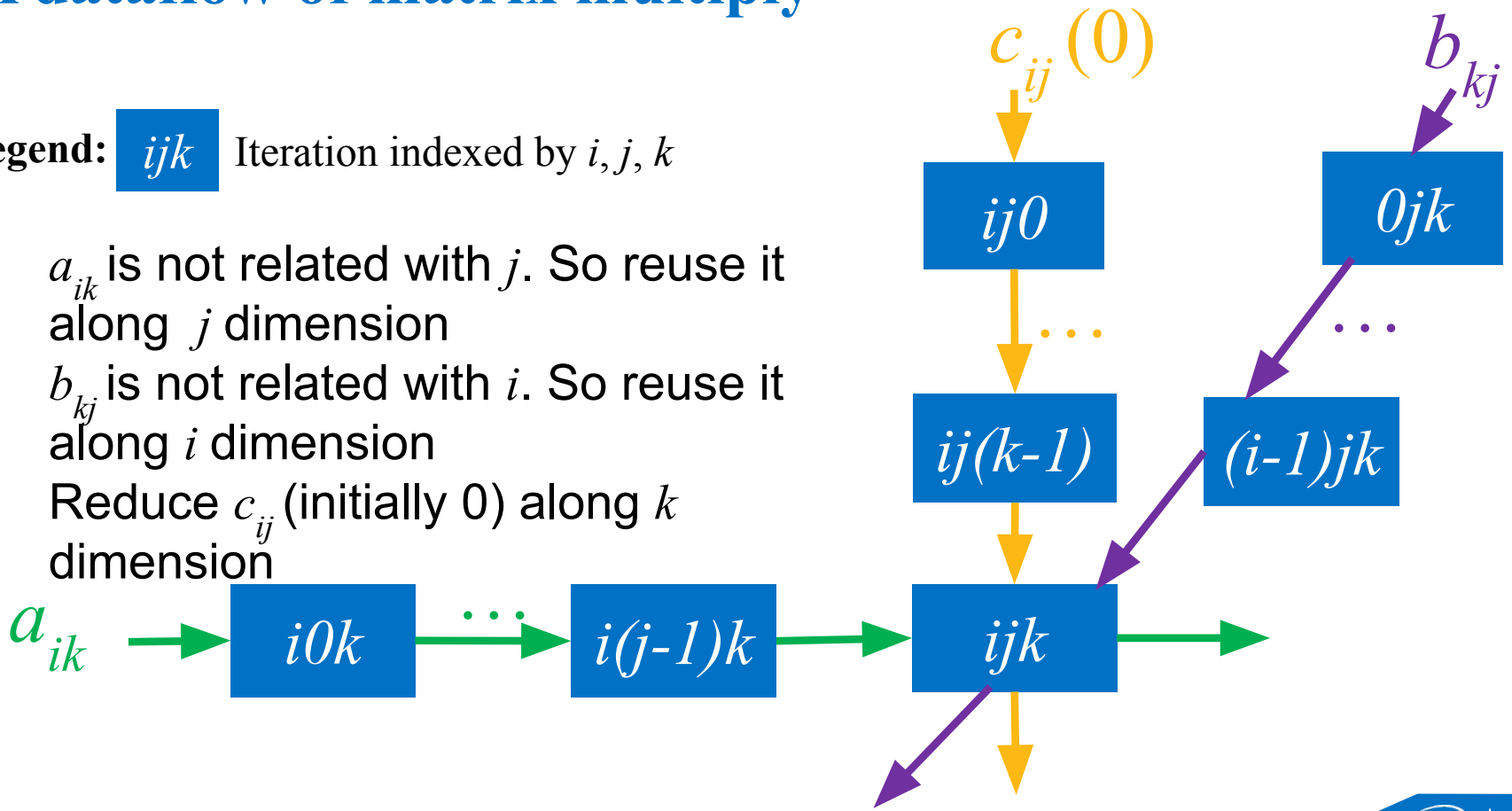


A deep dive with matrix multiply as an example

A dataflow of matrix multiply

Legend: *ijk* Iteration indexed by i, j, k

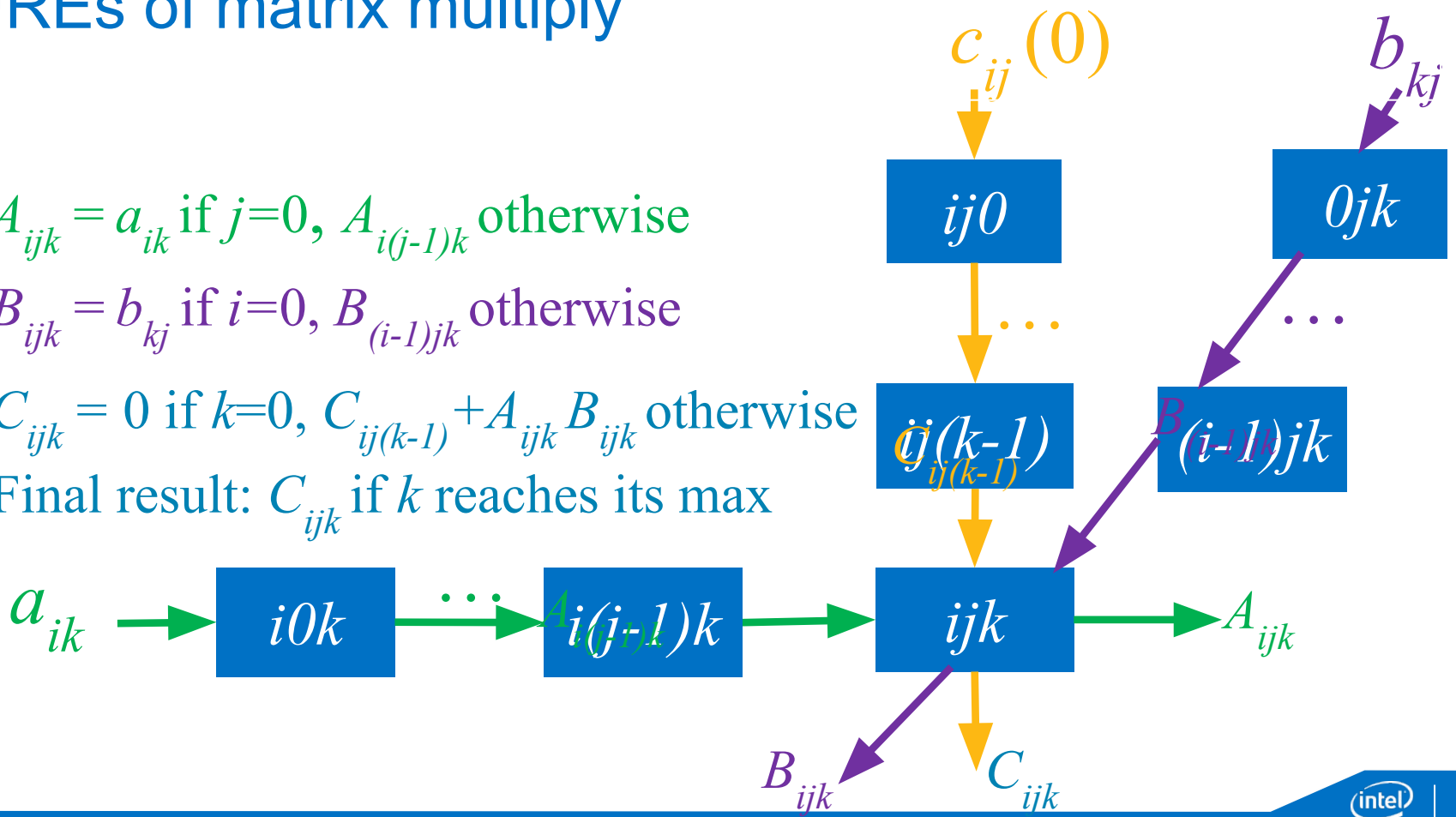
- a_{ik} is not related with j. So reuse it along j dimension
- b_{kj} is not related with i. So reuse it along i dimension
- Reduce c_{ij} (initially 0) along k dimension



UREs of matrix multiply

$$\begin{cases} A_{ijk} = a_{ik} \text{ if } j=0,\ A_{i(j-1)k} \text{ otherwise} \\ B_{ijk} = b_{kj} \text{ if } i=0,\ B_{(i-1)jk} \text{ otherwise} \\ C_{ijk} = 0 \text{ if } k=0,\ C_{ij(k-1)} + A_{ijk} B_{ijk} \text{ otherwise} \end{cases}$$

Final result: C_{ijk} if k reaches its max



T2S specification

- Following Halide convention
  - Writing arguments starting from the innermost loop
  - Matrices are column-major
- select(condition, x, y)
  - An expression (condition? x : y).

$$A_{ijk} = a_{ik} \text{ if } j=0, A_{i(j-1)k} \text{ otherwise}$$

$$B_{ijk} = b_{kj} \text{ if } i=0, B_{(i-1)jk} \text{ otherwise}$$

$$C_{ijk} = 0 \text{ if } k=0, C_{ij(k-1)} + A_{ijk} B_{ijk} \text{ otherwise}$$

Final result: C_{ijk} if k reaches its max

```
A(k, j, i) = select(j == 0, a(k, i), A(k, j - 1, i));
B(k, j, i) = select(i == 0, b(j, k), B(k, j, i - 1));
C(k, j, i) = select(k == 0, 0, C(k - 1, j, i)) + A(k, j, i) * B(k, j, i);
c(   j, i) = select(k == K - 1, C(k, j, i));
```

UREs

Output

T2S specification

```
1 for (i = 0; i < I; i++)
2  for (j = 0; j < J; j++)
3   for (k = 0; k < K; k++)
4    A(k, j, i) = select(j == 0, a(k, i), A(k, j - 1, i));
5    B(k, j, i) = select(i == 0, b(j, k), B(k, j, i - 1));
6    C(k, j, i) = select(k == 0, 0, C(k - 1, j, i))+ A(k, j, i) * B(k, j, i);
7    c(    j, i) = select(k == K - 1, C(k, j, i));
```

- A.merge_ures(B, C, c)
  - Merge all functions into a single loop nest
  - A will then represent this loop nest

```
A(k, j, i) = select(j == 0, a(k, i), A(k, j - 1, i));
B(k, j, i) = select(i == 0, b(j, k), B(k, j, i - 1));
C(k, j, i) = select(k == 0, 0, C(k - 1, j, i)) + A(k, j, i) * B(k, j, i);
c(   j, i) = select(k == K - 1, C(k, j, i));
```

UREs

Output

```
A.merge_ures(B, C, c)
 .set_bounds(k, 0, K, j, 0, J, i, 0, I);
```

Put UREs into the same loop nest.

T2S specification

```
#define I    1024
#define J    1024
#define K    256
#define TYPE Float(32)
```

Parameters

```
ImageParam a("a", TYPE, 2), b("b", TYPE, 2);
Var  k("k"), j("j"), i("i");
Func A("A", TYPE, {k, j, i}, Place::Device),
     B("B", TYPE, {k, j, i}, Place::Device),
     C("C", TYPE, {k, j, i}, Place::Device),
     c("c", Place::Device);
```

Declare inputs, loop variables, UREs

```
A(k, j, i) = select(j == 0, a(k, i), A(k, j - 1, i));
B(k, j, i) = select(i == 0, b(j, k), B(k, j, i - 1));
C(k, j, i) = select(k == 0, 0, C(k - 1, j, i)) + A(k, j, i) * B(k, j, i);
```

UREs

```
c(   j, i) = select(k == K - 1, C(k, j, i));
```

Output

```
A.merge_ures(B, C, c)
 .set_bounds(k, 0, K, j, 0, J, i, 0, I);
```

Put UREs into the same loop nest.

T2S specification (Cont.)

```
Buffer<float> ina(K, I), inb(J, K);
Initialize ina, inb (details skipped)
a.set(ina);
b.set(inb);
```

Set input data

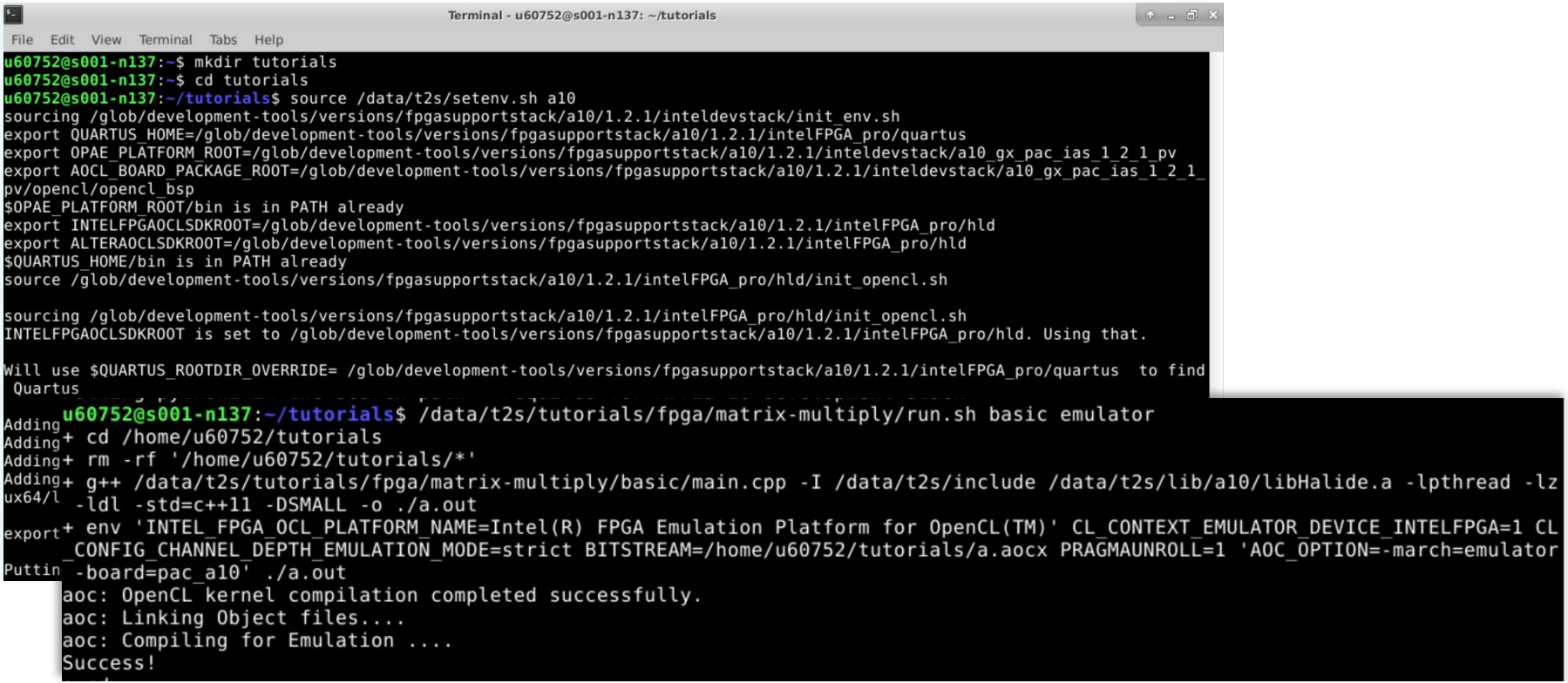
```
Target target = get_host_target();
target.set_feature(Target::IntelFPGA);
```

Get host CPU with an FPGA device

```
Buffer<float> result(J, I);
c.realize(result, target);
result.copy_to_host();
```

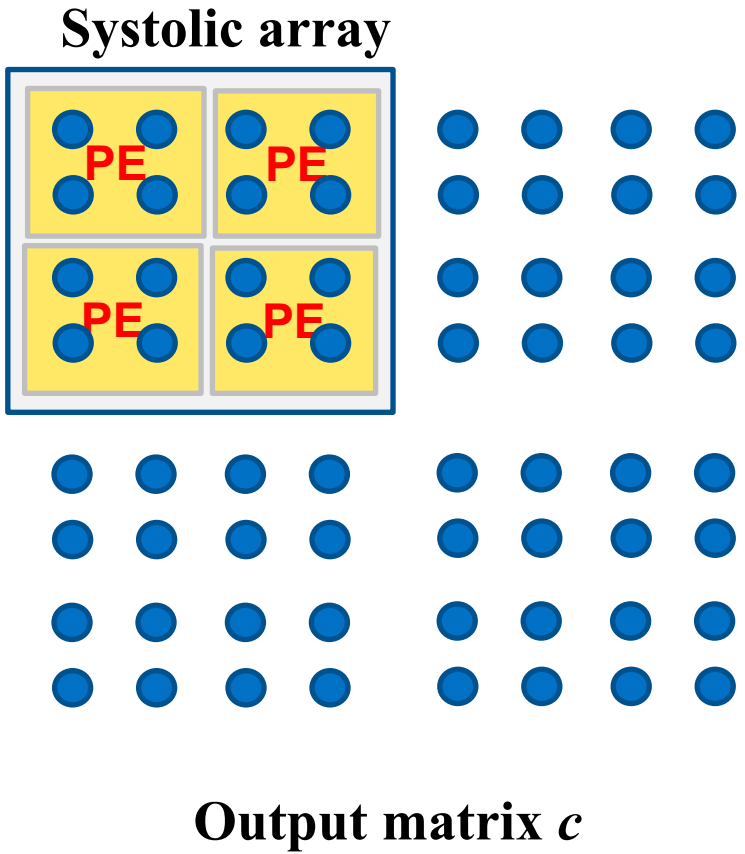
Compute the output. Copy to host.

Run it


Terminal - u60752@s001-n137: ~/tutorials
File Edit View Terminal Tabs Help
u60752@s001-n137:~$ mkdir tutorials
u60752@s001-n137:~$ cd tutorials
u60752@s001-n137:~/tutorials$ source /data/t2s/setenv.sh a10
sourcing /glob/development-tools/versions/fpgasupportstack/a10/1.2.1/inteldevstack/init_env.sh
export QUARTUS_HOME=/glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/quartus
export OPAE_PLATFORM_ROOT=/glob/development-tools/versions/fpgasupportstack/a10/1.2.1/inteldevstack/a10_gx_pac_ias_1_2_1_pv
export AOCL_BOARD_PACKAGE_ROOT=/glob/development-tools/versions/fpgasupportstack/a10/1.2.1/inteldevstack/a10_gx_pac_ias_1_2_1_pv/opencl/opencl_bsp
$OPAE_PLATFORM_ROOT/bin is in PATH already
export INTELFPGAOCLSDKROOT=/glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/hld
export ALTERAOCLSDKROOT=/glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/hld
$QUARTUS_HOME/bin is in PATH already
source /glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/hld/init_opencl.sh
sourcing /glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/hld/init_opencl.sh
INTELFPGAOCLSDKROOT is set to /glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/hld. Using that.
Will use $QUARTUS_ROOTDIR_OVERRIDE= /glob/development-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/quartus to find Quartus
u60752@s001-n137:~/tutorials$ /data/t2s/tutorials/fpga/matrix-multiply/run.sh basic emulator
Adding+ cd /home/u60752/tutorials
Adding+ rm -rf '/home/u60752/tutorials/*'
Adding+ g++ /data/t2s/tutorials/fpga/matrix-multiply/basic/main.cpp -I /data/t2s/include /data/t2s/lib/a10/libHalide.a -lpthread -lz
ux64/l -ldl -std=c++11 -DSMALL -o ./a.out
export+ env 'INTEL_FPGA_OCL_PLATFORM_NAME=Intel(R) FPGA Emulation Platform for OpenCL(TM)' CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1 CL_CONFIG_CHANNEL_DEPTH_EMULATION_MODE=strict BITSTREAM=/home/u60752/tutorials/a.aocx PRAGMAUNROLL=1 'AOC_OPTION=-march=emulator
Puttin -board=pac_a10' ./a.out
aoc: OpenCL kernel compilation completed successfully.
aoc: Linking Object files....
aoc: Compiling for Emulation ....
Success!

Tiling

- Matrices' sizes can be flexible
- Partition the output matrix into tiles
- Compute tile by tile use a systolic array



T2S specification

```
#define II   4
#define JJ   4
#define KK   256
#define III  2
#define JJJ  4
#define KKK  4
#define TYPE Float(32)
```

Parameters

```
#define I    (a.dim(1).extent() / (III * II))
#define J    (b.dim(0).extent() / (JJJ * JJ))
#define K    (a.dim(0).extent() / (KKK * KK))
```

Outermost loops' extents now determined by the inputs' sizes!

```
#define P             kkk,          jjj,     iii,     kk,          jj, ii, k,     j, i
#define P_iii_minus_1 kkk,          jjj,     iii - 1, kk,          jj, ii, k,     j, i
#define P_jjj_minus_1 kkk,          jjj - 1, iii,     kk,          jj, ii, k,     j, i
#define P_kkk_minus_1 kkk - 1,      jjj,     iii,     kk,          jj, ii, k,     j, i
#define P_kk_minus_1  kkk + KKK - 1, jjj,    iii,     kk - 1,      jj, ii, k,     j, i
#define P_k_minus_1   kkk + KKK - 1, jjj,    iii,     kk + KK - 1, jj, ii, k - 1, j, I
#define P_c                         jjj,     iii,                  jj, ii,        j, i
```

Iteration s

```
#define total_i      (iii + III * ii + III * II * i)
#define total_j      (jjj + JJJ * jj + JJJ * JJ * j)
#define total_k      (kkk + KKK * kk + KKK * KK * k)
```

Linearized addresses for reading inputs

T2S specification (Cont.)

```
ImageParam a("a", TYPE, 2), b("b", TYPE, 2);
Var  kkk("kkk"), jjj("jjj"), iii("iii"), kk("kk"), jj("jj"), ii("ii"), k("k"), j("j"), i("i");
Func A("A", TYPE, {P}, Place::Device),
     B("B", TYPE, {P}, Place::Device),
     C("C", TYPE, {P}, Place::Device),
     c("c", Place::Device);
```

Declare inputs, loop vars and UREs

```
A(P) = select(jjj == 0, a(total_k, total_i), A(P_jjj_minus_1));
B(P) = select(iii == 0, b(total_j, total_k), B(P_iii_minus_1));
C(P) = select(kkk == 0 && kk == 0 && k == 0,
              0,
              select(kkk == 0,
                     select(kk == 0, C(P_k_minus_1), C(P_kk_minus_1)),
                     C(P_kkk_minus_1)
              )
       ) + A(P) * B(P);
c(P_c) = select((kkk == KKK - 1) && (kk == KK -1) && (k == K - 1), C(P));
```

UREs

```
A.merge_ures(B, C, c);
 .set_bounds(kkk, 0, KKK, jjj, 0, JJJ, iii, 0, III)
 .set_bounds(kk,  0, KK,  jj,  0, JJ,  ii,  0, II)
 .set_bounds(k,   0, K,   j,   0, J,   i,   0, I);
```

Put UREs into the same loop nest.
Set bounds of the loops

Issues

```
#define __address_space__A __global
#define __address_space__B __global
#define __address_space__C __global ...
__kernel void kernel_c_WAIT_FINISH_(...
 __address_space__A float *restrict _A,
 __address_space__B float *restrict _B,
 __address_space__C float *restrict _C, ...){
for (int _A_s0_i = 0; _A_s0_i < 0 + _0; _A_s0_i++) { …
 for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++) { …
  for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++) { …
   for (int _A_s0_ii_jj_kk_iii_jjj_kkk = 0; _A_s0_ii_jj_kk_iii_jjj_kkk < 0 +
       131072; _A_s0_ii_jj_kk_iii_jjj_kkk++){
...     float _37 = _A[_36];
...     _A[_47] = _38;
...     float _83 = _B[_82];
...     float _116 = _C[_115];
..      float _118 = _C[_117];
...     float _121 = _C[_120];
...     float _124 = _A[_107];
        float _125 = _B[_107];
...     _C[_136] = _127;...
}
```

Intermediate results are allocated space in global memory

Sequential loops. No parallelism.

Access global memory for every intermediate result

Issues

- Very inefficient using global memory for intermediate results of function A, B, and C
- No optimization of memory sizes
  - Each Func is allocated a space of size KKK * JJJ * III * KK * JJ * II * K * J * I , i. e. the product of the extents of all the loops
  - When the input sizes are big, these intermediate results can waste a huge amount of

```
u60752@s001-n137:~/tutorials$ /data/t2s/tutorials/fpga/matrix-multiply/run.sh tiling small emulator
CL: halide_opencl_device_malloc failed: 68719476736 bytes are requested to allocate on the device. The size exceeds 2^32 - 1.
```

Space-time transform and vectorization

```
A.space_time_transform(kkk, jjj, iii)
 .vectorize(kkk);
```

- Fully unroll loop jjj and iii. Every iteration turns into a hardware PE.
  - PEs execute in parallel, subject only to the dependences between them
- Vectorize loop kkk
  - Enables data parallelism: KKK number of data from matrix a and b will be loaded together every cycle
- Allocate minimal shift registers for intermediate results.

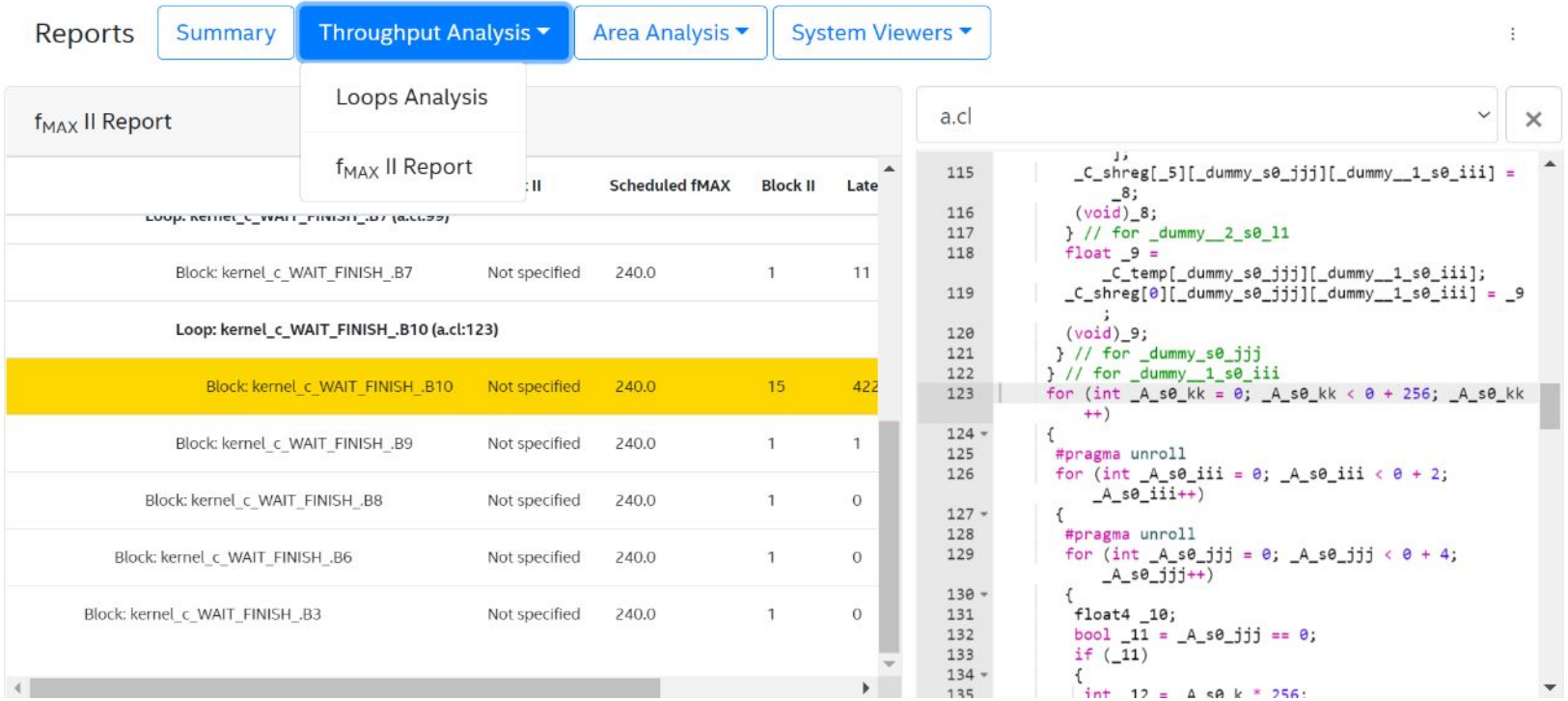
Generated code looks like…

```
__kernel void kernel_c_WAIT_FINISH_(...) {
  float  _C_shreg[16][4][2];
  float4 _B_shreg[4][2];
  float4 _A_shreg[4][2];
```

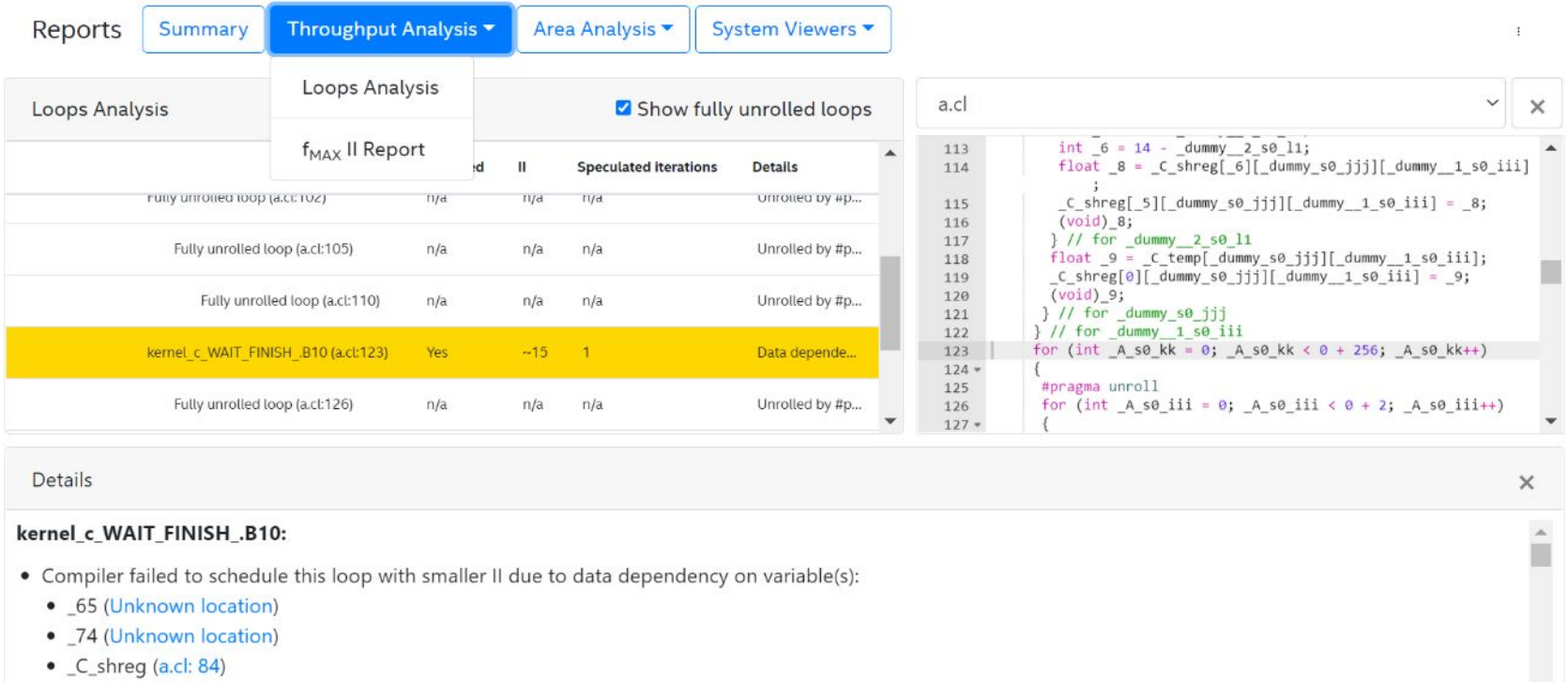
Constant size. Not related with the (dynami
extents of the outermost loops

4 values will be loaded together from
matrix a and b, repsectively

Static estimate of performance: fMax II report



Static estimate of performance: Loop analysis



Look at the generated code

```
__kernel void kernel_c_WAIT_FINISH_(...){
 float _C_shreg[16][4][2]; ...
  for (int _A_s0_i = 0; _A_s0_i < 0 + _0; _A_s0_i++) { ...
   for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++){ ...
    for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++)  { ...
     for (int _A_s0_ii_jj = 0; _A_s0_ii_jj < 0 + 16; _A_s0_ii_jj++) {...
      #pragma unroll for (int _dummy__1_s0_iii=0;_dummy__1_s0_iii < 0 + 2; _dummy__1_s0_iii++){ ...   #pragma unroll for (int
_dummy_s0_jjj = 0; _dummy_s0_jjj < 0 + 4; _dummy_s0_jjj++) {
       float _4 = _C_shreg[15][_dummy_s0_jjj][_dummy__1_s0_iii];
       _C_temp[_dummy_s0_jjj][_dummy__1_s0_iii] = _4;
       #pragma unroll for (int _dummy__2_s0_l1=0;_dummy__2_s0_l1 < 0 + 15; _dummy__2_s0_l1++){
        int _5 = 15 - _dummy__2_s0_l1;
        int _6 = 14 - _dummy__2_s0_l1;
        float _8 = _C_shreg[_6][_dummy_s0_jjj][_dummy__1_s0_iii];
        _C_shreg[_5][_dummy_s0_jjj][_dummy__1_s0_iii] = _8; }
       float _9 = _C_temp[_dummy_s0_jjj][_dummy__1_s0_iii];
       _C_shreg[0][_dummy_s0_jjj][_dummy__1_s0_iii] = _9;
     }}
     for (int _A_s0_kk = 0; _A_s0_kk < 0 + 256; _A_s0_kk++){
      #pragma unroll for (int _A_s0_iii = 0; _A_s0_iii < 0 + 2; _A_s0_iii++){
       #pragma unroll for (int _A_s0_jjj = 0; _A_s0_jjj < 0 + 4; _A_s0_jjj++){
...      _C_shreg[0][_A_s0_jjj][_A_s0_iii] = _65;
...
         float _74 = _C_shreg[0][_A_s0_jjj][_A_s0_iii];
```

C is allocated shift registers, and its size is constant ✔

Rotate the shift registers of C in each PE ✔

Dependence cycles across kk iterations ✗

Reordering

```
#define P                kkk,          jjj,     iii,
#define P_iii_minus_1 kkk,          jjj,     iii - 1,
#define P_jjj_minus_1 kkk,          jjj - 1, iii,
#define P_kkk_minus_1 kkk - 1,      jjj,     iii,
#define P_kk_minus_1  kkk + KKK - 1, jjj,    iii,
#define P_k_minus_1   kkk + KKK - 1, jjj,    iii,
```

```
kk,          jj, ii,    k,         j, i
kk,          jj, ii,    k,         j, i
kk,          jj, ii,    k,         j, i
kk,          jj, ii,    k,         j, i
kk - 1,      jj, ii,    k,         j, i
Kk + KK - 1, jj, ii,    k - 1,     j, i
```

```
__kernel void kernel_c_WAIT_FINISH_(...) { ...
float _C_shreg[16][4][2]; ...
 for (int _A_s0_i = 0; _A_s0_i < 0 + _0; _A_s0_i++){ ...
  for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++){ ...
   for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++){ ...
    for (int _A_s0_kk_ii_jj = 0; _A_s0_kk_ii_jj < 0 + 4096; _A_s0_kk_ii_jj++){ ...
     #pragma unroll for (int _dummy__1_s0_iii = 0; _dummy__1_s0_iii < 0 + 2; _dummy__1_s0_iii++){
     #pragma unroll for (int _dummy_s0_jjj = 0; _dummy_s0_jjj < 0 + 4; _dummy_s0_jjj++){
      float _4 = _C_shreg[15][_dummy_s0_jjj][_dummy__1_s0_iii];
      _C_temp[_dummy_s0_jjj][_dummy__1_s0_iii] = _4;
      #pragma unroll for (int _dummy__2_s0_l1 = 0; _dummy__2_s0_l1 < 0 + 15; _dummy__2_s0_l1++){
        int _5 = 15 - _dummy__2_s0_l1; int _6 = 14 - _dummy__2_s0_l1;
        float _8 = _C_shreg[_6][_dummy_s0_jjj][_dummy__1_s0_iii];
        _C_shreg[_5][_dummy_s0_jjj][_dummy__1_s0_iii] = _8;}
      float _9 = _C_temp[_dummy_s0_jjj][_dummy__1_s0_iii];
      _C_shreg[0][_dummy_s0_jjj][_dummy__1_s0_iii] = _9;}}
     #pragma unroll for (int _A_s0_iii = 0; _A_s0_iii < 0 + 2; _A_s0_iii++){
      #pragma unroll for (int _A_s0_jjj = 0; _A_s0_jjj < 0 + 4; _A_s0_jjj++){
...       _C_shreg[0][_A_s0_jjj][_A_s0_iii] = _69;
...       #pragma unroll for (int _A_s0_kkk = 0; _A_s0_kkk < 0 + 4; _A_s0_kkk++){
...         if (...) {
              float _79 = _C_shreg[0][_A_s0_jjj][_A_s0_iii];
...           _c[_103] = _79;
```

Loop kk moved outside of jj and ii (actually flattened with them)

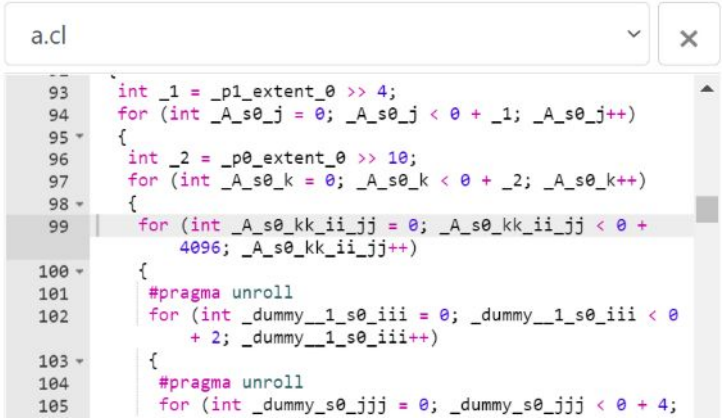
Rotate the shift regs of C in each PE

Dependence cycles not crossing kk loop, since registers rotated before the accesses

Static estimate of performance: fMax II report

f_{MAX} II Report

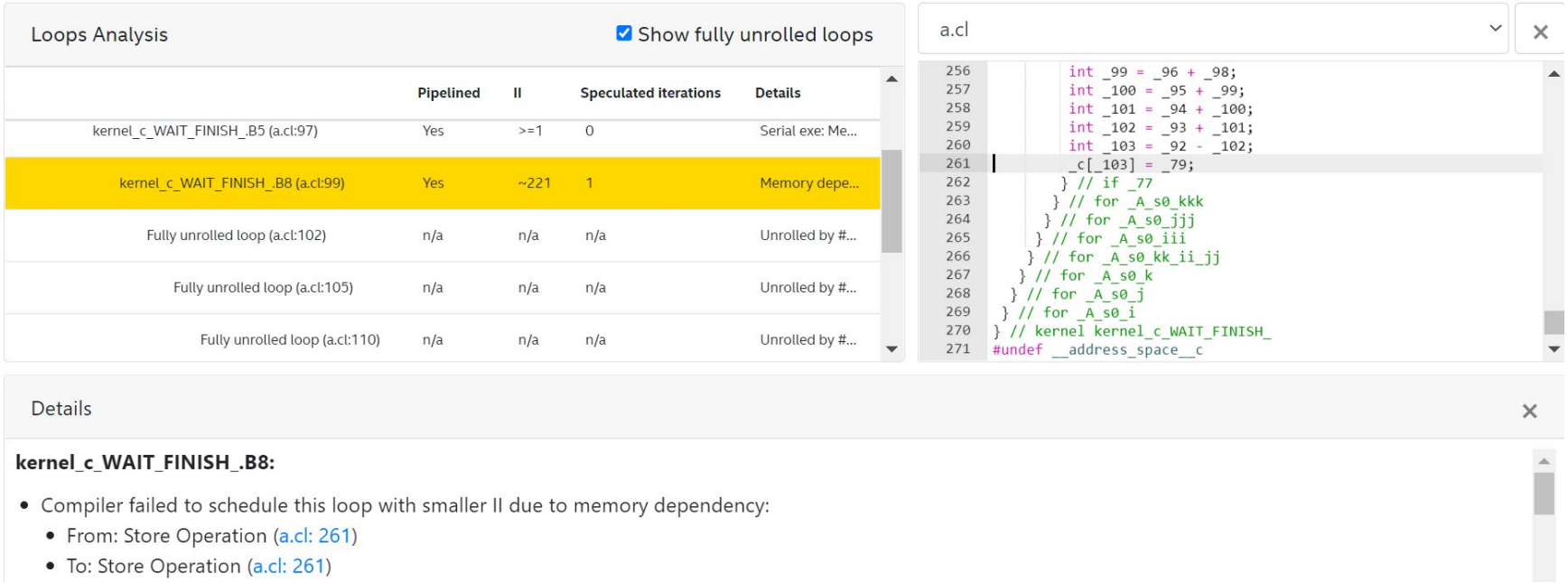
	Target II	Scheduled fMAX	Block II	Latency
Loop: kernel_c_WAIT_FINISH_.B8 (a.cl:99)				
Block: kernel_c_WAIT_FINISH_.B8	Not specified	240.0	221	461
Block: kernel_c_WAIT_FINISH_.B7	Not specified	240.0	1	0
Block: kernel_c_WAIT_FINISH_.B6	Not specified	240.0	1	0



a.cl

```
93   int _1 = _p1_extent_0 >> 4;
94   for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++)
95   {
96    int _2 = _p0_extent_0 >> 10;
97    for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++)
98    {
99     for (int _A_s0_kk_ii_jj = 0; _A_s0_kk_ii_jj < 0 +
          4096; _A_s0_kk_ii_jj++)
100    {
101     #pragma unroll
102     for (int _dummy__1_s0_iii = 0; _dummy__1_s0_iii < 0
          + 2; _dummy__1_s0_iii++)
103     {
104      #pragma unroll
105      for (int _dummy_s0_jjj = 0; _dummy_s0_jjj < 0 + 4;
```

Static estimate of performance: Loop analysis



Look at the generated code again

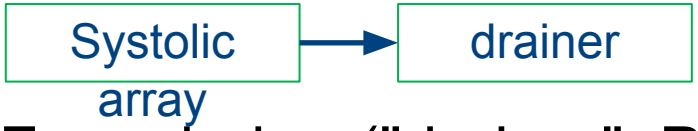
```
__kernel void kernel_c_WAIT_FINISH_(...) { ...
float _C_shreg[16][4][2]; ...
 for (int _A_s0_i = 0; _A_s0_i < 0 + _0; _A_s0_i++){ ...
  for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++){ ...
   for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++){ ...
    for (int _A_s0_kk_ii_jj = 0; _A_s0_kk_ii_jj < 0 + 4096; _A_s0_kk_ii_jj++){ ...
     #pragma unroll for (int _dummy__1_s0_iii = 0; _dummy__1_s0_iii < 0 + 2; _dummy__1_s0_iii++){
      #pragma unroll for (int _dummy_s0_jjj = 0; _dummy_s0_jjj < 0 + 4; _dummy_s0_jjj++){
       float _4 = _C_shreg[15][_dummy_s0_jjj][_dummy__1_s0_iii];
       _C_temp[_dummy_s0_jjj][_dummy__1_s0_iii] = _4;
       #pragma unroll for (int _dummy__2_s0_l1 = 0; _dummy__2_s0_l1 < 0 + 15; _dummy__2_s0_l1++){
          int _5 = 15 - _dummy__2_s0_l1; int _6 = 14 - _dummy__2_s0_l1;
          float _8 = _C_shreg
          _C_shreg[_5][_dur
       float _9 = _C_temp[_
       _C_shreg[0][_dummy
     #pragma unroll for (int
     #pragma unroll for (in
...        _C_shreg[0][_A_s
...         #pragma unroll for
...          if (...) {
              float _79 =
...             _c[_103] = _79;
```

The code corresponds to this line of the specification:

c(P_c)=select((kkk==KKK-1)&&(kk==KK-1)&&
(k==K-1),C(P))

A write happens only when a reduction is done. But the OpenCL compiler seems to be conservative, and assume a write happens every _A_s0_kk_ii_jj iteration. That is why there is a write-write dependence cycle across the loop.

Isolating the output




```
Func drainer("drainer", Place::Device);
c.isolate_consumer(drainer);
drainer.space_time_transform(jjj, iii);
```

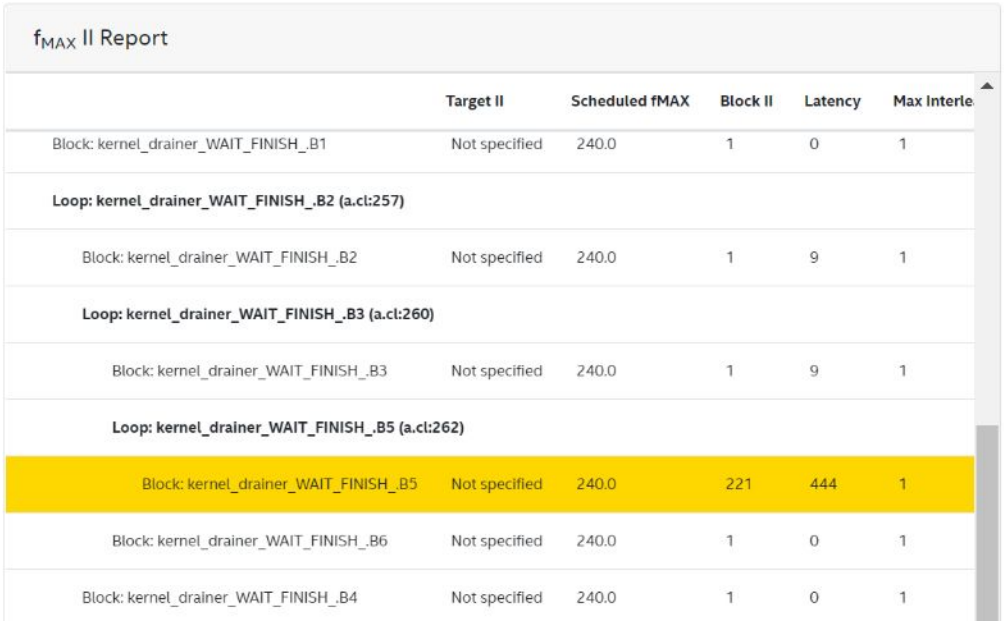
f_{MAX} II Report

	Target II	Scheduled fMAX	Block II	Latency	Max Interle
Loop: kernel_c.B7 (a.cl:87)					
Block: kernel_c.B7	Not specified	240.0	1	187	1
Block: kernel_c.B8	Not specified	240.0	1	0	1
Block: kernel_c.B6	Not specified	240.0	1	0	1
Block: kernel_c.B3	Not specified	240.0	1	0	1

a.cl

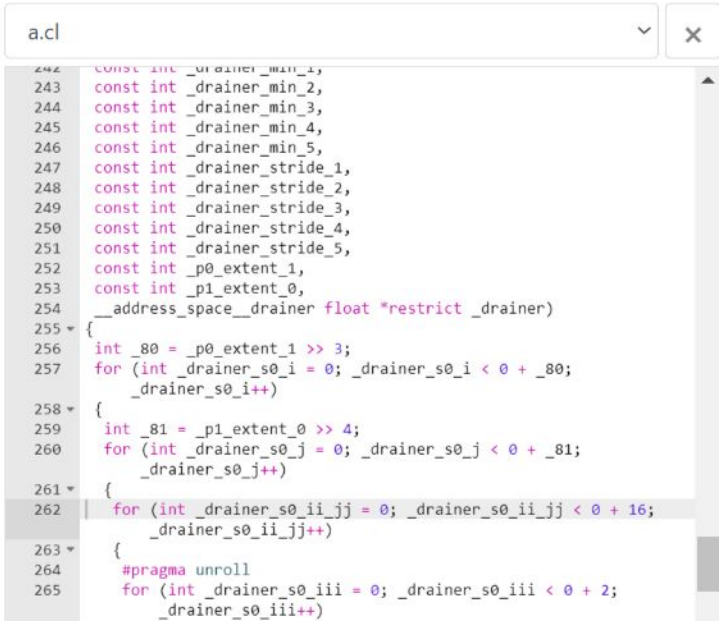
```
80  {
81   int _1 = _p1_extent_0 >> 4;
82   for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++)
83   {
84    int _2 = _p0_extent_0 >> 10;
85    for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++)
86    {
87     for (int _A_s0_kk_ii_jj = 0; _A_s0_kk_ii_jj < 0 + 4096;
           _A_s0_kk_ii_jj++)
88     {
89      #pragma unroll
90      for (int _dummy__1_s0_iii = 0; _dummy__1_s0_iii < 0 + 2;
           _dummy__1_s0_iii++)
91      {
92       #pragma unroll
93       for (int _dummy_s0_jjj = 0; _dummy_s0_jjj < 0 + 4;
           _dummy_s0_jjj++)
```

Bad II in the drainer now



f_{MAX} II Report

	Target II	Scheduled fMAX	Block II	Latency	Max Interle
Block: kernel_drainer_WAIT_FINISH_.B1	Not specified	240.0	1	0	1
Loop: kernel_drainer_WAIT_FINISH_.B2 (a.cl:257)					
Block: kernel_drainer_WAIT_FINISH_.B2	Not specified	240.0	1	9	1
Loop: kernel_drainer_WAIT_FINISH_.B3 (a.cl:260)					
Block: kernel_drainer_WAIT_FINISH_.B3	Not specified	240.0	1	9	1
Loop: kernel_drainer_WAIT_FINISH_.B5 (a.cl:262)					
Block: kernel_drainer_WAIT_FINISH_.B5	Not specified	240.0	221	444	1
Block: kernel_drainer_WAIT_FINISH_.B6	Not specified	240.0	1	0	1
Block: kernel_drainer_WAIT_FINISH_.B4	Not specified	240.0	1	0	1



a.cl

```
243  const int _drainer_min_2,
244  const int _drainer_min_3,
245  const int _drainer_min_4,
246  const int _drainer_min_5,
247  const int _drainer_stride_1,
248  const int _drainer_stride_2,
249  const int _drainer_stride_3,
250  const int _drainer_stride_4,
251  const int _drainer_stride_5,
252  const int _p0_extent_1,
253  const int _p1_extent_0,
254  __address_space__drainer float *restrict _drainer)
255 {
256  int _80 = _p0_extent_1 >> 3;
257  for (int _drainer_s0_i = 0; _drainer_s0_i < 0 + _80;
         _drainer_s0_i++)
258  {
259   int _81 = _p1_extent_0 >> 4;
260   for (int _drainer_s0_j = 0; _drainer_s0_j < 0 + _81;
          _drainer_s0_j++)
261   {
262    for (int _drainer_s0_ii_jj = 0; _drainer_s0_ii_jj < 0 + 16;
           _drainer_s0_ii_jj++)
263    {
264     #pragma unroll
265     for (int _drainer_s0_iii = 0; _drainer_s0_iii < 0 + 2;
            _drainer_s0_iii++)
```

Drainer: loop analysis

Loops Analysis — Show fully unrolled loops

	Pipelined	II	Speculated iterations	Details
kernel_drainer_WAIT_FINISH_.B2 (a.cl:257)	Yes	>=1	0	Serial exe: Me...
kernel_drainer_WAIT_FINISH_.B3 (a.cl:260)	Yes	>=1	0	Serial exe: Me...
kernel_drainer_WAIT_FINISH_.B5 (a.cl:262)	Yes	~221	1	Memory depe...
Fully unrolled loop (a.cl:265)	n/a	n/a	n/a	Unrolled by #...
Fully unrolled loop (a.cl:268)	n/a	n/a	n/a	Unrolled by #...

a.cl

```
286       int _98 = _drainer_min_2 * _drainer_stride_2;
287       int _99 = _drainer_min_1 * _drainer_stride_1;
288       int _100 = _99 + _drainer_min_0;
289       int _101 = _98 + _100;
290       int _102 = _97 + _101;
291       int _103 = _96 + _102;
292       int _104 = _95 + _103;
293       int _105 = _94 - _104;
294       _drainer[_105] = __82;
295     } // for _drainer_s0_jjj
296    } // for _drainer_s0_iii
297   } // for _drainer_s0_ii_jj
298  } // for _drainer_s0_j
299 } // for _drainer_s0_i
300 } // kernel kernel_drainer_WAIT_FINISH_
301 #undef __address_space__drainer
```

Details

kernel_drainer_WAIT_FINISH_.B5:

- Compiler failed to schedule this loop with smaller II due to memory dependency:
  - From: Store Operation (a.cl: 294)
  - To: Store Operation (a.cl: 294)

Look at the code

```
channel float _c_channel[2][4] __attribute__((depth(0))) ;
__kernel void kernel_c(...){...
 for (int _A_s0_i = 0; _A_s0_i < 0 + _0; _A_s0_i++){...
  for (int _A_s0_j = 0; _A_s0_j < 0 + _1; _A_s0_j++){...
   for (int _A_s0_k = 0; _A_s0_k < 0 + _2; _A_s0_k++){...
    for (int _A_s0_kk_ii_jj = 0; _A_s0_kk_ii_jj < 0 + 4096; _A_s0_kk_ii_jj++){...
        float _79 = _C_shreg[0][_A_s0_jjj][_A_s0_iii];
        write_channel_intel(_c_channel[_A_s0_iii][_A_s0_jjj], _79); ...
} // kernel kernel_c
```

The systolic array drains results through channels, instead of directly writing memory ✔

```
__kernel void kernel_drainer_WAIT_FINISH_(...){...
 for (int _drainer_s0_i = 0; _drainer_s0_i < 0 + _80; _drainer_s0_i++){...
  for (int _drainer_s0_j = 0; _drainer_s0_j < 0 + _81; _drainer_s0_j++){...
   for (int _drainer_s0_ii_jj = 0; _drainer_s0_ii_jj < 0 + 16; _drainer_s0_ii_jj++){
    #pragma unroll for (int _drainer_s0_iii = 0; _drainer_s0_iii < 0 + 2; _drainer_s0_iii++){
     #pragma unroll for (int _drainer_s0_jjj = 0; _drainer_s0_jjj < 0 + 4; _drainer_s0_jjj++){
      float __82 = read_channel_intel(_c_channel[_drainer_s0_iii][_drainer_s0_jjj]);
       ...
```

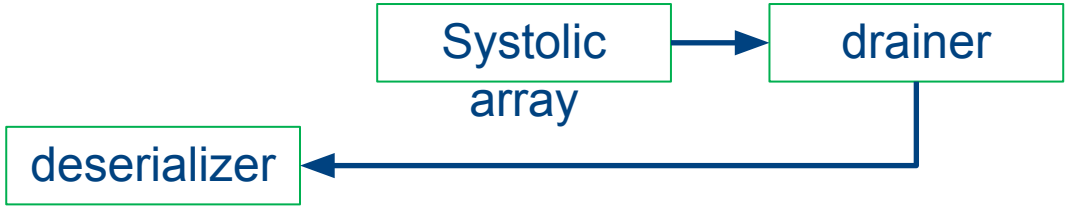
Look at the code (Cont.)

```
__kernel void kernel_drainer_WAIT_FINISH_(...){...
 for (int _drainer_s0_i = 0; _drainer_s0_i < 0 + _80; _drainer_s0_i++){...
  for (int _drainer_s0_j = 0; _drainer_s0_j < 0 + _81; _drainer_s0_j++){...
   for (int _drainer_s0_ii_jj = 0; _drainer_s0_ii_jj < 0 + 16; _drainer_s0_ii_jj++){
   #pragma unroll for (int _drainer_s0_iii = 0; _drainer_s0_iii < 0 + 2; _drainer_s0_iii++){
    #pragma unroll for (int _drainer_s0_jjj = 0; _drainer_s0_jjj < 0 + 4; _drainer_s0_jjj++){
     float __82 = read_channel_intel(_c_channel[_drainer_s0_iii][_drainer_s0_jjj]);
     int _83 = _drainer_s0_i * _drainer_stride_5;
     int _84 = _drainer_s0_j * _drainer_stride_4;
     int _85 = _drainer_s0_ii_jj >> 2;  int _86 = _85 * _drainer_stride_3;
     int _87 = _drainer_s0_ii_jj & 3;   int _88 = _87 * _drainer_stride_2;
     int _89 = _drainer_s0_iii * _drainer_stride_1; int _90 = _89 + _drainer_s0_jjj;
     int _91 = _88 + _90; int _92 = _86 + _91;      int _93 = _84 + _92;
     int _94 = _83 + _93; int _95 = _drainer_min_5 * _drainer_stride_5;
     int _96 = _drainer_min_4 * _drainer_stride_4;
     int _97 = _drainer_min_3 * _drainer_stride_3;
     int _98 = _drainer_min_2 * _drainer_stride_2;
     int _99 = _drainer_min_1 * _drainer_stride_1;
     int _100 = _99 + _drainer_min_0; int _101 = _98 + _100; int _102 = _97 + _101;
     int _103 = _96 + _102; int _104 = _95 + _103; int _105 = _94 - _104;
     _drainer[_105] = __82;
```

Address generation

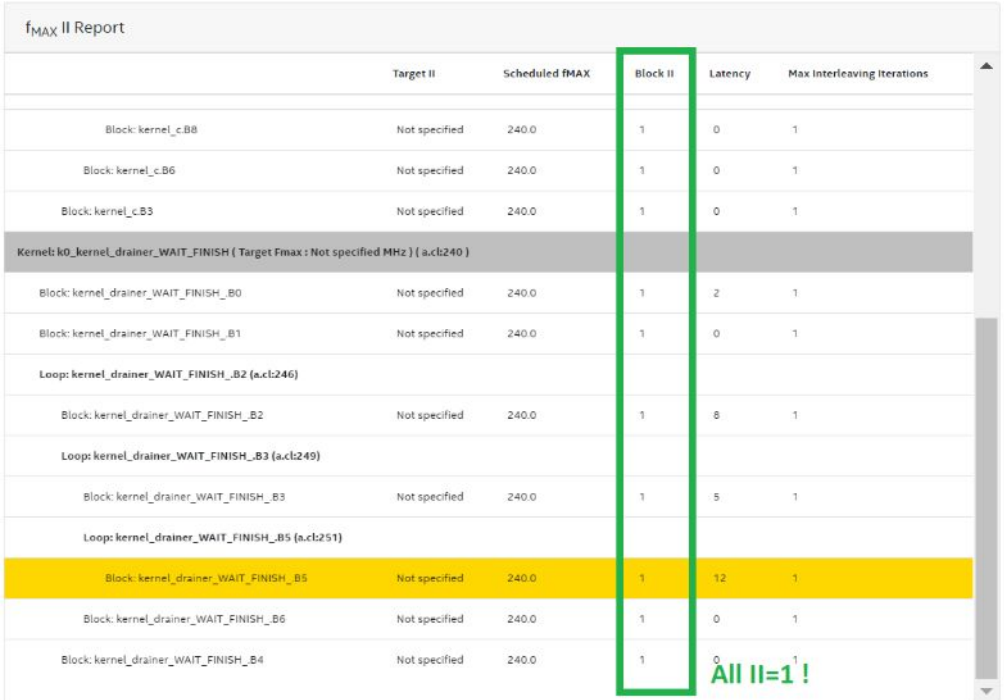
The complex address might have confused the OpenCL compiler, which then assumes dependency for safety

Isolating for serialization and de-serialization




```
Func drainer("drainer", Place::Device),
     deserializer("deserializer", Place::Host);
c.isolate_consumer(drainer);
drainer.space_time_transform(jjj, iii);
drainer.isolate_consumer(deserializer);
```

fMax II report



fMAX II Report

	Target II	Scheduled fMAX	Block II	Latency	Max Interleaving Iterations
Block: kernel_c.B8	Not specified	240.0	1	0	1
Block: kernel_c.B6	Not specified	240.0	1	0	1
Block: kernel_c.B3	Not specified	240.0	1	0	1
Kernel: k0_kernel_drainer_WAIT_FINISH (Target Fmax : Not specified MHz) (a.cl:240)					
Block: kernel_drainer_WAIT_FINISH_B0	Not specified	240.0	1	2	1
Block: kernel_drainer_WAIT_FINISH_B1	Not specified	240.0	1	0	1
Loop: kernel_drainer_WAIT_FINISH_B2 (a.cl:246)					
Block: kernel_drainer_WAIT_FINISH_B2	Not specified	240.0	1	8	1
Loop: kernel_drainer_WAIT_FINISH_B3 (a.cl:249)					
Block: kernel_drainer_WAIT_FINISH_B3	Not specified	240.0	1	5	1
Loop: kernel_drainer_WAIT_FINISH_B5 (a.cl:251)					
Block: kernel_drainer_WAIT_FINISH_B5	Not specified	240.0	1	12	1
Block: kernel_drainer_WAIT_FINISH_B6	Not specified	240.0	1	0	1
Block: kernel_drainer_WAIT_FINISH_B4	Not specified	240.0	1	0	1

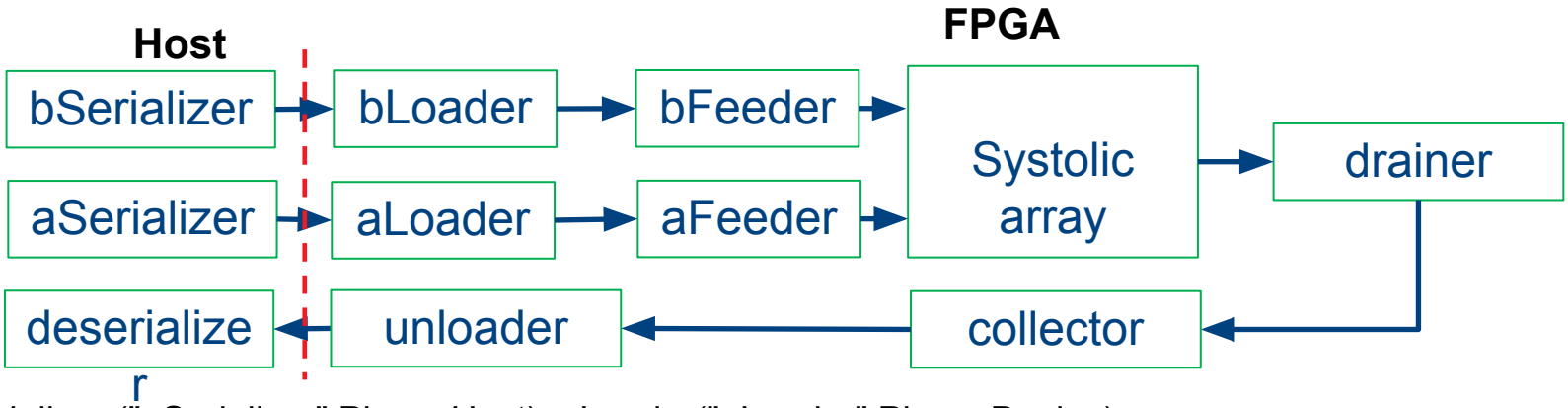
All II=1 !

a.cl

```
235 } // kernel kernel_c
236 #undef __address_space__p0
237 #undef __address_space__p1
238 // Address spaces for kernel_drainer_WAIT_FINISH_
239 #define __address_space__drainer __global
240 __kernel void kernel_drainer_WAIT_FINISH_(
241  const int _p0_extent_1,
242  const int _p1_extent_0,
243  __address_space__drainer float *restrict _drainer)
244 {
245  int _80 = _p0_extent_1 >> 3;
246  for (int _drainer_s0_i = 0; _drainer_s0_i < 0 + _80; _drainer_s0_i++)
247  {
248   int _81 = _p1_extent_0 >> 4;
249   for (int _drainer_s0_j = 0; _drainer_s0_j < 0 + _81; _drainer_s0_j++)
250   {
251    for (int _drainer_s0_ii_jj = 0; _drainer_s0_ii_jj < 0 + 16; _drainer_s0_ii_jj++)
252    {
253     #pragma unroll
254     for (int _drainer_s0_iii = 0; _drainer_s0_iii < 0 + 2; _drainer_s0_iii++)
255     {
256      #pragma unroll
257      for (int _drainer_s0_jjj = 0; _drainer_s0_jjj < 0 + 4; _drainer_s0_jjj++)
258      {
259       float __82 = read_channel_intel(c_channel[_drainer_s0_iii][_drainer_s0_jjj]);
260       int _83 = _p1_extent_0 >> 4;
261       int _84 = _83 * _drainer_s0_i;
262       int _85 = _84 * 128;
263       int _86 = _drainer_s0_j * 128;
264       int _87 = _drainer_s0_ii_jj >> 2;
265       int _88 = _87 * 32;
266       int _89 = _drainer_s0_ii_jj & 3;
267       int _90 = _89 * 8;
268       int _91 = _drainer_s0_iii * 4;
269       int _92 = _91 + _drainer_s0_jjj;
270       int _93 = _90 + _92;
271       int _94 = _88 + _93;
272       int _95 = _86 + _94;
273       int _96 = _85 + _95;
274       _drainer[_96] = __82;
275      } // for _drainer_s0_jjj
276     } // for _drainer_s0_iii
277    } // for _drainer_s0_ii_jj
278   } // for _drainer_s0_j
279  } // for _drainer_s0_i
280 } // kernel kernel_drainer_WAIT_FINISH_
281 #undef __address_space__drainer
282
```

Much simpler address generation but still could be further optimized

Isolating full I/O paths




```
Func aSerializer ("aSerializer",Place::Host), aLoader("aLoader",Place::Device),
    aFeeder("aFeeder", Place::Device), bSerializer("bSerializer",Place::Host),
    bLoader("bLoader", Place::Device), bFeeder("bFeeder", Place::Device),
    drainer("drainer", Place::Device), collector("collector", Place::Device),
    unloader("unloader", Place::Device),
    deserializer("deserializer",Place::Host);
A.isolate_producer_chain(a, aSerializer, aLoader, aFeeder);
A.isolate_producer_chain(b, bSerializer, bLoader, bFeeder);
c.isolate_consumer(drainer);
drainer.space_time_transform(jjj, iii);
drainer.isolate_consumer_chain(collector, unloader, deserializer);
```

fMax II report

Fmax II Report

	Target II	Scheduled Fmax	Block II	Latency	Max Interleaving Iterations
Kernel: kernel_aLoader_1 (Target Fmax : Not specified MHz) (/home/hrong1/tmp/a.cl:63)					
Block: kernel_aLoader_1.B0	Not specified	240.0	1	2	1
Block: kernel_aLoader_1.B1	Not specified	240.0	1	0	1
Loop: kernel_aLoader_1.B2 (/home/hrong1/tmp/a.cl:71)					
Block: kernel_aLoader_1.B2	Not specified	240.0	1	7	1
Loop: kernel_aLoader_1.B4 (/home/hrong1/tmp/a.cl:74)					
Block: kernel_aLoader_1.B4	Not specified	240.0	1	8	1
Loop: kernel_aLoader_1.B5 (/home/hrong1/tmp/a.cl:77)					
Block: kernel_aLoader_1.B5	Not specified	240.0	1	4	1
Loop: kernel_aLoader_1.B7 (/home/hrong1/tmp/a.cl:79)					
Block: kernel_aLoader_1.B7	Not specified	240.0	1	149	1
Block: kernel_aLoader_1.B8	Not specified	240.0	1	0	1
Block: kernel_aLoader_1.B6	Not specified	240.0	1	0	1
Block: kernel_aLoader_1.B3	Not specified	240.0	1	0	1
Kernel: kernel_aFeeder_1 (Target Fmax : Not specified MHz) (/home/hrong1/tmp/a.cl:123)					
Block: kernel_aFeeder_1.B0	Not specified	240.0	1	2	1
Block: kernel_aFeeder_1.B1	Not specified	240.0	1	0	1
Loop: kernel_aFeeder_1.B2 (/home/hrong1/tmp/a.cl:129)					

Fmax II Report

	Target II	Scheduled Fmax	Block II	Latency	Max Interleaving Iterations
Loop: kernel_aFeeder_1.B2 (/home/hrong1/tmp/a.cl:129)					
Block: kernel_aFeeder_1.B2	Not specified	240.0	1	4	1
Loop: kernel_aFeeder_1.B4 (/home/hrong1/tmp/a.cl:132)					
Block: kernel_aFeeder_1.B4	Not specified	240.0	1	4	1
Loop: kernel_aFeeder_1.B5 (/home/hrong1/tmp/a.cl:135)					
Block: kernel_aFeeder_1.B5	Not specified	240.0	1	4	1
Loop: kernel_aFeeder_1.B7 (/home/hrong1/tmp/a.cl:137)					
Block: kernel_aFeeder_1.B7	Not specified	240.0	1	4	1
Block: kernel_aFeeder_1.B8	Not specified	240.0	1	0	1
Block: kernel_aFeeder_1.B6	Not specified	240.0	1	0	1
Block: kernel_aFeeder_1.B3	Not specified	240.0	1	0	1
Kernel: kernel_bLoader_1 (Target Fmax : Not specified MHz) (/home/hrong1/tmp/a.cl:161)					
Block: kernel_bLoader_1.B0	Not specified	240.0	1	2	1
Block: kernel_bLoader_1.B1	Not specified	240.0	1	0	1
Loop: kernel_bLoader_1.B2 (/home/hrong1/tmp/a.cl:169)					
Block: kernel_bLoader_1.B2	Not specified	240.0	1	7	1
Loop: kernel_bLoader_1.B4 (/home/hrong1/tmp/a.cl:172)					
Block: kernel_bLoader_1.B4	Not specified	240.0	1	8	1

Fmax II Report

	Target II	Scheduled Fmax	Block II	Latency	Max Interleaving Iterations
Block: kernel_bLoader_1.B4	Not specified	240.0	1	8	1
Loop: kernel_bLoader_1.B5 (/home/hrong1/tmp/a.cl:175)					
Block: kernel_bLoader_1.B5	Not specified	240.0	1	4	1
Loop: kernel_bLoader_1.B7 (/home/hrong1/tmp/a.cl:177)					
Block: kernel_bLoader_1.B7	Not specified	240.0	1	134	1
Block: kernel_bLoader_1.B8	Not specified	240.0	1	0	1
Block: kernel_bLoader_1.B6	Not specified	240.0	1	0	1
Block: kernel_bLoader_1.B3	Not specified	240.0	1	0	1

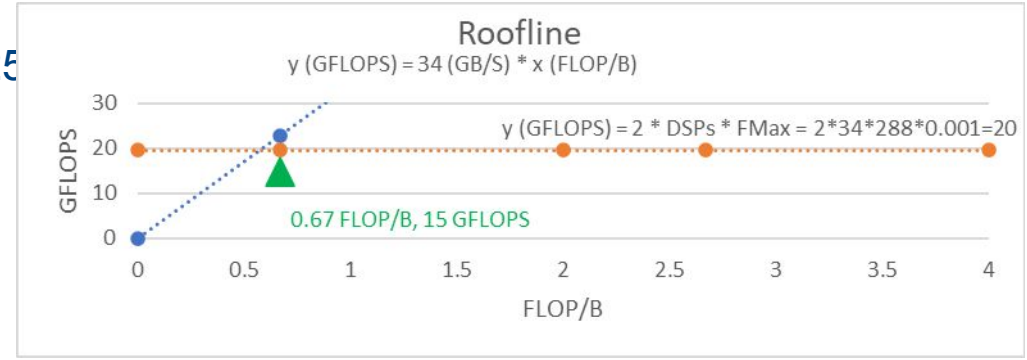
Fmax II Report

	Target II	Scheduled Fmax	Block II	Latency	Max Interleaving Iterations
Loop: kernel_bFeeder_1.B7 (/home/hrong1/tmp/a.cl:235)					
Block: kernel_bFeeder_1.B7	Not specified	240.0	1	5	1
Block: kernel_bFeeder_1.B8	Not specified	240.0	1	0	1
Block: kernel_bFeeder_1.B6	Not specified	240.0	1	0	1
Block: kernel_bFeeder_1.B3	Not specified	240.0	1	0	1
Kernel: kernel_c (Target Fmax : Not specified MHz) (/home/hrong1/tmp/a.cl:258)					
Block: kernel_c.B0	Not specified	240.0	1	2	1
Block: kernel_c.B1	Not specified	240.0	1	0	1

Dynamic profile (2*4 PEs, each vectorized by 4)

FPGA GEMM exec time = 2.325

operations = 34359738368
Throughput: 14.77792 GFLOPS



Board	pac_a10
Global Memory BW (DDR)	34133 MB/s

Source Code | Kernel Execution | kernel_bLoader | kernel_aLoader | kernel_collector | kernel_drainer | kernel_c | kernel_bFeeder | kernel_aFeeder

File Name	Directory
a.cl	/home/u60752/tutorials/a.cl

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
383	float __79 = read_channel_intel(_c_channel[_drainer_s0_iii][_drainer_s0_jjj]);	0: channel,read	0: 99.92%	0: 0.1%	0: 0.9MB/s
384	write_channel_intel(_drainer_channel[_drainer_s0_iii][_drainer_s0_iii], __79);	0: channel,w...	0: 0.0%	0: 0.1%	0: 0.9MB/s

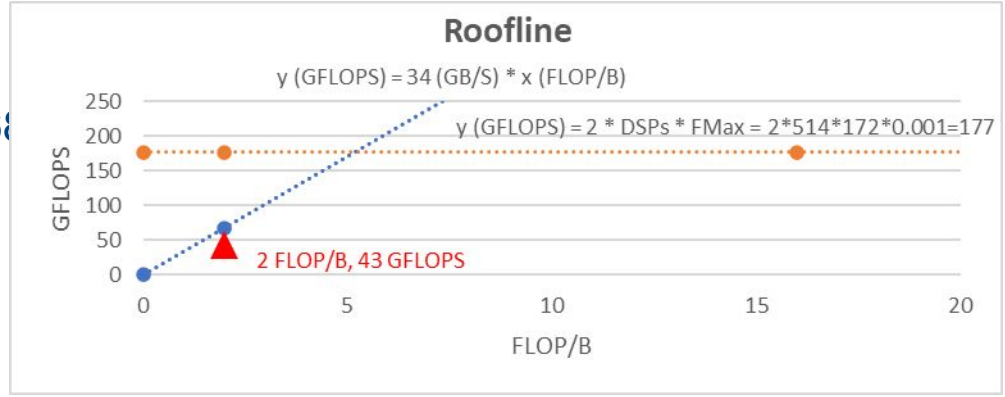
Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
409	for (int _collector_s0_jjj = 0; _collector_s0_jjj < 0 + 4; _collector_s0_jjj++)				
410	{				
411	float __82 = read_channel_intel(_drainer_channel[_collector_s0_iii][_collector_s0...	0: channel,read	0: 99.92%	0: 0.1%	0: 0.9MB/s
412	write_channel_intel(_collector_channel[_collector_s0_iii][_collector_s0_jjj], __82);	0: channel,w...	0: 0.0%	0: 0.1%	0: 0.9MB/s

Next target: move ▲ right, mov ●—● up

Scaling up to medium size (8 * 8 PEs, ech vectorized by 8)

FPGA GEMM exec time = 0.8068

operations = 34359738368
Throughput: 42.58627 GFLOPS



Board	pac_a10
Global Memory BW (DDR)	34133 MB/s

Source Code | Kernel Execution | kernel_bLoader | kernel_aLoader | kernel_collector | kernel_drainer | kernel_c | kernel_bFeeder | kernel_aFeeder

File Name	Directory
a.cl	/home/u60752/tutorials/isolate-all-8-8-16-32-32-32-8-8-8/a.cl

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
84	float8 _5 = vload8(0, (__address_space__aSerializer float*)_aSerializer + _4);	(__global{DD...	(94.1%)	(23.6%)	(10685.5MB/s...

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
213	float8 _51 = vload8(0, (__address_space__bSerializer float*)_bSerializer + _50);	(__global{DD...	(94.1%)	(23.6%)	(10685.7MB/s...

Memory bandwidth consumed by the loadings of the input matrices is totally about 22 GB/s

Stalls in reading from bLoader

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
290	float8 __84 = read_channel_intel(_bLoader_channel[0][0]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
291	write_channel_intel(_bFeeder_channel[0][0], __84);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
292	(void)__84;				
293	float8 __85 = read_channel_intel(_bLoader_channel[0][1]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
294	write_channel_intel(_bFeeder_channel[0][1], __85);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
295	(void)__85;				
296	float8 __86 = read_channel_intel(_bLoader_channel[0][2]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
297	write_channel_intel(_bFeeder_channel[0][2], __86);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
298	(void)__86;				
299	float8 __87 = read_channel_intel(_bLoader_channel[0][3]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
300	write_channel_intel(_bFeeder_channel[0][3], __87);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
301	(void)__87;				
302	float8 __88 = read_channel_intel(_bLoader_channel[0][4]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
303	write_channel_intel(_bFeeder_channel[0][4], __88);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
304	(void)__88;				
305	float8 __89 = read_channel_intel(_bLoader_channel[0][5]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
306	write_channel_intel(_bFeeder_channel[0][5], __89);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
307	(void)__89;				
308	float8 __90 = read_channel_intel(_bLoader_channel[0][6]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)
309	write_channel_intel(_bFeeder_channel[0][6], __90);	(channel,write)	(0.31%)	(23.6%)	(1335.8MB/s)
310	(void)__90;				
311	float8 __91 = read_channel_intel(_bLoader_channel[0][7]);	(channel,read)	(76.11%)	(23.6%)	(1335.8MB/s)

In short, the input paths become a bottleneck, which makes the design memory-bound.

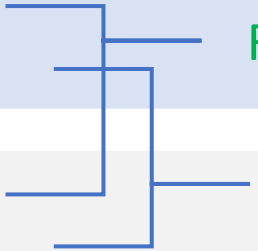
Optimize input paths for memory bandwidth

```
aSerializer.remove(jjj, jj, j);
bSerializer.remove(iii, ii, i);
```

Remove redundant host-device data transfer

```
aLoader.remove(jjj, jj);
aFeeder.buffer(aLoader, k);
```

```
bLoader.remove(iii, ii);
bFeeder.buffer(bLoader, k);
```



Remove reuse loops in loaders

Insert a buffer at a loop level that encloses all removed loops in a producer

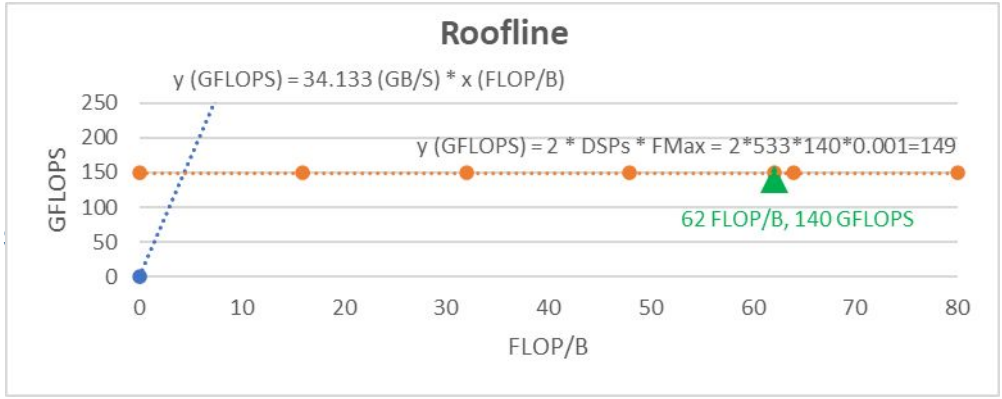
```
aFeeder.scatter(aLoader, iii);
bFeeder.scatter(bLoader, jjj);
```

Scatter data across consumer PEs

Dynamic profile

FPGA GEMM exec time = 0.24568

operations = 34359738368
Throughput: 139.85691 GFLOPS



Source Code | Kernel Execution | kernel_bLoader | kernel_aLoader | kernel_collector | kernel_drainer | kernel_c | kernel_bFeeder | kernel_aFeeder

Statistic	Measured	Optimal
Kernel Clock Frequency	176 MHz	na

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
108	float8 _29 = vload8(0, (__address_space__aSerializer float*) aSerializer + 28);	(__global{DD...	(0.0%)	(24.9%)	(1400.7MB/s,...
109	write_channel_intel(_aLoader_channel[0][0], _29);	(channel,write)	(74.66%)	(24.9%)	(1400.6MB/s)

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
834	float8 _667 = vload8(0, (__address_space__bSerializer float*) bSerializer + 666);	(__global{DD...	(0.0%)	(24.9%)	(1400.9MB/s,...
835	write_channel_intel(_bLoader_channel[0][0], _667);	(channel,write)	(74.66%)	(24.9%)	(1400.9MB/s)

Almost an order of magnitude saving of the memory bandwidth

- Memory bandwidth consumed by the loaders are about 2.8 GB/s instead 22GB/s

Still many stalls in the output paths

File Name	Directory
a.cl	/home/u60752/tutorials/new-buffer-scatter/a.cl

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
203609	float __137335 = read_channel_intel(_c_channel[0][0]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203610	write_channel_intel(_drainer_channel[0][0], __137335);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203611	(void)__137335;				
203612	float __137336 = read_channel_intel(_c_channel[0][1]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203613	write_channel_intel(_drainer_channel[0][1], __137336);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203614	(void)__137336;				
203615	float __137337 = read_channel_intel(_c_channel[0][2]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203616	write_channel_intel(_drainer_channel[0][2], __137337);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203617	(void)__137337;				
203618	float __137338 = read_channel_intel(_c_channel[0][3]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203619	write_channel_intel(_drainer_channel[0][3], __137338);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203620	(void)__137338;				
203621	float __137339 = read_channel_intel(_c_channel[0][4]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203622	write_channel_intel(_drainer_channel[0][4], __137339);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203623	(void)__137339;				
203624	float __137340 = read_channel_intel(_c_channel[0][5]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203625	write_channel_intel(_drainer_channel[0][5], __137340);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203626	(void)__137340;				
203627	float __137341 = read_channel_intel(_c_channel[0][6]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203628	write_channel_intel(_drainer_channel[0][6], __137341);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203629	(void)__137341;				
203630	float __137342 = read_channel_intel(_c_channel[0][7]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)
203631	write_channel_intel(_drainer_channel[0][7], __137342);	(channel,write)	(0.57%)	(0.2%)	(1.4MB/s)
203632	(void)__137342;				
203633	float __137343 = read_channel_intel(_c_channel[1][0]);	(channel,read)	(99.24%)	(0.2%)	(1.4MB/s)

128 output channels, all stalled most of the time.

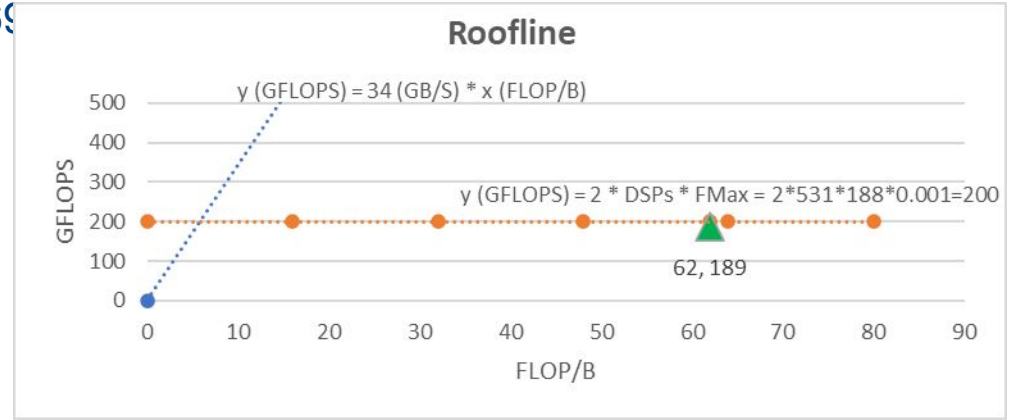
- 8 * 8 drainer PEs, communicating with 8 * 8 systolic array PEs directly
- 8 * 8 collector PEs, communicating with 8 * 8 drainer PEs directly

Simplifying the output paths

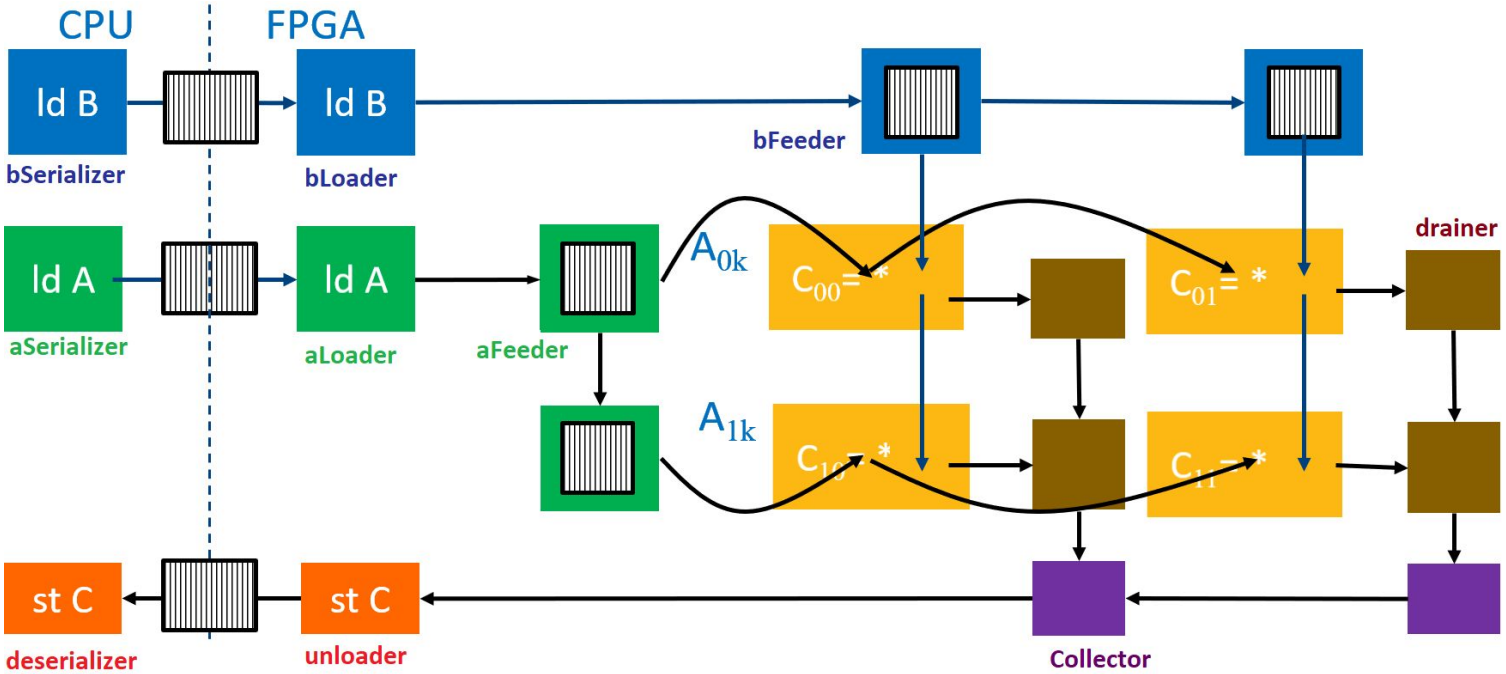
```
drainer.gather(c, iii);
collector.gather(drainer, jjj);
collector.vectorize(jjj);
unloader.vectorize(jjj);
```

FPGA GEMM exec time = 0.18169

operations = 34359738368
Throughput: 189.11542 GFLOPS



Now we have full I/O paths



Remaining bottlenecks

Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
204526	float __137660 = read_channel_intel(_drainer_channel[_137659]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204527	int _137661 = _137657 + 1;				
204528	int _137662 = (int)(_137661);				
204529	float __137663 = read_channel_intel(_drainer_channel[_137662]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204530	int _137664 = _137657 + 2;				
204531	int _137665 = (int)(_137664);				
204532	float __137666 = read_channel_intel(_drainer_channel[_137665]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204533	int _137667 = _137657 + 3;				
204534	int _137668 = (int)(_137667);				
204535	float __137669 = read_channel_intel(_drainer_channel[_137668]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204536	int _137670 = _137657 + 4;				
204537	int _137671 = (int)(_137670);				
204538	float __137672 = read_channel_intel(_drainer_channel[_137671]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204539	int _137673 = _137657 + 5;				
204540	int _137674 = (int)(_137673);				
204541	float __137675 = read_channel_intel(_drainer_channel[_137674]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204542	int _137676 = _137657 + 6;				
204543	int _137677 = (int)(_137676);				
204544	float __137678 = read_channel_intel(_drainer_channel[_137677]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)
204545	int _137679 = _137657 + 7;				
204546	int _137680 = (int)(_137679);				
204547	float __137681 = read_channel_intel(_drainer_channel[_137680]);	(channel,read)	(98.44%)	(1.6%)	(12.1MB/s)

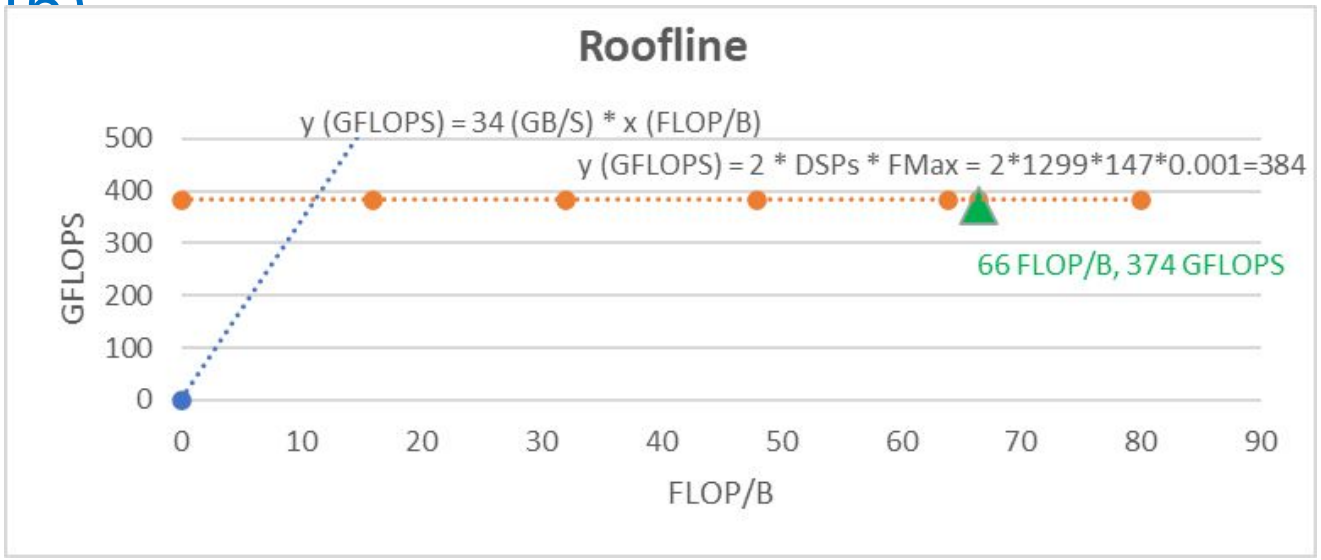
Line #	Source: a.cl	Attributes	Stall%	Occupancy%	Bandwidth
203614	float __137337 = read_channel_intel(_c_channel[0][0]);	(channel,read)	(98.44%)	(1.6%)	(1.6MB/s)
203615	_drainer_gather_c_shreg[0][0] = __137337;				
203616	(void)__137337;				
203617	} // if _137336				
203618	else				
203619	{				
203620	} // if _137336 else				
203621	int _137338 = _drainer_s0_ii_jj_iii & 7;				
203622	bool _137339 = _137338 == 0;				
203623	if (_137339)				
203624	{				
203625	float __137340 = read_channel_intel(_c_channel[0][1]);	(channel,read)	(98.44%)	(1.6%)	(1.6MB/s)
203626	_drainer_gather_c_shreg[0][1] = __137340;				
203627	(void)__137340;				
203628	} // if _137339				
203629	else				
203630	{				
203631	} // if _137339 else				
203632	int _137341 = _drainer_s0_ii_jj_iii & 7;				
203633	bool _137342 = _137341 == 0;				
203634	if (_137342)				
203635	{				
203636	float __137343 = read_channel_intel(_c_channel[0][2]);	(channel,read)	(98.44%)	(1.6%)	(1.6MB/s)
203637	_drainer_gather_c_shreg[0][2] = __137343;				
203638	(void)__137343;				

16 stalls

- 8 stalls in the drainer
- 8 stalls in the collector

Much less than before

Scaling up to a large array (10*8 PEs, each vectorized by 16)



LUTs	Registers	Logic	I/O pins	DSPs	Memory bits	BRAMs	fmax
188149	399,683	190,304/427,200 (45%)	310/826(38%)	1,299/1,518 (86%)	32,490,792 / 55,562,240 (58 %)	2,065/2,713 (76%)	147.73

Still working on

- Isolate out control signals to simplify the systolic array
- Further increase array size to 11 * 8 PEs, each vectorized by 16
- Add "-fpc -fp-relaxed" to the compilation flag for simpler logic.
- Add "-fmax=500” for possibly higher frequency.
- Add “-high-effort” to increase the chance of success in place and route.
- Seed sweeping.

Summary

- A tool for incremental, intuitive design space exploration
  - Guided by static profile, dynamic profile, and rooflines
  - Hosted on DevCloud, a free and well-maintained software and hardware environment for academics and researchers
  - We commit to continual updating and maintenance
- Productivity comes from telling the compiler what to do
- Performance comes from sophisticated implementation of the compiler
  - Still a valuable tool even eventually you implement your design in RTL
  - Help quickly eliminate potential bottlenecks in your design before spending time on RTL