Performance of Electronic Structure Calculations on Built-in FPGA Systems

Seungmin Lee

(E: smlee76@kisti.re.kr**)**

Intel® Parallel Computing Center
Korea Institute of Science and Technology Information (KISTI)

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

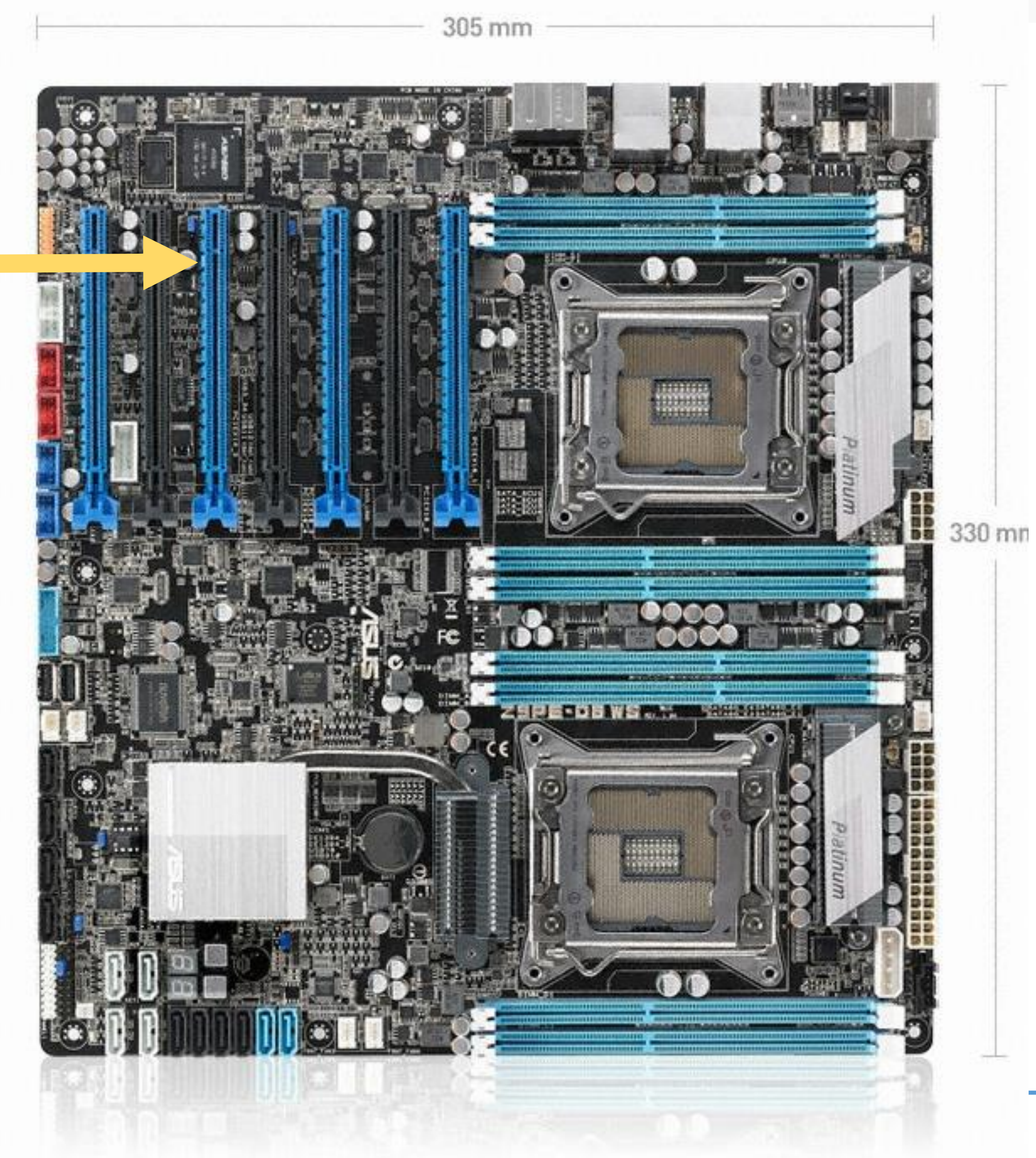
국가슈퍼컴퓨팅연구소 National Institute of Supercomputing and Networking

Ipcc KISTI

Contents

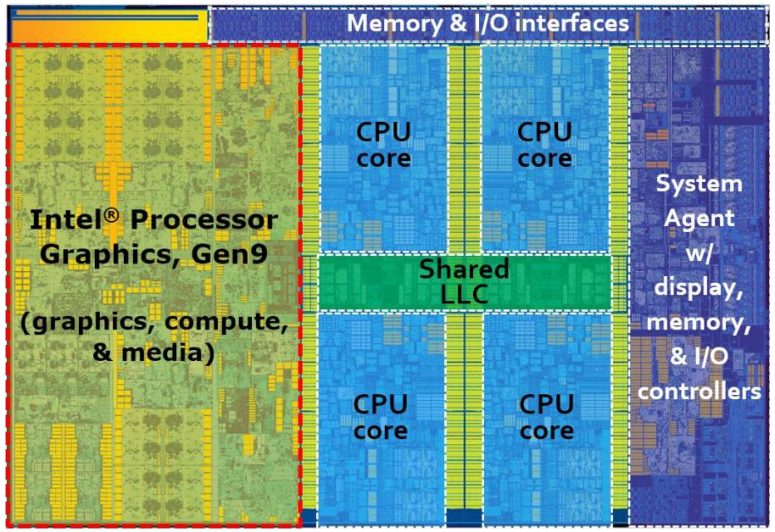
- Introduction
- Built-in FPGA System
- Methodology
- Results
- Conclusion

Accelerator/Co-processor Devices



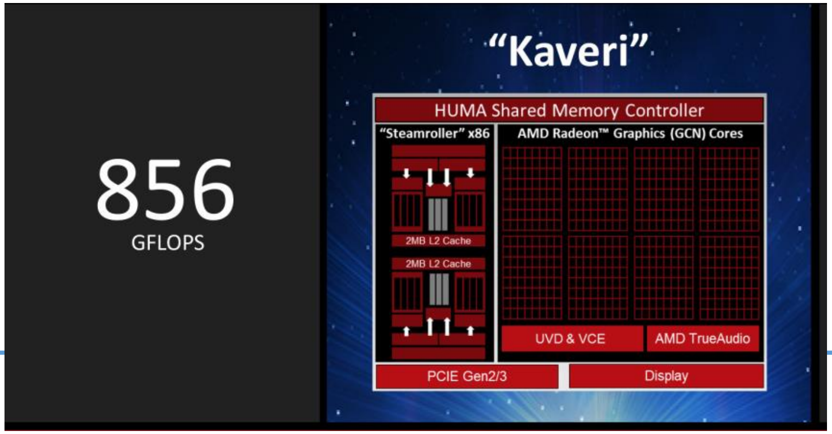
Processors with a Built-in Accelerator

Intel Skylake Gen9 GT4/e

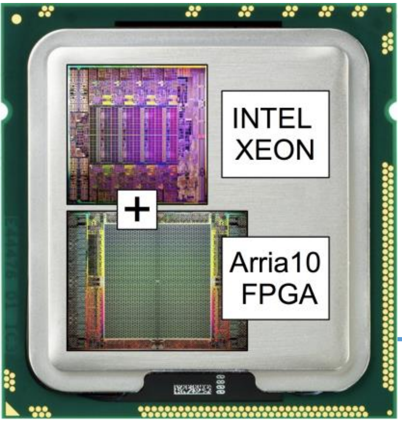

Memory & I/O interfaces
Intel® Processor Graphics, Gen9 (graphics, compute, & media)
CPU core
CPU core
Shared LLC
CPU core
CPU core
System Agent w/ display, memory, & I/O controllers

Tegra K1 (ARM Cortex-A15 with NVIDIA GPU)

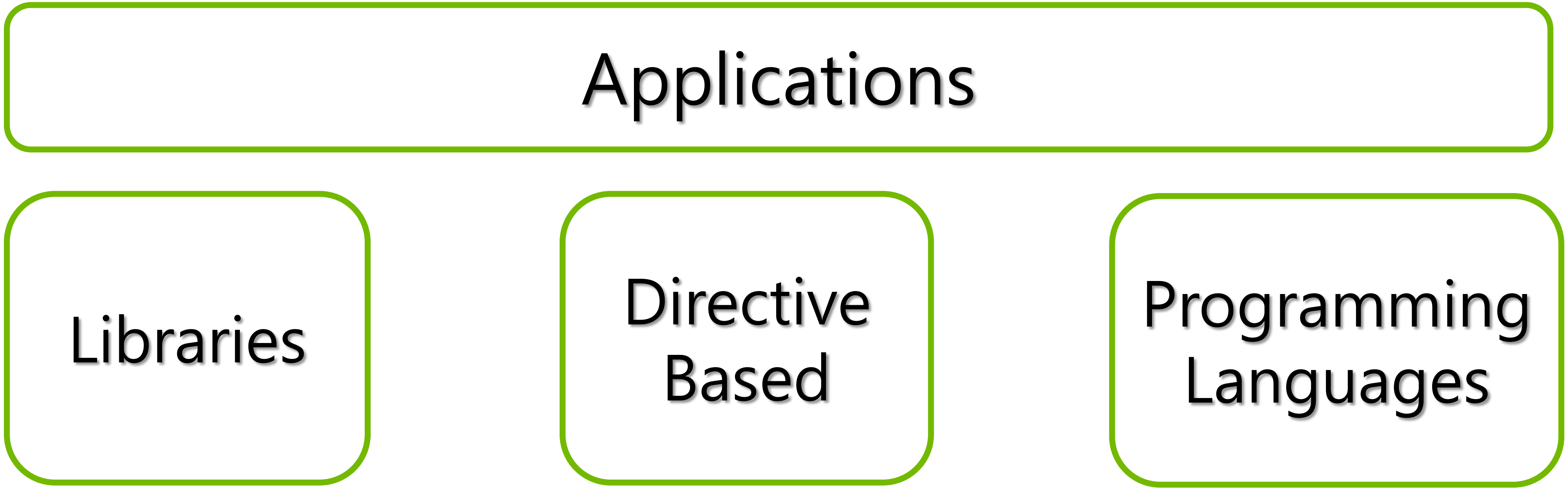
AMD APU Kaveri(2013)



Intel Broadwell Xeon with a Built-in FPGA



FPGA Programming Model



Applications

Libraries

Directive Based

Programming Languages

???

HLS
(High Level Synthesis)

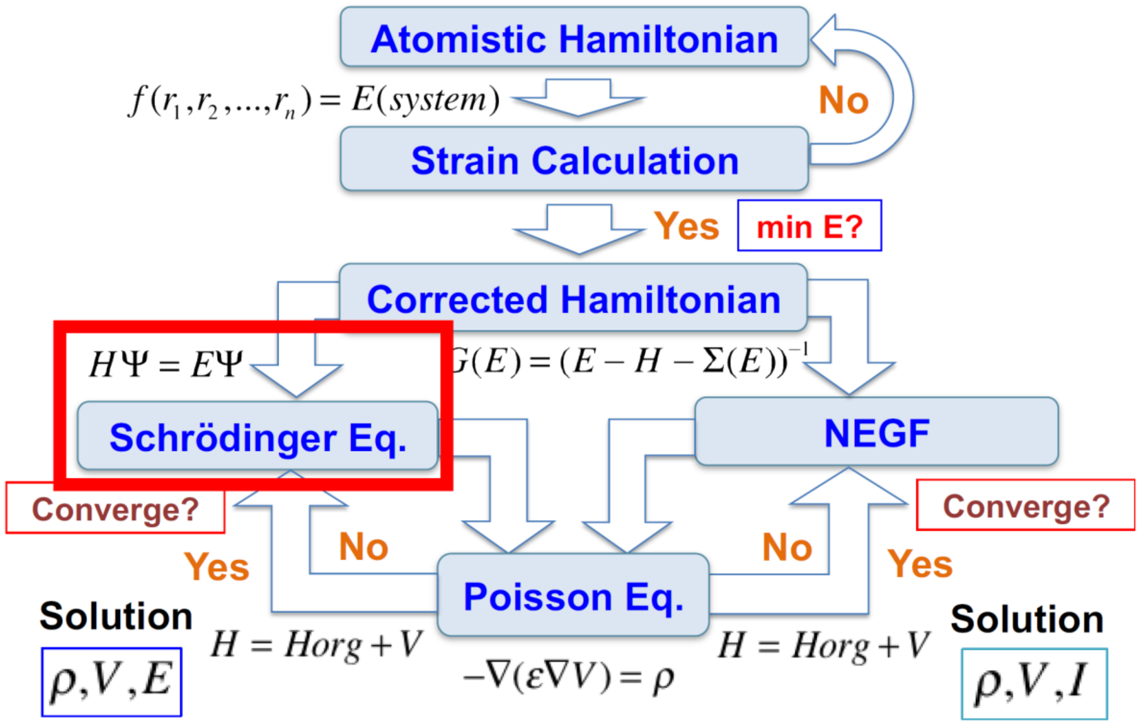
OpenCL

HDL
(Hardware Description Language)

Workload for Test

Electronic Structures: Sparse Matrix-vector Multiplications

Flow of NanoElectronics Modeling



Schrödinger Equation: Electronic Structure

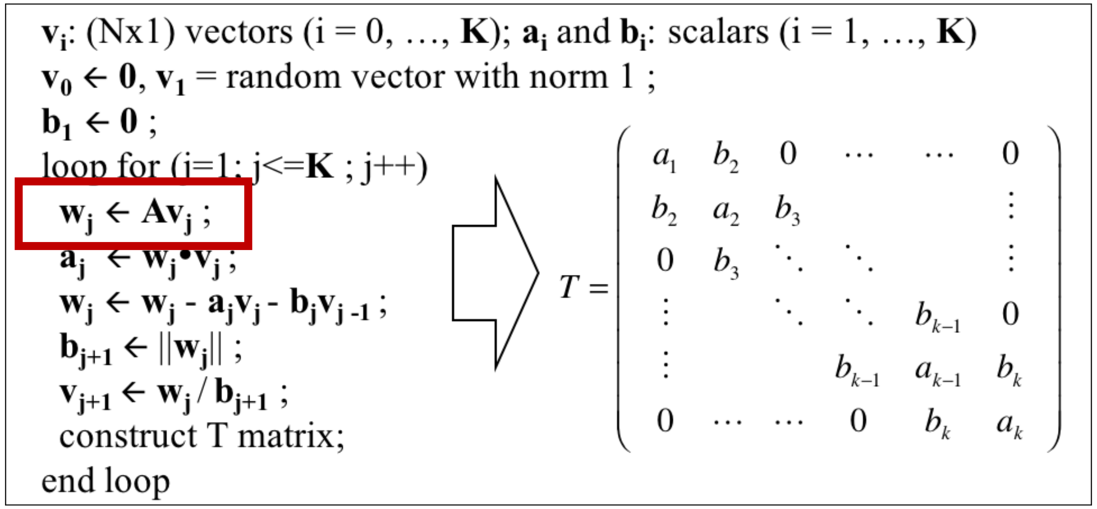
$$H\Psi=E\Psi$$

Eigenvalue problem

Schrödinger Eqs. w/ LANCZOS Algorithm

→ C. Lanczos, *J. Res. Natl. Bur. Stand.* 45, 255

- Normal Eigenvalue Problem (Electronic Structure)
- Hamiltonian is always symmetric
- Steps for Iteration: Purely Scalable Algebraic Ops.



```
$v_i$: (Nx1) vectors (i = 0, …, K); $a_i$ and $b_i$: scalars (i = 1, …, K)
$v_0 \leftarrow 0$, $v_1$ = random vector with norm 1 ;
$b_1 \leftarrow 0$ ;
loop for (j=1; j<=K ; j++)
  $w_j \leftarrow Av_j$ ;
  $a_j \leftarrow w_j \bullet v_j$ ;
  $w_j \leftarrow w_j - a_j v_j - b_j v_{j-1}$ ;
  $b_{j+1} \leftarrow ||w_j||$ ;
  $v_{j+1} \leftarrow w_j / b_{j+1}$ ;
  construct T matrix;
end loop
```

$$T=\begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & \vdots \\ 0 & b_3 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & b_{k-1} & 0 \\ \vdots & & & b_{k-1} & a_{k-1} & b_k \\ 0 & \cdots & \cdots & 0 & b_k & a_k \end{pmatrix}$$

Compressed Sparse Row(CSR) Format

- Storage for Large-scale Hamiltonian Matrices.

Workload for Test

Execution time of LANCZOS algorithm

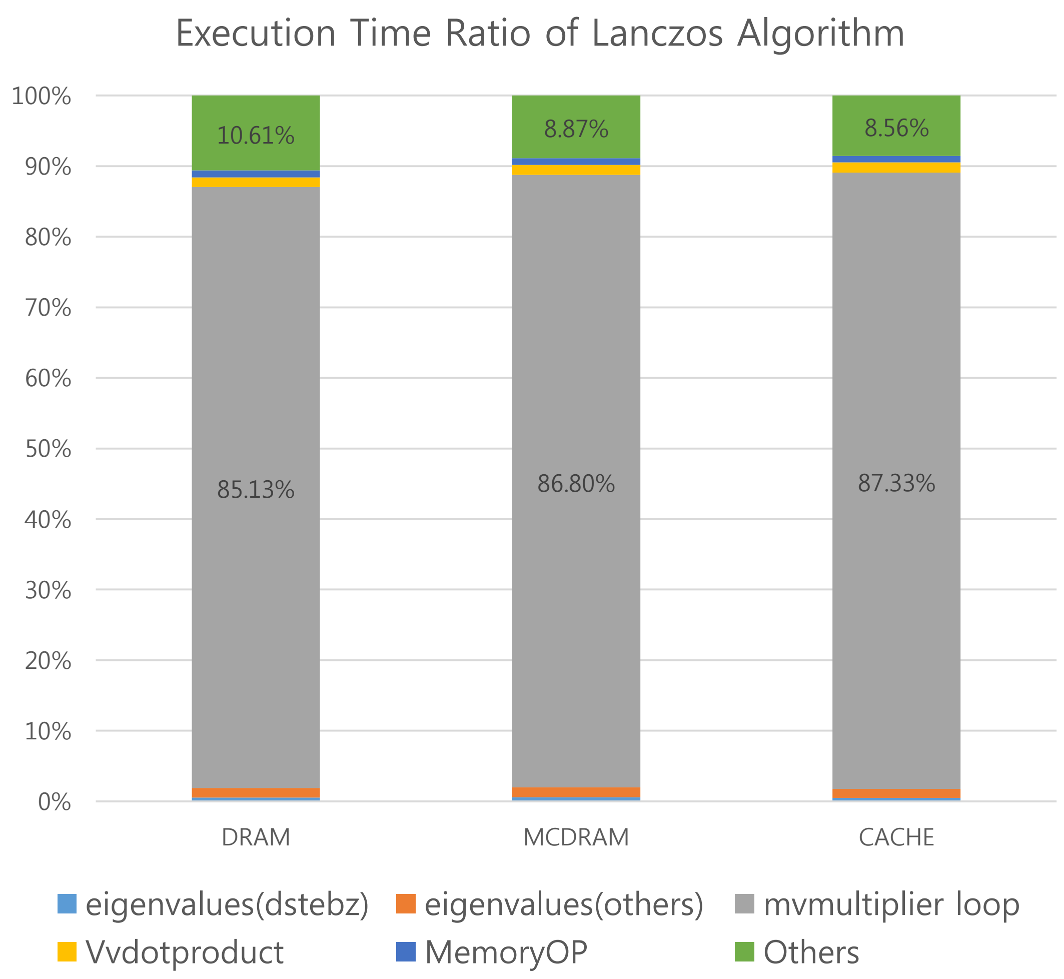
Schrödinger Eqs. w/ LANCZOS Algorithm

→ C. Lanczos, *J. Res. Natl. Bur. Stand.* 45, 255

- Normal Eigenvalue Problem (Electronic Structure)
- Hamiltonian is always symmetric
- Steps for Iteration: Purely Scalable Algebraic Ops.

```
$\mathbf{v_i}$: (Nx1) vectors (i = 0, …, $\mathbf{K}$); $\mathbf{a_i}$ and $\mathbf{b_i}$: scalars (i = 1, …, $\mathbf{K}$)
$\mathbf{v_0} \leftarrow \mathbf{0}$, $\mathbf{v_1}$ = random vector with norm 1 ;
$\mathbf{b_1} \leftarrow \mathbf{0}$ ;
loop for (j=1; j<=$\mathbf{K}$ ; j++)
    $\boxed{\mathbf{w_j} \leftarrow \mathbf{Av_j}}$ ;
    $\mathbf{a_j} \leftarrow \mathbf{w_j} \bullet \mathbf{v_j}$ ;
    $\mathbf{w_j} \leftarrow \mathbf{w_j} - \mathbf{a_j v_j} - \mathbf{b_j v_{j-1}}$ ;
    $\mathbf{b_{j+1}} \leftarrow ||\mathbf{w_j}||$ ;
    $\mathbf{v_{j+1}} \leftarrow \mathbf{w_j} / \mathbf{b_{j+1}}$ ;
    construct T matrix;
end loop
```

$$\Rightarrow \quad T = \begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & \vdots \\ 0 & b_3 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & b_{k-1} & 0 \\ \vdots & & & b_{k-1} & a_{k-1} & b_k \\ 0 & \cdots & \cdots & 0 & b_k & a_k \end{pmatrix}$$



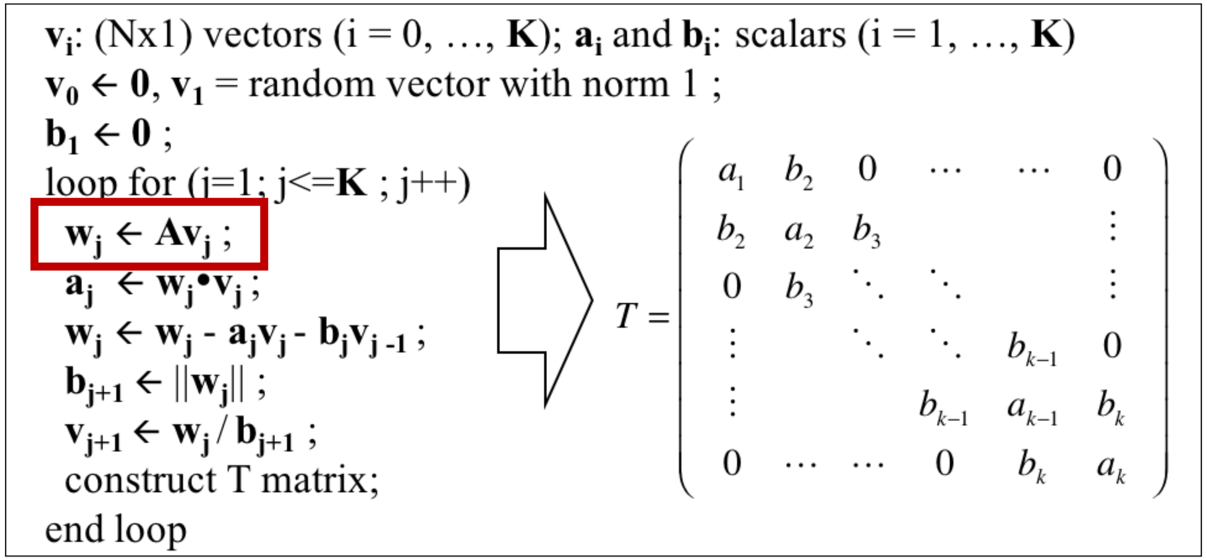
Workload for Test

Serial code of SpMV

Schrödinger Eqs. w/ LANCZOS Algorithm

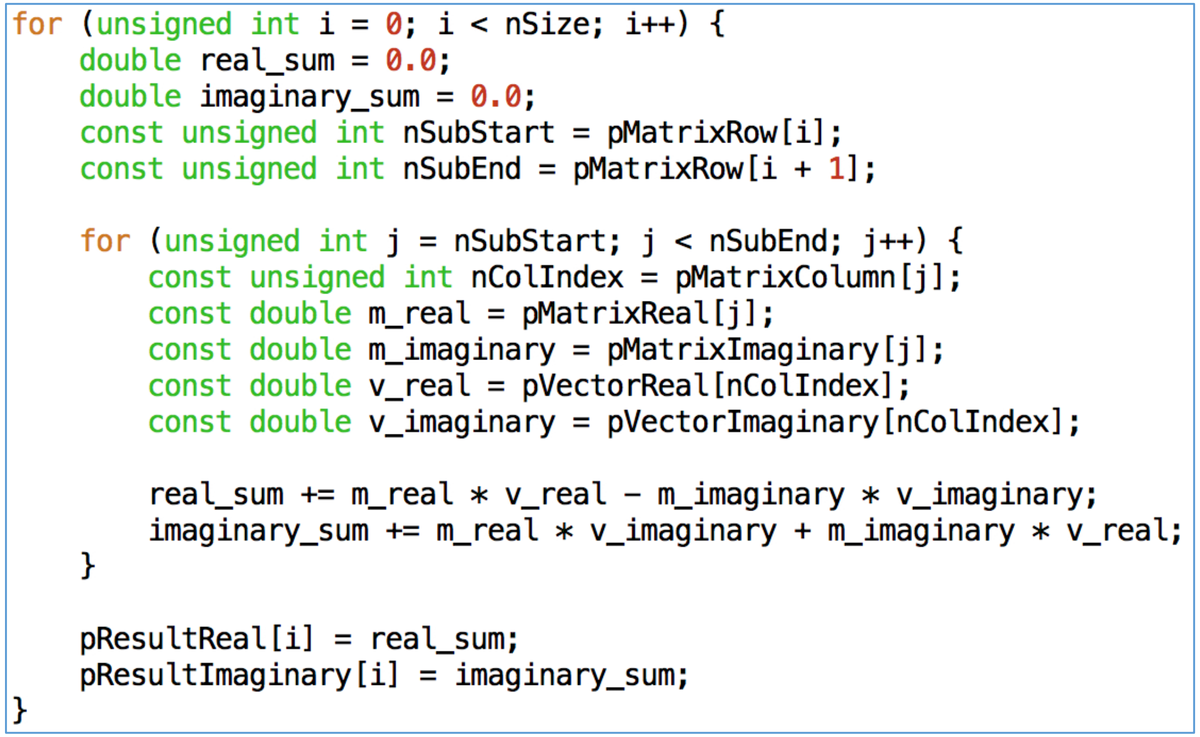
→ C. Lanczos, *J. Res. Natl. Bur. Stand.* 45, 255

- Normal Eigenvalue Problem (Electronic Structure)
- Hamiltonian is always symmetric
- Steps for Iteration: Purely Scalable Algebraic Ops.



```
$\mathbf{v}_i$: (Nx1) vectors (i = 0, …, $\mathbf{K}$); $\mathbf{a}_i$ and $\mathbf{b}_i$: scalars (i = 1, …, $\mathbf{K}$)
$\mathbf{v}_0 \leftarrow \mathbf{0}$, $\mathbf{v}_1$ = random vector with norm 1 ;
$\mathbf{b}_1 \leftarrow \mathbf{0}$ ;
loop for (j=1; j<=$\mathbf{K}$ ; j++)
    $\boxed{\mathbf{w}_j \leftarrow \mathbf{A}\mathbf{v}_j}$ ;
    $\mathbf{a}_j \leftarrow \mathbf{w}_j \bullet \mathbf{v}_j$ ;
    $\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{a}_j\mathbf{v}_j - \mathbf{b}_j\mathbf{v}_{j-1}$ ;
    $\mathbf{b}_{j+1} \leftarrow ||\mathbf{w}_j||$ ;
    $\mathbf{v}_{j+1} \leftarrow \mathbf{w}_j / \mathbf{b}_{j+1}$ ;
    construct T matrix;
end loop
```

$$T = \begin{pmatrix} a_1 & b_2 & 0 & \cdots & \cdots & 0 \\ b_2 & a_2 & b_3 & & & \vdots \\ 0 & b_3 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & b_{k-1} & 0 \\ \vdots & & & b_{k-1} & a_{k-1} & b_k \\ 0 & \cdots & \cdots & 0 & b_k & a_k \end{pmatrix}$$



```
for (unsigned int i = 0; i < nSize; i++) {
    double real_sum = 0.0;
    double imaginary_sum = 0.0;
    const unsigned int nSubStart = pMatrixRow[i];
    const unsigned int nSubEnd = pMatrixRow[i + 1];

    for (unsigned int j = nSubStart; j < nSubEnd; j++) {
        const unsigned int nColIndex = pMatrixColumn[j];
        const double m_real = pMatrixReal[j];
        const double m_imaginary = pMatrixImaginary[j];
        const double v_real = pVectorReal[nColIndex];
        const double v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }

    pResultReal[i] = real_sum;
    pResultImaginary[i] = imaginary_sum;
}
```

Compressed Sparse Row Format

- Storage for Large-scale Hamiltonian Matrices.

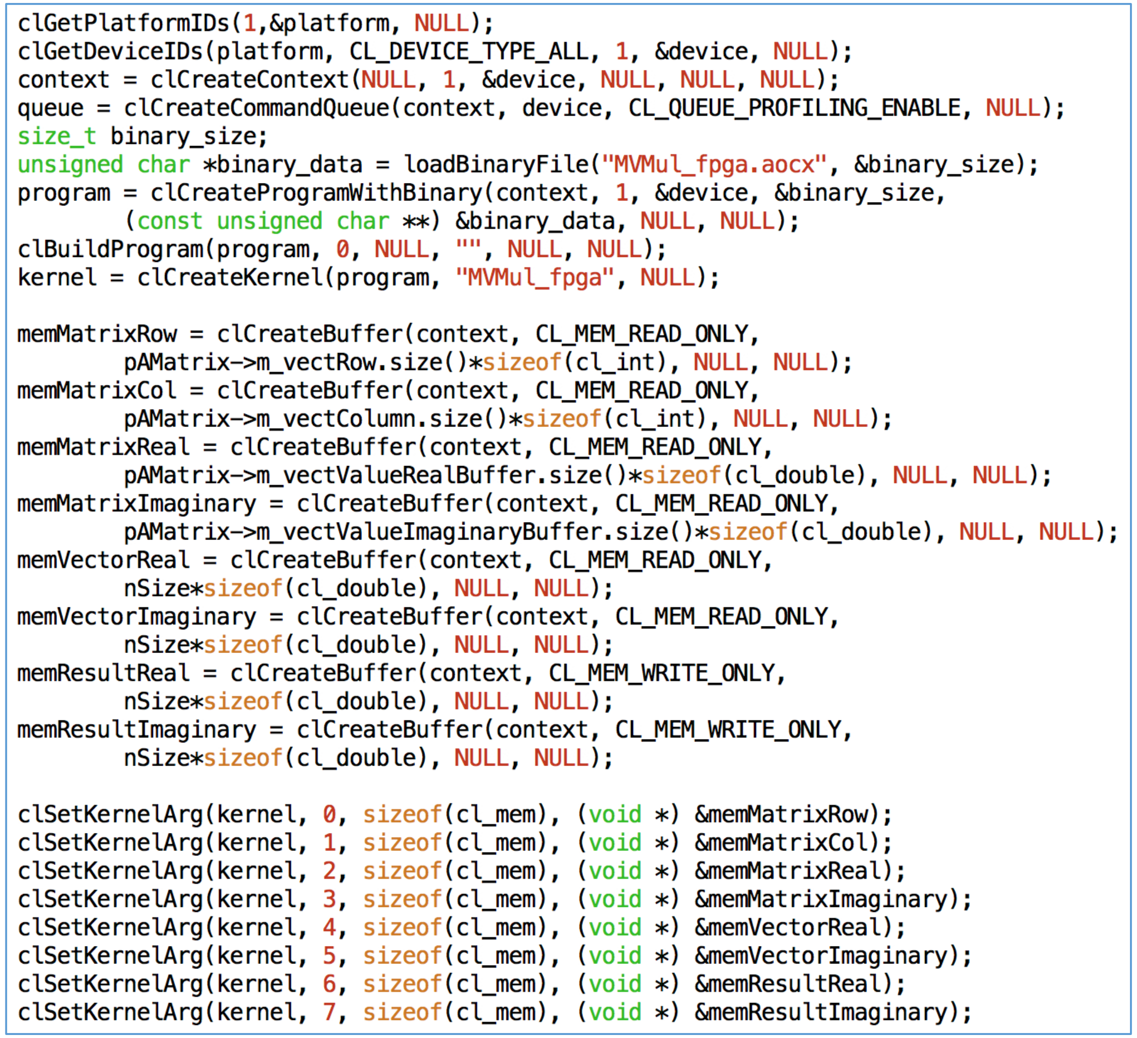
Strategy for Porting to FPGA

Core Module for SpMV

Computing Strategy

- All the multiplications are performed on a FPGA side
- Need to create buffers to store matrix, in/out vectors
- Involves copy in/out to /from FPGA to get the correct results

OpenCL Host Code →



```
clGetPlatformIDs(1,&platform, NULL);
clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL, 1, &device, NULL);
context = clCreateContext(NULL, 1, &device, NULL, NULL, NULL);
queue = clCreateCommandQueue(context, device, CL_QUEUE_PROFILING_ENABLE, NULL);
size_t binary_size;
unsigned char *binary_data = loadBinaryFile("MVMul_fpga.aocx", &binary_size);
program = clCreateProgramWithBinary(context, 1, &device, &binary_size,
        (const unsigned char **) &binary_data, NULL, NULL);
clBuildProgram(program, 0, NULL, "", NULL, NULL);
kernel = clCreateKernel(program, "MVMul_fpga", NULL);

memMatrixRow = clCreateBuffer(context, CL_MEM_READ_ONLY,
        pAMatrix->m_vectRow.size()*sizeof(cl_int), NULL, NULL);
memMatrixCol = clCreateBuffer(context, CL_MEM_READ_ONLY,
        pAMatrix->m_vectColumn.size()*sizeof(cl_int), NULL, NULL);
memMatrixReal = clCreateBuffer(context, CL_MEM_READ_ONLY,
        pAMatrix->m_vectValueRealBuffer.size()*sizeof(cl_double), NULL, NULL);
memMatrixImaginary = clCreateBuffer(context, CL_MEM_READ_ONLY,
        pAMatrix->m_vectValueImaginaryBuffer.size()*sizeof(cl_double), NULL, NULL);
memVectorReal = clCreateBuffer(context, CL_MEM_READ_ONLY,
        nSize*sizeof(cl_double), NULL, NULL);
memVectorImaginary = clCreateBuffer(context, CL_MEM_READ_ONLY,
        nSize*sizeof(cl_double), NULL, NULL);
memResultReal = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        nSize*sizeof(cl_double), NULL, NULL);
memResultImaginary = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
        nSize*sizeof(cl_double), NULL, NULL);

clSetKernelArg(kernel, 0, sizeof(cl_mem), (void *) &memMatrixRow);
clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *) &memMatrixCol);
clSetKernelArg(kernel, 2, sizeof(cl_mem), (void *) &memMatrixReal);
clSetKernelArg(kernel, 3, sizeof(cl_mem), (void *) &memMatrixImaginary);
clSetKernelArg(kernel, 4, sizeof(cl_mem), (void *) &memVectorReal);
clSetKernelArg(kernel, 5, sizeof(cl_mem), (void *) &memVectorImaginary);
clSetKernelArg(kernel, 6, sizeof(cl_mem), (void *) &memResultReal);
clSetKernelArg(kernel, 7, sizeof(cl_mem), (void *) &memResultImaginary);
```

Select FPGA device and load kernel

Create buffer for

1) row/column index (RO)
2) A matrix (RO)
3) X vector (RO)
4) Y result vector (WO)

RO : Read-Only WO : Write-Only

Set kernel argument (parameters of func. Call)

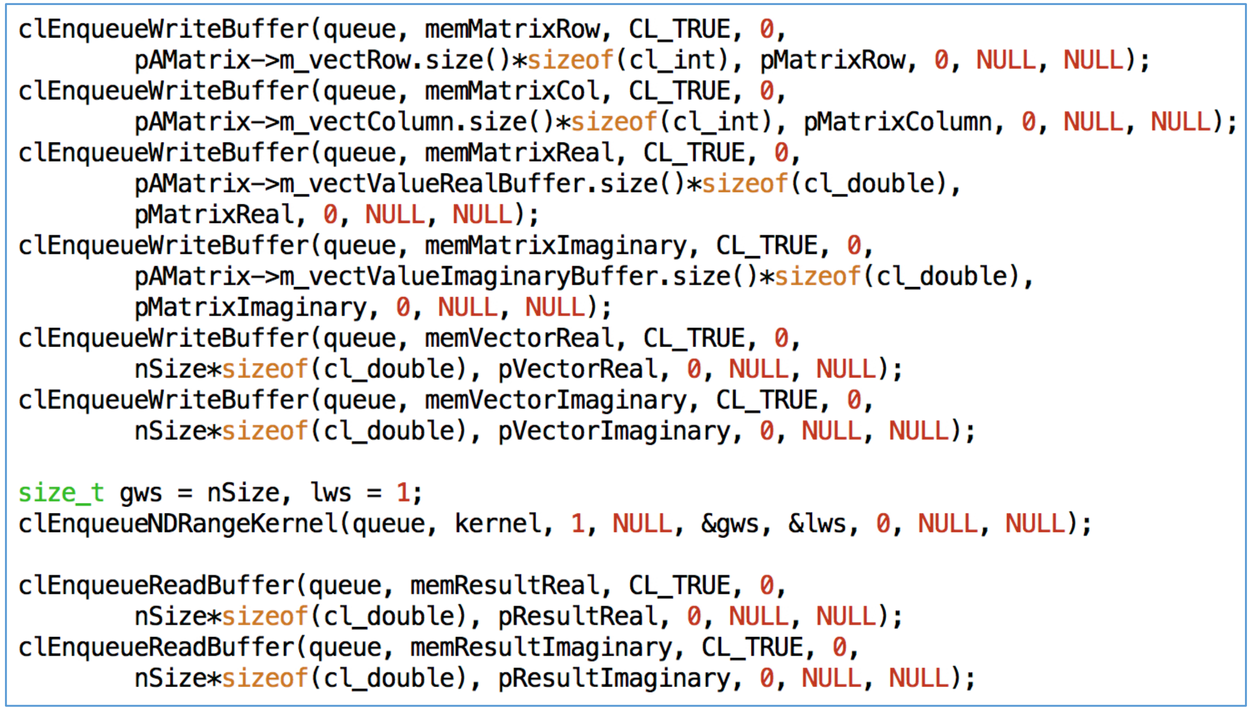
Strategy for Porting to FPGA (Cont.)

Core Module for SpMV

Computing Strategy

- All the multiplications are performed on a FPGA side
- Need to create buffers to store matrix, in/out vectors
- Involves copy in/out to /from FPGA to get the correct results

OpenCL Host Code → (con't)



```
clEnqueueWriteBuffer(queue, memMatrixRow, CL_TRUE, 0,
        pAMatrix->m_vectRow.size()*sizeof(cl_int), pMatrixRow, 0, NULL, NULL);
clEnqueueWriteBuffer(queue, memMatrixCol, CL_TRUE, 0,
        pAMatrix->m_vectColumn.size()*sizeof(cl_int), pMatrixColumn, 0, NULL, NULL);
clEnqueueWriteBuffer(queue, memMatrixReal, CL_TRUE, 0,
        pAMatrix->m_vectValueRealBuffer.size()*sizeof(cl_double),
        pMatrixReal, 0, NULL, NULL);
clEnqueueWriteBuffer(queue, memMatrixImaginary, CL_TRUE, 0,
        pAMatrix->m_vectValueImaginaryBuffer.size()*sizeof(cl_double),
        pMatrixImaginary, 0, NULL, NULL);
clEnqueueWriteBuffer(queue, memVectorReal, CL_TRUE, 0,
        nSize*sizeof(cl_double), pVectorReal, 0, NULL, NULL);
clEnqueueWriteBuffer(queue, memVectorImaginary, CL_TRUE, 0,
        nSize*sizeof(cl_double), pVectorImaginary, 0, NULL, NULL);

size_t gws = nSize, lws = 1;
clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &gws, &lws, 0, NULL, NULL);

clEnqueueReadBuffer(queue, memResultReal, CL_TRUE, 0,
        nSize*sizeof(cl_double), pResultReal, 0, NULL, NULL);
clEnqueueReadBuffer(queue, memResultImaginary, CL_TRUE, 0,
        nSize*sizeof(cl_double), pResultImaginary, 0, NULL, NULL);
```

Data transfer
From host to device

Call kernel function

Data transfer
From device to host

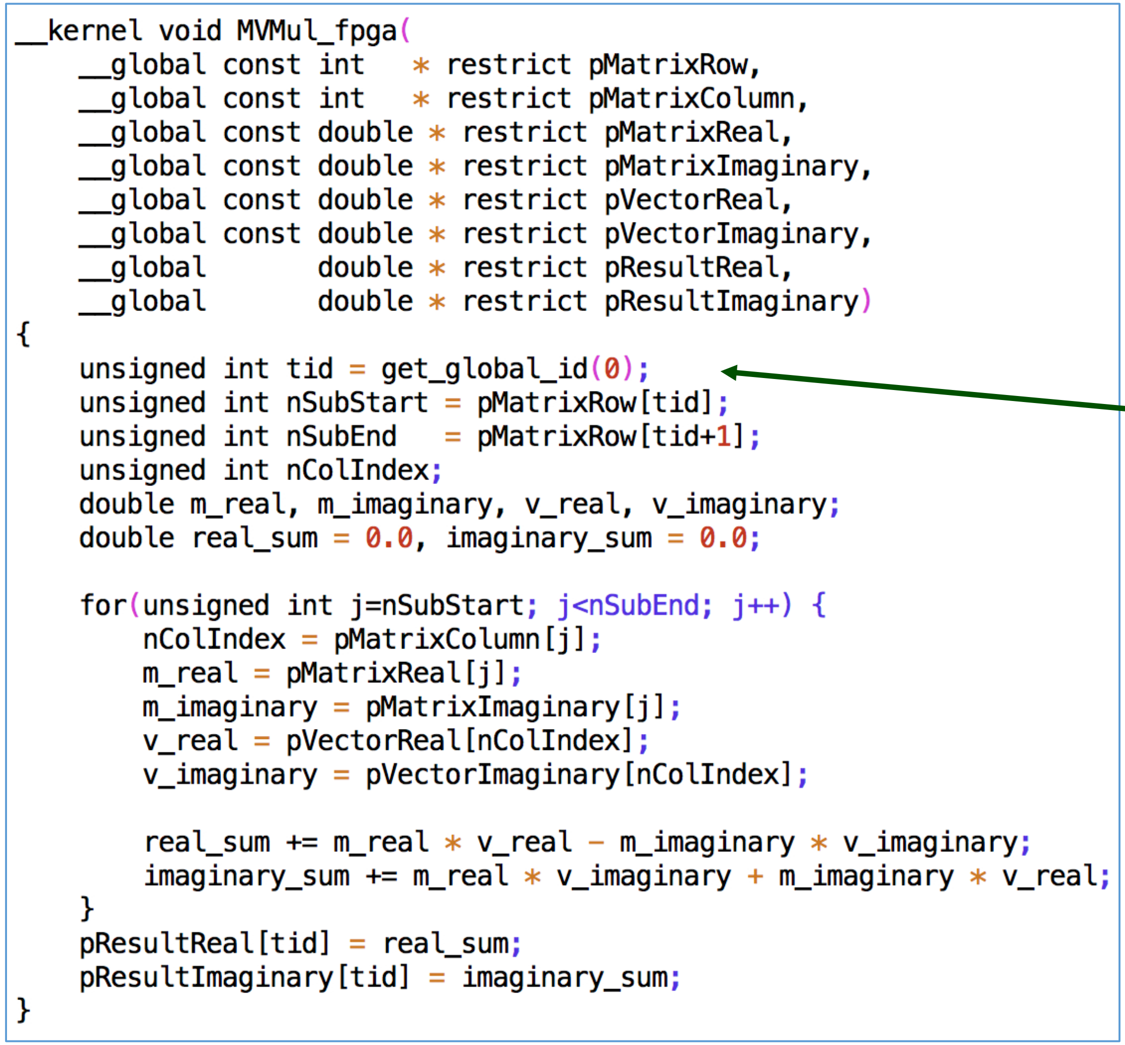
Strategy for Porting to FPGA (Cont.)

Core Module for SpMV

Computing Strategy

- All the multiplications are performed on a FPGA side
- Need to create buffers to store matrix, in/out vectors
- Involves copy in/out to /from FPGA to get the correct results

OpenCL Kernel Code →



```
__kernel void MVMul_fpga(
    __global const int    * restrict pMatrixRow,
    __global const int    * restrict pMatrixColumn,
    __global const double * restrict pMatrixReal,
    __global const double * restrict pMatrixImaginary,
    __global const double * restrict pVectorReal,
    __global const double * restrict pVectorImaginary,
    __global       double * restrict pResultReal,
    __global       double * restrict pResultImaginary)
{
    unsigned int tid = get_global_id(0);
    unsigned int nSubStart = pMatrixRow[tid];
    unsigned int nSubEnd   = pMatrixRow[tid+1];
    unsigned int nColIndex;
    double m_real, m_imaginary, v_real, v_imaginary;
    double real_sum = 0.0, imaginary_sum = 0.0;

    for(unsigned int j=nSubStart; j<nSubEnd; j++) {
        nColIndex = pMatrixColumn[j];
        m_real = pMatrixReal[j];
        m_imaginary = pMatrixImaginary[j];
        v_real = pVectorReal[nColIndex];
        v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }
    pResultReal[tid] = real_sum;
    pResultImaginary[tid] = imaginary_sum;
}
```

Function parameter

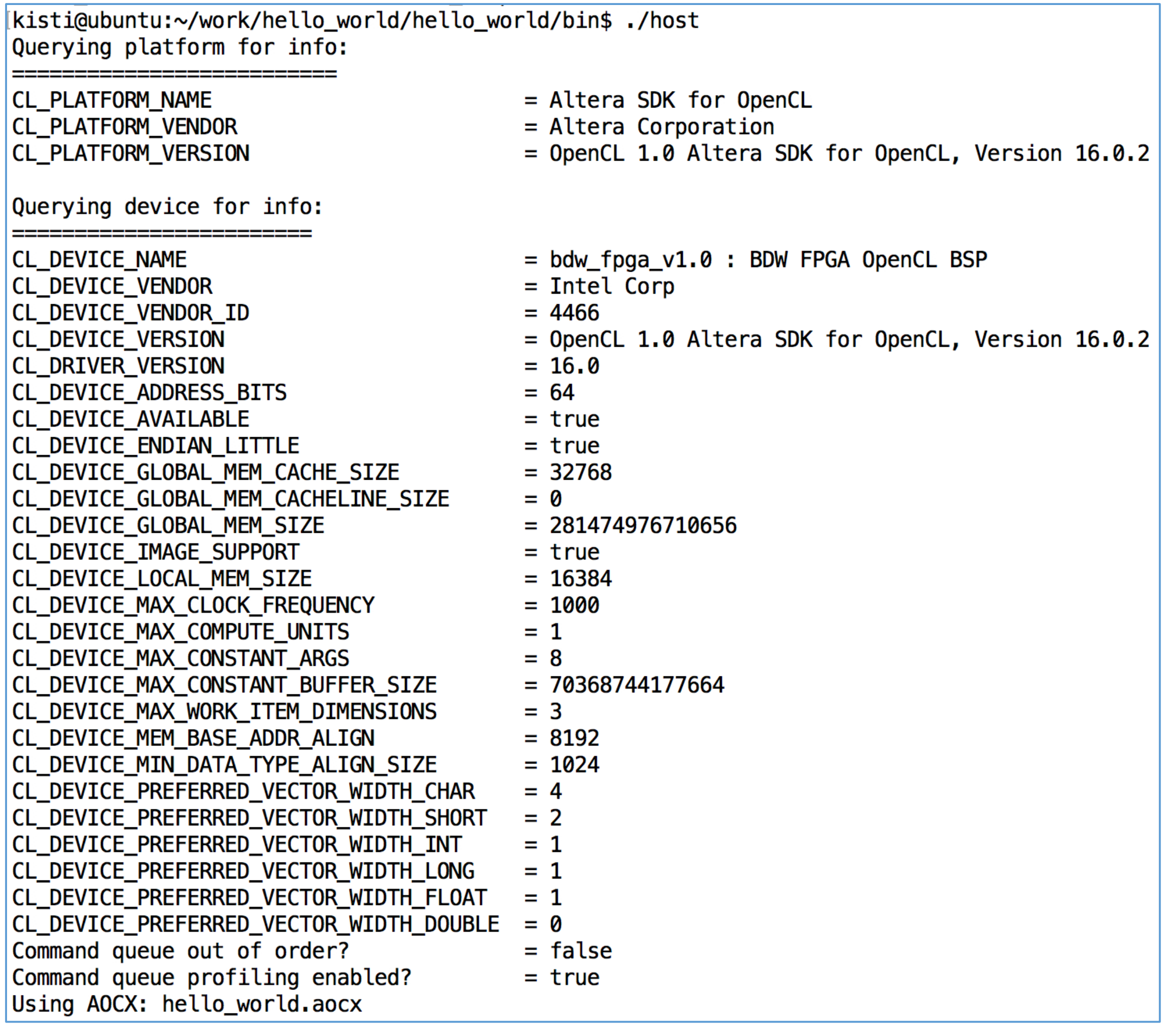
Map each thread for each row (i.e. parallelize for row)

Multiplication for each row

Environment for Development

- **Setup condition**
  - Intel FPGA SDK for OpenCL ver. 16.0.2
  - Quartus Prime Pro 16.0
  - AAL(Accelerator Abstract Layer) Kernel 5.0.2
  - OpenCL BSP(board support package) 1.0

- **Test environment**
  - Convergence criterion: 10^{-8}eV
  - Maximum # of iterations: 10^4
  - Mission: find 10 lowest conduction band energy levels of Si:P QDs

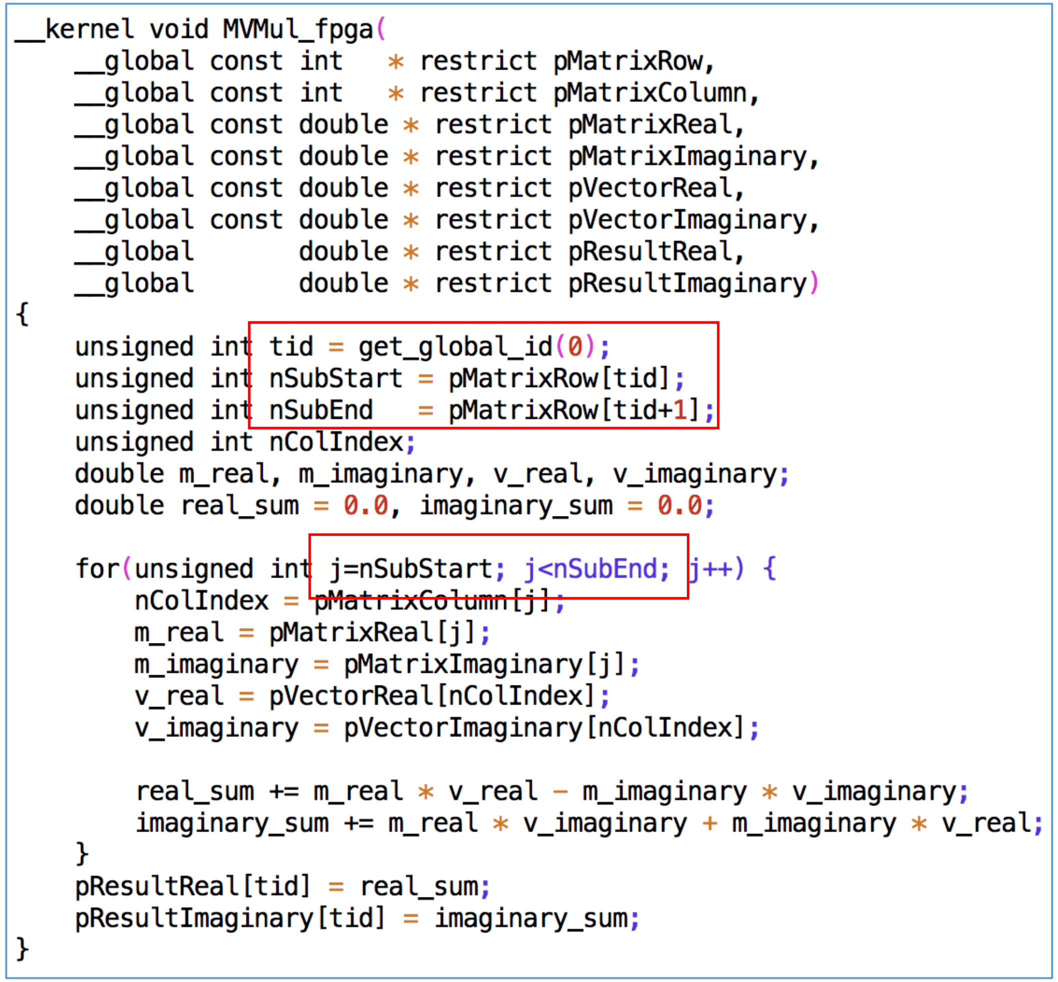


```
kisti@ubuntu:~/work/hello_world/hello_world/bin$ ./host
Querying platform for info:
==========================
CL_PLATFORM_NAME                         = Altera SDK for OpenCL
CL_PLATFORM_VENDOR                       = Altera Corporation
CL_PLATFORM_VERSION                      = OpenCL 1.0 Altera SDK for OpenCL, Version 16.0.2

Querying device for info:
========================
CL_DEVICE_NAME                           = bdw_fpga_v1.0 : BDW FPGA OpenCL BSP
CL_DEVICE_VENDOR                         = Intel Corp
CL_DEVICE_VENDOR_ID                      = 4466
CL_DEVICE_VERSION                        = OpenCL 1.0 Altera SDK for OpenCL, Version 16.0.2
CL_DRIVER_VERSION                        = 16.0
CL_DEVICE_ADDRESS_BITS                   = 64
CL_DEVICE_AVAILABLE                      = true
CL_DEVICE_ENDIAN_LITTLE                  = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE          = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE      = 0
CL_DEVICE_GLOBAL_MEM_SIZE                = 281474976710656
CL_DEVICE_IMAGE_SUPPORT                  = true
CL_DEVICE_LOCAL_MEM_SIZE                 = 16384
CL_DEVICE_MAX_CLOCK_FREQUENCY            = 1000
CL_DEVICE_MAX_COMPUTE_UNITS              = 1
CL_DEVICE_MAX_CONSTANT_ARGS              = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE       = 70368744177664
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS       = 3
CL_DEVICE_MEM_BASE_ADDR_ALIGN            = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE       = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR    = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT   = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT     = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG    = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT   = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE  = 0
Command queue out of order?              = false
Command queue profiling enabled?         = true
Using AOCX: hello_world.aocx
```

Optimization with SIMD

Cannot be directly applied: thread ID dependency



```
__kernel void MVMul_fpga(
    __global const int    * restrict pMatrixRow,
    __global const int    * restrict pMatrixColumn,
    __global const double * restrict pMatrixReal,
    __global const double * restrict pMatrixImaginary,
    __global const double * restrict pVectorReal,
    __global const double * restrict pVectorImaginary,
    __global       double * restrict pResultReal,
    __global       double * restrict pResultImaginary)
{
    unsigned int tid = get_global_id(0);
    unsigned int nSubStart = pMatrixRow[tid];
    unsigned int nSubEnd   = pMatrixRow[tid+1];
    unsigned int nColIndex;
    double m_real, m_imaginary, v_real, v_imaginary;
    double real_sum = 0.0, imaginary_sum = 0.0;

    for(unsigned int j=nSubStart; j<nSubEnd; j++) {
        nColIndex = pMatrixColumn[j];
        m_real = pMatrixReal[j];
        m_imaginary = pMatrixImaginary[j];
        v_real = pVectorReal[nColIndex];
        v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }
    pResultReal[tid] = real_sum;
    pResultImaginary[tid] = imaginary_sum;
}
```

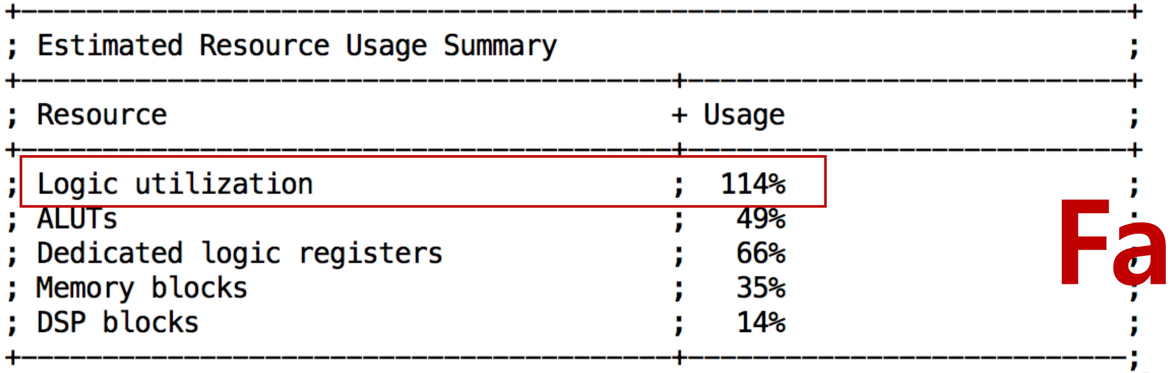
(Warning Message)

```
Compiler Warning: Kernel Vectorization: branching is thread ID
dependent ... cannot vectorize.
```

Changing computing logic to avoid dependency on Thread ID

- Encounters another problem

  → Logic is too complicate to be fit to FPGA



```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                   ;
+----------------------------------------+---------------------------+
; Resource                               + Usage                     ;
+----------------------------------------+---------------------------+
; Logic utilization                      ;  114%                     ;
; ALUTs                                  ;   49%                     ;
; Dedicated logic registers              ;   66%                     ;
; Memory blocks                          ;   35%                     ;
; DSP blocks                             ;   14%                     ;
+----------------------------------------+---------------------------;
```

Fail!

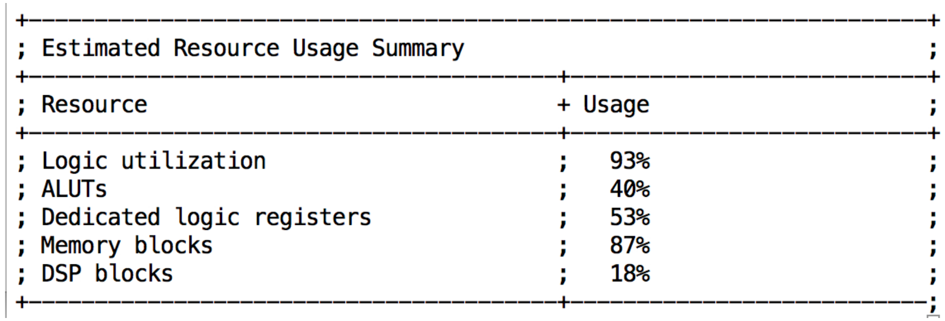
What about employing more computing unit then?

Optimization with copying "compute units"

Number of "compute units"

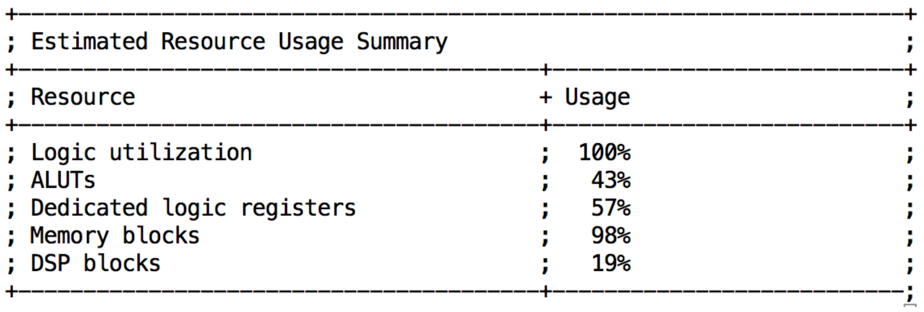
- The maximum copy units is 6 for this code
- Compilation error (logic util > 100%) w/ 7 units)

Report for num_compute_units(6)

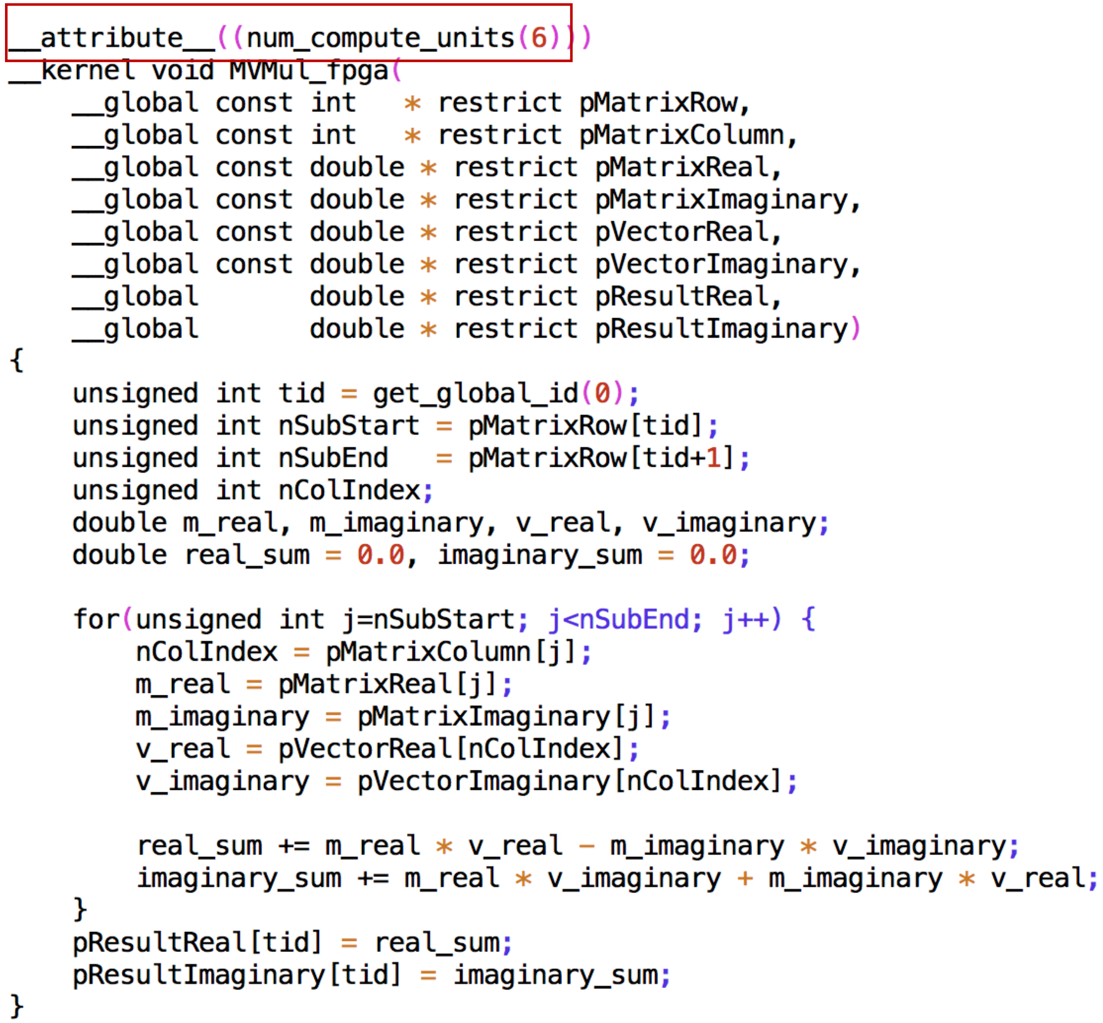


```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                   ;
+----------------------------------------+---------------------------+
; Resource                               + Usage                     ;
+----------------------------------------+---------------------------+
; Logic utilization                      ;   93%                     ;
; ALUTs                                  ;   40%                     ;
; Dedicated logic registers              ;   53%                     ;
; Memory blocks                          ;   87%                     ;
; DSP blocks                             ;   18%                     ;
+----------------------------------------+---------------------------;
```

Report for num_compute_units(7)



```
+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                   ;
+----------------------------------------+---------------------------+
; Resource                               + Usage                     ;
+----------------------------------------+---------------------------+
; Logic utilization                      ;  100%                     ;
; ALUTs                                  ;   43%                     ;
; Dedicated logic registers              ;   57%                     ;
; Memory blocks                          ;   98%                     ;
; DSP blocks                             ;   19%                     ;
+----------------------------------------+---------------------------;
```



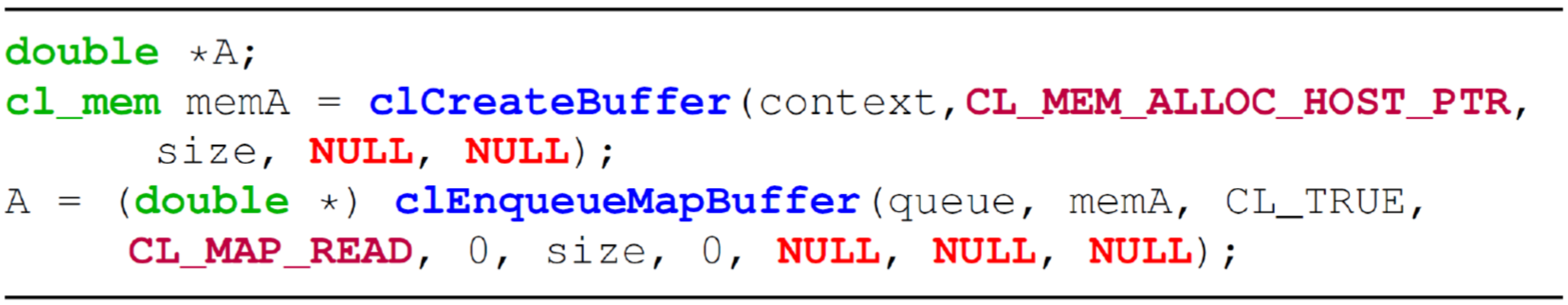
```
__attribute__((num_compute_units(6)))
__kernel void MVMul_fpga(
    __global const int    * restrict pMatrixRow,
    __global const int    * restrict pMatrixColumn,
    __global const double * restrict pMatrixReal,
    __global const double * restrict pMatrixImaginary,
    __global const double * restrict pVectorReal,
    __global const double * restrict pVectorImaginary,
    __global       double * restrict pResultReal,
    __global       double * restrict pResultImaginary)
{
    unsigned int tid = get_global_id(0);
    unsigned int nSubStart = pMatrixRow[tid];
    unsigned int nSubEnd   = pMatrixRow[tid+1];
    unsigned int nColIndex;
    double m_real, m_imaginary, v_real, v_imaginary;
    double real_sum = 0.0, imaginary_sum = 0.0;

    for(unsigned int j=nSubStart; j<nSubEnd; j++) {
        nColIndex = pMatrixColumn[j];
        m_real = pMatrixReal[j];
        m_imaginary = pMatrixImaginary[j];
        v_real = pVectorReal[nColIndex];
        v_imaginary = pVectorImaginary[nColIndex];

        real_sum += m_real * v_real - m_imaginary * v_imaginary;
        imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;
    }
    pResultReal[tid] = real_sum;
    pResultImaginary[tid] = imaginary_sum;
}
```

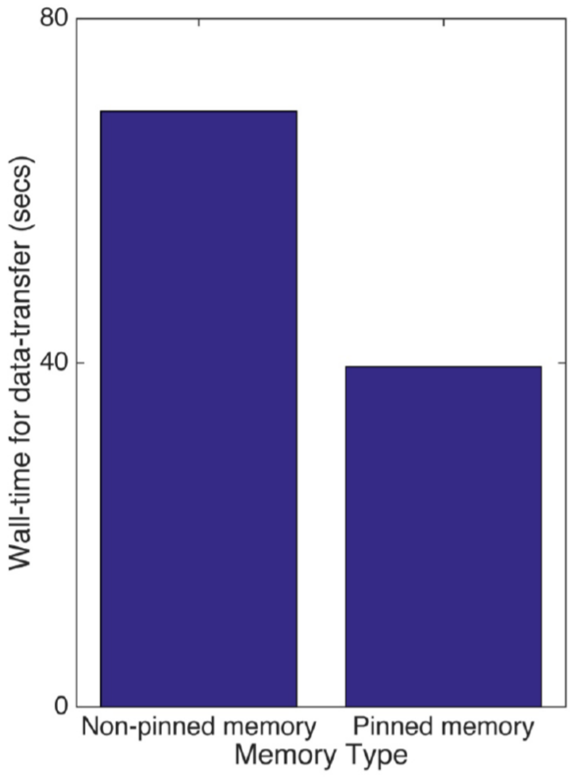
Optimization with Memory Pinning

- A well known technique to accelerate data transfer between host and device(FPGA)
- Data transfer is performed implicitly when pinned memory is used



```
double *A;
cl_mem memA = clCreateBuffer(context,CL_MEM_ALLOC_HOST_PTR,
      size, NULL, NULL);
A = (double *) clEnqueueMapBuffer(queue, memA, CL_TRUE,
     CL_MAP_READ, 0, size, 0, NULL, NULL, NULL);
```

- Utilization of pinned memory : ~1.64x bandwidth

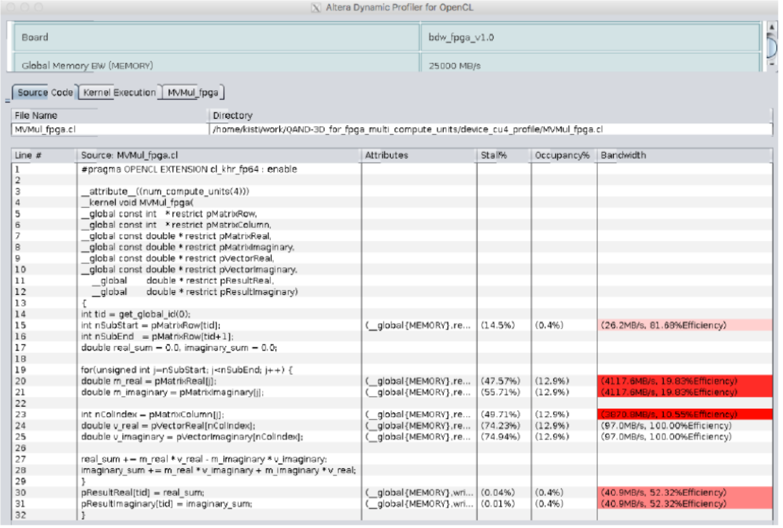


Conclusion and Future Plan

- **Implementation OpenCL kernel code for the built-in FPGA device**

- **Two strategies for performance improvement**
  - **Duplication of compute unit : up-to 6, ~1.58x speed-up**
  - **Utilization of pinned memory : ~1.64x bandwidth**

- **SpMV code with Altera Dynamic Profiler for OpenCL to analyze**
  - **Memory/channel/pipe accesses**
  - **SIMD usage**



Board	bdw_fpga_v1.0
Global Memory BW (MEMORY)	25000 MB/s

File Name	Directory
MVMul_fpga.cl	/home/kisti/work/QAND-3D_for_fpga_multi_compute_units/device_cu4_profile/MVMul_fpga.cl

Line #	Source: MVMul_fpga.cl	Attributes	Stall%	Occupancy%	Bandwidth
1	#pragma OPENCL EXTENSION cl_khr_fp64 : enable				
2					
3	__attribute__((num_compute_units(4)))				
4	__kernel void MVMul_fpga(				
5	__global const int *restrict pMatrixRow,				
6	__global const int *restrict pMatrixColumn,				
7	__global const double * restrict pMatrixReal,				
8	__global const double * restrict pMatrixImaginary,				
9	__global const double * restrict pVectorReal,				
10	__global const double * restrict pVectorImaginary,				
11	__global double * restrict pResultReal,				
12	__global double * restrict pResultImaginary)				
13	{				
14	int tid = get_global_id(0);				
15	int nSubStart = pMatrixRow[tid];	(__global{MEMORY},re...	(14.5%)	(0.4%)	(26.2MB/s, 81.68%Efficiency)
16	int nSubEnd = pMatrixRow[tid+1];				
17	double real_sum = 0.0, imaginary_sum = 0.0;				
18					
19	for(unsigned int j=nSubStart; j<nSubEnd; j++) {				
20	double m_real = pMatrixReal[j];	(__global{MEMORY},re...	(47.57%)	(12.9%)	(4117.6MB/s, 19.83%Efficiency)
21	double m_imaginary = pMatrixImaginary[j];	(__global{MEMORY},re...	(55.71%)	(12.9%)	(4117.6MB/s, 19.83%Efficiency)
22					
23	int nColIndex = pMatrixColumn[j];	(__global{MEMORY},re...	(49.71%)	(12.9%)	(3870.8MB/s, 10.55%Efficiency)
24	double v_real = pVectorReal[nColIndex];	(__global{MEMORY},re...	(74.23%)	(12.9%)	(97.0MB/s, 100.00%Efficiency)
25	double v_imaginary = pVectorImaginary[nColIndex];	(__global{MEMORY},re...	(74.94%)	(12.9%)	(97.0MB/s, 100.00%Efficiency)
26					
27	real_sum += m_real * v_real - m_imaginary * v_imaginary;				
28	imaginary_sum += m_real * v_imaginary + m_imaginary * v_real;				
29	}				
30	pResultReal[tid] = real_sum;	(__global{MEMORY},wri...	(0.04%)	(0.4%)	(40.9MB/s, 52.32%Efficiency)
31	pResultImaginary[tid] = imaginary_sum;	(__global{MEMORY},wri...	(0.01%)	(0.4%)	(40.9MB/s, 52.32%Efficiency)
32	}				

Questions? / Comments?

Thank you!