# Advanced Optimizations

# Task creation example

# MxM w/ blocking w/ task creation… (I/III)

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(2)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
            for (unsigned int j = 0; j < msize/BSIZE; j++) {
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci);
   }}}
   #pragma oss taskwait
```

# MxM w/ blocking w/ task creation… (II/III)

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(2)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
            for (unsigned int j = 0; j < msize/BSIZE; j++) {
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci);
   }}}
   #pragma oss taskwait
```

- Both tasks are FPGA tasks
- Blocking matmulFull
  - **Same as SMP code**
  - **Parameters dimensions are not constants**
  - **No copies of parameters**
  - **Task creation**
- Picos runtime takes care of the dependencies

# Task creation: Compiler Support (II/III)

```
void matmulFull_hw(int msize, __mcxx_ptr_t<elem_t > a, __mcxx_ptr_t<elem_t > b, __mcxx_ptr_t<elem_t > c, hls::stream<ap_uint<8> > &mcxx_spawnInPort, hls::stream<mcxx_outaxis> &mcxx_ou
tPort, hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    const unsigned int m2size = msize * msize;
    const unsigned int b2size = 32 * 32;
    for (unsigned int i = 0; i < msize / 32; i++) {
        for (unsigned int k = 0; k < msize / 32; k++) {
            const unsigned int ai = k * b2size + i * 32 * msize;
            for (unsigned int j = 0; j < msize / 32; j++) {
                const unsigned int ci = j * b2size + i * 32 * msize;
                const unsigned int bi = j * b2size + k * 32 * msize;
                {
                    unsigned long long __mcxx_args[3];
                    unsigned long long __mcxx_deps[3];
                    __fpga_copyinfo_t __mcxx_copies[3];
                    __mcxx_ptr_t<elem_t > __mcxx_arg_0;
                    __mcxx_arg_0 = a + ai;
                    __mcxx_args[0U] = __mcxx_arg_0.val;
                    __mcxx_copies[0U].copy_address = __mcxx_arg_0.val;
                    __mcxx_copies[0U].arg_idx = 0U;
                    __mcxx_copies[0U].flags = 1U;
                    __mcxx_copies[0U].size = 4096ULL;
                    __mcxx_ptr_t<elem_t > __mcxx_dep_0;
                    __mcxx_dep_0 = a + ai;
                    __mcxx_deps[0U] = 1ULL << 58ULL | __mcxx_dep_0.val;
                    __mcxx_ptr_t<elem_t > __mcxx_arg_1;
                    __mcxx_arg_1 = b + bi;
```

...

```
                    __mcxx_copies[2U].flags = 3U;
                    __mcxx_copies[2U].size = 4096ULL;
                    __mcxx_ptr_t<elem_t > __mcxx_dep_2;
                    __mcxx_dep_2 = c + ci;
                    __mcxx_deps[2U] = 3ULL << 58ULL | __mcxx_dep_2.val;
                    {
                        mcxx_instrument_event(85U, 5ULL, mcxx_instr);
                        mcxx_task_create(5998371774ULL, 255U, 3U, __mcxx_args, 3U, __mcxx_deps, 3U, __mcxx_copies, mcxx_outPort);
                        mcxx_instrument_event(213U, 5ULL, mcxx_instr);
                    }
                }
            }
        }
    }
    {
        mcxx_instrument_event(85U, 1ULL, mcxx_instr);
        mcxx_taskwait(mcxx_spawnInPort, mcxx_outPort);
        mcxx_instrument_event(213U, 1ULL, mcxx_instr);
    }
```

- Task preparation
- Task creation
- Taskwait

# Task Preparation

```
void matmulFull_hw(int msize, __mcxx_ptr_t<elem_t > a, __mcxx_ptr_t<elem_t > b, __mcxx_ptr_t<elem_t > c, hls::stream<ap_uint<8> > &mcxx_spawnInPor
tPort, hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    const unsigned int m2size = msize * msize;
    const unsigned int b2size = 32 * 32;
    for (unsigned int i = 0; i < msize / 32; i++) {
        for (unsigned int k = 0; k < msize / 32; k++) {
            const unsigned int ai = k * b2size + i * 32 * msize;
            for (unsigned int j = 0; j < msize / 32; j++) {
                const unsigned int ci = j * b2size + i * 32 * msize;
                const unsigned int bi = j * b2size + k * 32 * msize;
                {
                    unsigned long long __mcxx_args[3];
                    unsigned long long __mcxx_deps[3];
                    __fpga_copyinfo_t __mcxx_copies[3];
    ...             __mcxx_ptr_t<elem_t > __mcxx_arg_0;
                    __mcxx_arg_0 = a + ai;
                    __mcxx_args[0U] = __mcxx_arg_0.val;
                    __mcxx_copies[0U].copy_address = __mcxx_arg_0.val;
                    __mcxx_copies[0U].arg_idx = 0U;
                    __mcxx_copies[0U].flags = 1U;
                    __mcxx_copies[0U].size = 4096ULL;
                    __mcxx_ptr_t<elem_t > __mcxx_dep_0;
                    __mcxx_dep_0 = a + ai;
                    __mcxx_deps[0U] = 1ULL << 58ULL | __mcxx_dep_0.val;
                    __mcxx_ptr_t<elem_t > __mcxx_arg_1;
                    __mcxx_arg_1 = b + bi;
```

# Task Creation and Taskwait

```
                __mcxx_copies[2U].flags = 3U;
                __mcxx_copies[2U].size = 4096ULL;
                __mcxx_ptr_t<elem_t > __mcxx_dep_2;
                __mcxx_dep_2 = c + ci;
                __mcxx_deps[2U] = 3ULL << 58ULL | __mcxx_dep_2.val;
                {
                    mcxx_instrument_event(85U, 5ULL, mcxx_instr);
                    mcxx_task_create(5998371774ULL, 255U, 3U, __mcxx_args, 3U, __mcxx_deps, 3U, __mcxx_copies, mcxx_outPort);
                    mcxx_instrument_event(213U, 5ULL, mcxx_instr);
                }
            }
        }
    }
}
{
    mcxx_instrument_event(85U, 1ULL, mcxx_instr);
    mcxx_taskwait(mcxx_spawnInPort, mcxx_outPort);
    mcxx_instrument_event(213U, 1ULL, mcxx_instr);
}
```

# What about the MxM Tile? (III/III)

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(2)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]) {
   unsigned int i, j, k;
   const unsigned int FACTOR=BSIZE/2;
...
   for (i = 0; i < BSIZE; i++) {
      for (j = 0; j < BSIZE; j++) {
         elem_t sum = c[i*BSIZE + j];
         for (k = 0; k < BSIZE; k++) {
            sum += a[i*BSIZE + k] * b[k*BSIZE + j];
         }
         c[i*BSIZE + j] = sum;
      }
   }
}
```

- The same!
  - There are **two instances** of MxM
  - **Copy_deps**, local copy of TILE

# MxM Tile: Compiler Support - NO CHANGE!

```
void matmulBlock_hw(elem_t a[1024], elem_t b[1024], elem_t c[1024], hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    unsigned int i, j, k;
  loop_i_matmul:
    for (i = 0; i < 32; i++) {
      loop_j_matmul:
        for (j = 0; j < 32; j++) {
            elem_t sum = c[i * 32 + j];
          loop_k_matmul:
            for (k = 0; k < 32; k++) {
                sum += a[i * 32 + k] * b[k * 32 + j];
            }
            c[i * 32 + j] = sum;
        }
    }
}
const unsigned int CONST_BSIZE = 32;
void mcxx_write_out_port(const ap_uint<64> data, const ap_uint<8> dest, const ap_uint<1> last, hls::stream<mcxx_outaxis>& mcxx_outPort) {
  #pragma HLS inline
  mcxx_outaxis axis_word;
  axis_word.data = data;
./fpga-naive-zedboard/HLS-32x32-naive/zedboard/matmul_ait/xilinx/HLS/matmulBlock_hw/matmulBlock_hw_hls_automatic_clang.cpp   39,1   25%
    ap_wait();
  }
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
  }
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
  matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
  }
```

- Kernel is the same
- Local copies and management in the wrapper

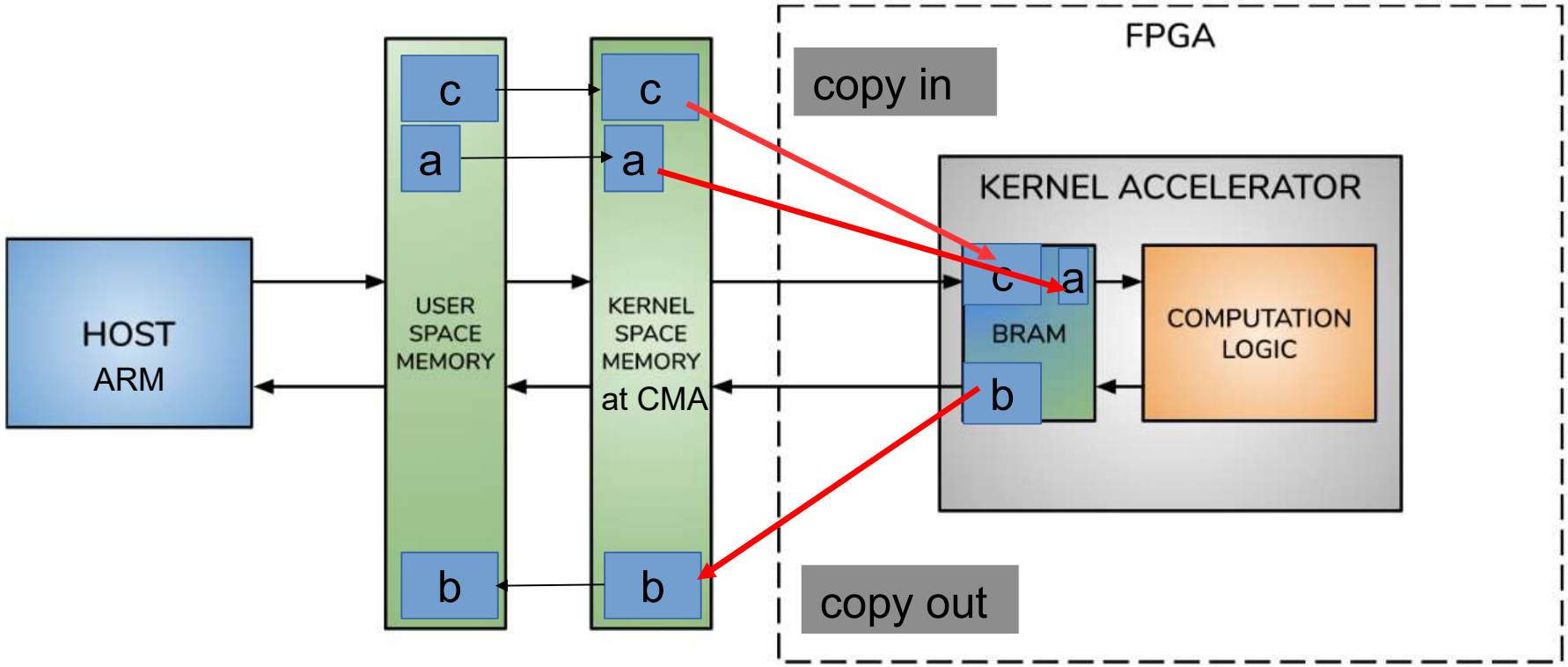
- Local copies using ports

```
void matmulBlock_hw_wrapper(hls::stream<ap_uint<
rData_t>& mcxx_instr) {
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface axis port=mcxx_inPort
#pragma HLS interface axis port=mcxx_outPort
#pragma HLS interface m_axi port=mcxx_a
#pragma HLS interface m_axi port=mcxx_b
#pragma HLS interface m_axi port=mcxx_c
#pragma HLS interface axis port=mcxx_instr
  static elem_t a[1024];
  static elem_t b[1024];
  static elem_t c[1024];
```

Local declaration for local copies

# Task creation example

# Execution Model (Data Transfer): scale




- Copies Optimizations:
  - Runtime take care of consecutives copies to the **SAME** accelerator
    - Ready Task Queue have info about copies that can be disabled.

# Affinity w/ blocking w/ task creations… (I/III)

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(NUM_INSTANCES) affinity(af)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE], int af);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
     for (unsigned int jj = 0; jj < msize/BSIZE; jj+=NUM_INSTANCES) {
       for (unsigned int k = 0; k < msize/BSIZE; k++) {
         for (unsigned int j = jj; j < jj+NUM_INSTANCES ; j++) {
            unsigned int const ai = k*b2size + i*BSIZE*msize;
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci, j%NUM_INSTANCES);
   }}}
   #pragma oss taskwait
```

- Affinity guarantees task scheduling to the accelerator we want to
- Programmer code should do it
- Which data transfer can be disabled?
  - C[i][j] for several k's?

# Affinity w/ blocking w/ task creation but... (II/III)

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(NUM_INSTANCES) affinity(af)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE], int af);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
     for (unsigned int jj = 0; jj < msize/BSIZE; jj+=NUM_INSTANCES) {
       for (unsigned int k = 0; k < msize/BSIZE; k++) {
         for (unsigned int j = jj; j < jj+NUM_INSTANCES ; j++) {
            unsigned int const ai = k*b2size + i*BSIZE*msize;
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci, j%NUM_INSTANCES);
   }}}
   #pragma oss taskwait
```

- Affinity guarantees task scheduling
- Which data transfer can be disabled?
  - NONE! There are inout dependences!

# Affinity w/ task creations… but no deps (III/III)

```
#pragma oss task device(fpga) copy_in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
copy_inout([CONST_BSIZE*CONST_BSIZE]c) num_instances(NUM_INSTANCES) affinity(af)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE], int af);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
     for (unsigned int jj = 0; jj < msize/BSIZE; jj+=NUM_INSTANCES) {
       for (unsigned int k = 0; k < msize/BSIZE; k++) {
         for (unsigned int j = jj; j < jj+NUM_INSTANCES ; j++) {
            unsigned int const ai = k*b2size + i*BSIZE*msize;
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci, j%NUM_INSTANCES);
   }}}
   #pragma oss taskwait
```

- No dependencies but copies
- Affinity guarantees task execution order
- Affinity guarantees task scheduling
- Now, inout copies are disabled

# MxM Tile: the same

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(2)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]) {
   unsigned int i, j, k;
   const unsigned int FACTOR=BSIZE/2;
...
   for (i = 0; i < BSIZE; i++) {
      for (j = 0; j < BSIZE; j++) {
         elem_t sum = c[i*BSIZE + j];
         for (k = 0; k < BSIZE; k++) {
            sum += a[i*BSIZE + k] * b[k*BSIZE + j];
         }
         c[i*BSIZE + j] = sum;
      }
   }
}
```

- There are **two instances** of MxM
- **Copy_deps**, local copy of TILE

# Execution Model (Data Transfer)

```
void matmulBlock_hw(elem_t a[1024], elem_t b[1024], elem_t c[1024], hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    unsigned int i, j, k;
  loop_i_matmul:
    for (i = 0; i < 32; i++) {
      loop_j_matmul:
        for (j = 0; j < 32; j++) {
            elem_t sum = c[i * 32 + j];
          loop_k_matmul:
            for (k = 0; k < 32; k++) {
                sum += a[i * 32 + k] * b[k * 32 + j];
            }
            c[i * 32 + j] = sum;
        }
    }
}
const unsigned int CONST_BSIZE = 32;
void mcxx_write_out_port(const ap_uint<64> data, const ap_uint<8> dest, const ap_uint<1> last, hls::stream<mcxx_outaxis>& mcxx_outPort) {
  #pragma HLS inline
  mcxx_outaxis axis_word;
  axis_word.data = data;
```

- Kernel is the same
- Local copies and management in the wrapper

./fpga-naive-zedboard/HLS-32x32-naive/zedboard/matmul_ait/xilinx/HLS/matmulBlock_hw/matmulBlock_hw_hls_automatic_clang.cpp 39,1 25%

```
    ap_wait();
  }
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
  }
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
  matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
  }
```

- Local copies using ports

```
void matmulBlock_hw_wrapper(hls::stream<ap_uint<
rData_t>& mcxx_instr) {
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface axis port=mcxx_inPort
#pragma HLS interface axis port=mcxx_outPort
#pragma HLS interface m_axi port=mcxx_a
#pragma HLS interface m_axi port=mcxx_b
#pragma HLS interface m_axi port=mcxx_c
#pragma HLS interface axis port=mcxx_instr
  static elem_t a[1024];
  static elem_t b[1024];
  static elem_t c[1024];
```

Local declaration for local copies

# Manual blocking

# Data Movement w/ Blocking (but no tasks)

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
inout([msize*msize]c) num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      memcpy(cc,&c[ci],BSIZE*BSIZE*sizeof(elem_t));
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         memcpy(aa,&a[ai],BSIZE*BSIZE*sizeof(elem_t));
         memcpy(bb,&b[bi],BSIZE*BSIZE*sizeof(elem_t));
         matmulBlock_hw(aa, bb, cc);
      }
      memcpy(&c[ci],cc,BSIZE*BSIZE*sizeof(elem_t));
    }
   }
}
```

- **No copy_deps:**
- LLVM/Clang will not generate memory copies inside the accelerator
- Runtime still will copy the data to accesible FPGA memory
- Variable size are not constants!

# FPGA Blocking: User should copy data

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
                        inout([msize*msize]c) num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

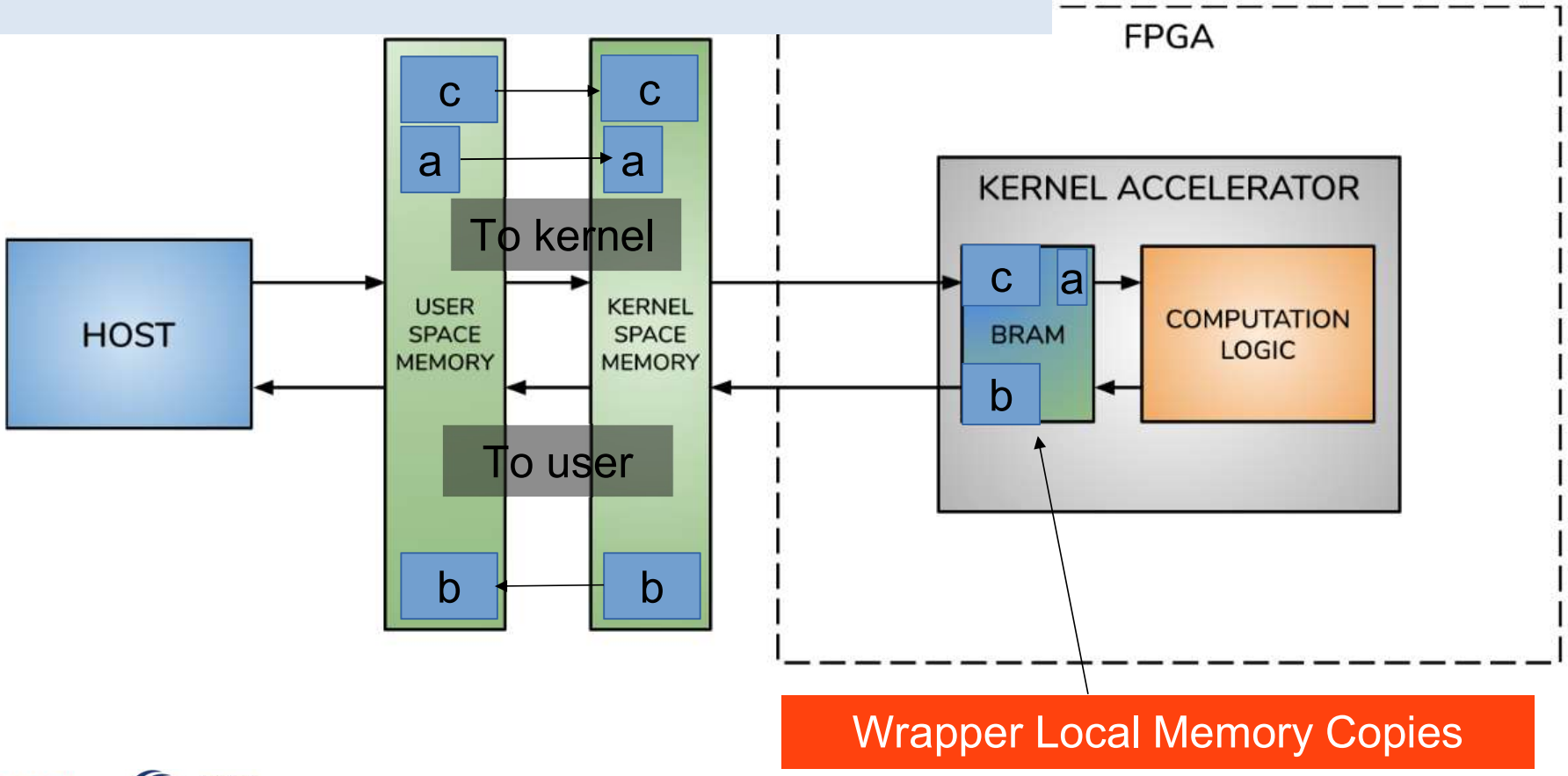
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      memcpy(cc,&c[ci],BSIZE*BSIZE*sizeof(elem_t));
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         memcpy(aa,&a[ai],BSIZE*BSIZE*sizeof(elem_t));
         memcpy(bb,&b[bi],BSIZE*BSIZE*sizeof(elem_t));
         matmulBlock_hw(aa, bb, cc);
      }
      memcpy(&c[ci],cc,BSIZE*BSIZE*sizeof(elem_t));
    }
   }
}
```

Arguments with no copies, LLVM helps:

- Addr is automatically passed in
- Addr transparent access to SMP Memory using AXI ports
- Transparent to the Programmer!
  - Be careful with memcpy !!!

# Data Transfer w/ Wrapper Copies: scale

```
const int SIZE=1024;
#pragma oss task device(fpga) in([SIZE]c, a) out([SIZE]b) copy_deps \
    num_instances(1)
void scale_task(double *b, double *c, double *a);
```



# Remember MxM Tile example

```
void matmulBlock_hw(elem_t a[1024], elem_t b[1024], elem_t c[1024], hls::stream<__mcxx_instrData_t> &mcxx_instr) {
    unsigned int i, j, k;
  loop_i_matmul:
    for (i = 0; i < 32; i++) {
      loop_j_matmul:
        for (j = 0; j < 32; j++) {
            elem_t sum = c[i * 32 + j];
          loop_k_matmul:
            for (k = 0; k < 32; k++) {
                sum += a[i * 32 + k] * b[k * 32 + j];
            }
            c[i * 32 + j] = sum;
        }
    }
}
const unsigned int CONST_BSIZE = 32;
void mcxx_write_out_port(const ap_uint<64> data, const ap_uint<8> dest, const ap_uint<1> last, hls::stream<mcxx_outaxis>& mcxx_outPort) {
  #pragma HLS inline
  mcxx_outaxis axis_word;
  axis_word.data = data;
./fpga-naive-zedboard/HLS-32x32-naive/zedboard/matmul_ait/xilinx/HLS/matmulBlock_hw/matmulBlock_hw_hls_automatic_clang.cpp   39,1   25%
    ap_wait();
  }
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
  }
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
  matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
  }
```

- Kernel is the same
- Local copies and management in the wrapper

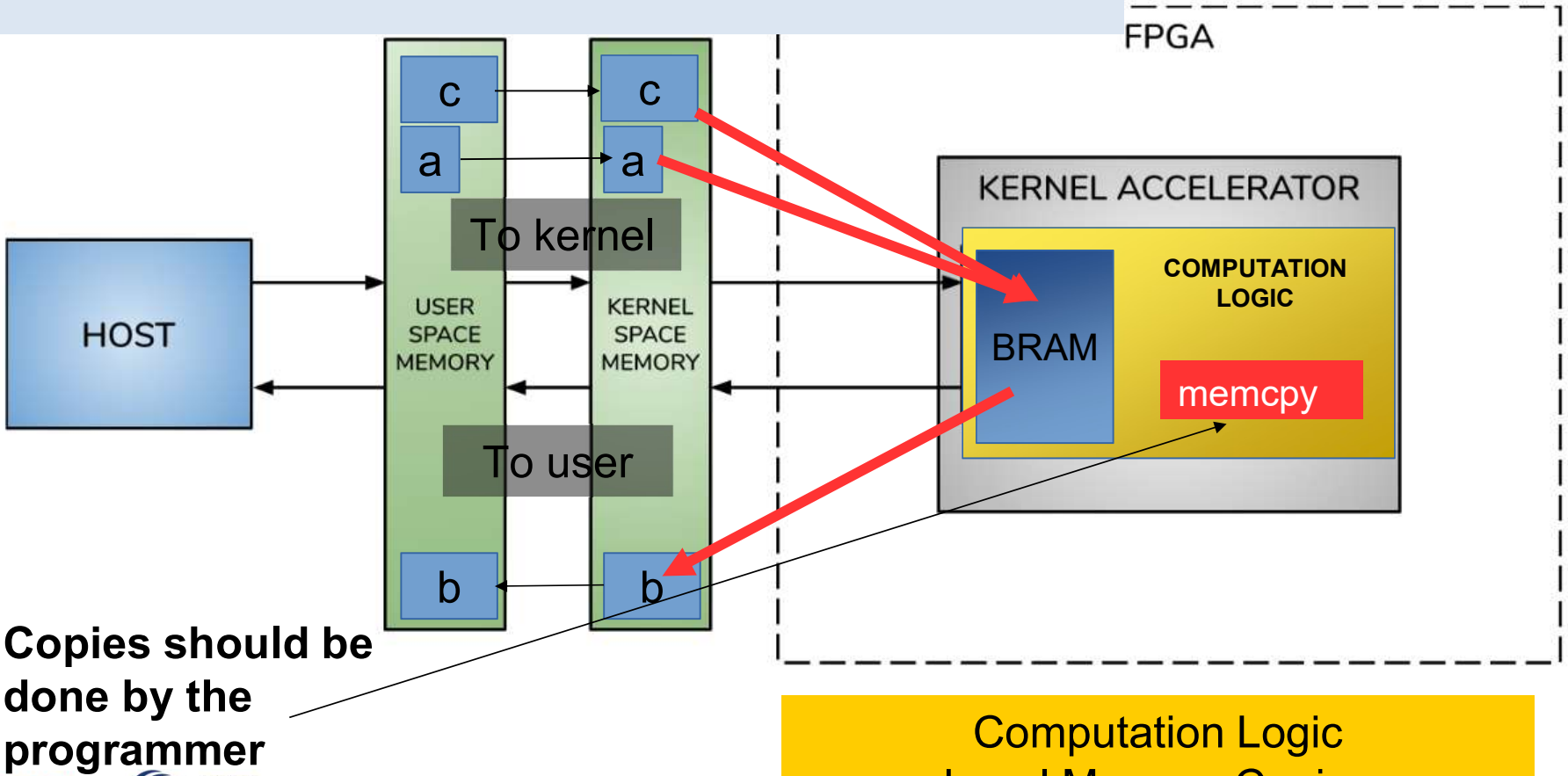
- Local copies using ports

```
void matmulBlock_hw_wrapper(hls::stream<ap_uint<
rData_t>& mcxx_instr) {
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface axis port=mcxx_inPort
#pragma HLS interface axis port=mcxx_outPort
#pragma HLS interface m_axi port=mcxx_a
#pragma HLS interface m_axi port=mcxx_b
#pragma HLS interface m_axi port=mcxx_c
#pragma HLS interface axis port=mcxx_instr
  static elem_t a[1024];
  static elem_t b[1024];
  static elem_t c[1024];
```

Local declaration for local copies

# Data Transfer w/o Wrapper Copies: scale

```
const int SIZE=1024;
#pragma oss task device(fpga) in([SIZE]c, a) out([SIZE]b) num_instances(1)
void scale_task(double *b, double *c, double *a);
```




- **Copies should be done by the programmer**

Computation Logic
Local Memory Copies

# Execution Model (Data Transfer): MxM example

- Arguments with no copies
  - **Local variables are pointers**
  - **Addr is automatically passed in**
  - **Addr transparent access to SMP Memory using AXI ports**

```
void matmulFull_hw_wrapper(hls::stream<ap_uint<64> >& mcxx_
_c, hls::stream<__mcxx_instrData_t>& mcxx_instr) {
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface axis port=mcxx_inPort
#pragma HLS interface axis port=mcxx_outPort
#pragma HLS interface m_axi port=mcxx_mmport
#pragma HLS interface m_axi port=mcxx_a
#pragma HLS interface m_axi port=mcxx_b
#pragma HLS interface m_axi port=mcxx_c
#pragma HLS interface axis port=mcxx_instr
  ap_uint<64> __command = mcxx_inPort.read(); //command wor
  if (__command(7,0) == 2) {
    __mcxx_instrData_t tmpSetup;
    tmpSetup(63,0) = mcxx_inPort.read();
    tmpSetup(79,64) =  (__command>>8)&0xFFFFFF;
    tmpSetup[104] = 0;
    mcxx_instr.write(tmpSetup);
    return;
  }
  __mcxx_taskId = mcxx_inPort.read();
  ap_uint<64> __mcxx_parent_taskId = mcxx_inPort.read();
  int msize;
  elem_t *a;
  elem_t *b;
  elem_t *c;
  {
  #pragma HLS protocol fixed
    {
      ap_uint<8> mcxx_flags_0;
      ap_uint<64> mcxx_offset_0;
      mcxx_flags_0 = mcxx_inPort.read()(7,0);
      ap_wait();
      __mcxx_cast<int> mcxx_arg_0;
      mcxx_arg_0.raw = mcxx_inPort.read();
      msize = mcxx_arg_0.typed;
    }
    ap_wait();
```

```
    ap_wait();
  }
mcxx_instrument_event(70, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
  }
mcxx_instrument_event(206, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(80, __mcxx_taskId, mcxx_instr);
  matmulBlock_hw(a, b, c, mcxx_instr);
mcxx_instrument_event(208, __mcxx_taskId, mcxx_instr);
mcxx_instrument_event(79, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_2[5]) {
    memcpy(mcxx_c + mcxx_offset_2/sizeof(elem_t), c, (4096));
```

```
ap_wait();
{
  ap_uint<8> mcxx_flags_1;
  ap_uint<64> mcxx_offset_1;
  mcxx_flags_1 = mcxx_inPort.read()(7,0);
  ap_wait();
  mcxx_offset_1 = mcxx_inPort.read();
  a = mcxx_a + mcxx_offset_1/sizeof(elem_t);
}
```

# Remember: WIDE PORT FPGA compilation into OmpSs

# Compilation – Wide Port

Few details on the steps to compile

- Compiler: use clang or clang++ (C++)
  - » compiler -fompss-2 -o program program.c
  - » compiler -fompss-2 -fompss-fpga-instrumentation \
  - » **-fompss-fpga-memory-port-width 128** -o program program.c

- Details...
  - » AIT options:
    - --board=$(BOARD_NAME),--clock=100
    - --name=vivado_project_name,--dir=$(VIVADO_WORKSPACE)”

# Wrapper Copies and Wide Port

```
    ap_wait();
  }
mcxx_instrument_event(78, __mcxx_taskId, mcxx_instr);
  if (mcxx_flags_0[4]) {
    memcpy(a, mcxx_a + mcxx_offset_0/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_1[4]) {
    memcpy(b, mcxx_b + mcxx_offset_1/sizeof(elem_t), (4096));
  }
  if (mcxx_flags_2[4]) {
    memcpy(c, mcxx_c + mcxx_offset_2/sizeof(elem_t), (4096));
  }
```

- Memcopy is transformed
  - Wide copy
  - Then, split elements

```
if (mcxx_flags_0[4]) {
  for (int __i = 0; __i < ((4096) - 1)/sizeof(ap_uint<128>)+1; ++__i) {
  #pragma HLS pipeline II=1
    ap_uint<128> __tmpBuffer;
    __tmpBuffer = *(mcxx_memport + mcxx_offset_0/sizeof(ap_uint<128>) + __i);
    for (int __j=0; __j <(sizeof(ap_uint<128>)/4); __j++) {
      __mcxx_cast<elem_t> cast_tmp;
      cast_tmp.raw = __tmpBuffer((__j+1)*4*8-1,__j*4*8);
      a[__i*(sizeof(ap_uint<128>)/4)+__j] = cast_tmp.typed;
    }
  }
}
```

# FPGA Blocking: Avoid unnecessary copies...

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
inout([msize*msize]c) num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      memcpy(cc,&c[ci],BSIZE*BSIZE*sizeof(elem_t));
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         memcpy(aa,&a[ai],BSIZE*BSIZE*sizeof(elem_t));
         memcpy(bb,&b[bi],BSIZE*BSIZE*sizeof(elem_t));
         matmulBlock_hw(aa, bb, cc);
      }
      memcpy(&c[ci],cc,BSIZE*BSIZE*sizeof(elem_t));
    }
   }
}
```

# User data copy without wide port

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
inout([msize*msize]c) num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      memcpy(cc,&c[ci],BSIZE*BSIZE*sizeof(elem_t));
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         memcpy(aa,&a[ai],BSIZE*BSIZE*sizeof(elem_t));
         memcpy(bb,&b[bi],BSIZE*BSIZE*sizeof(elem_t));
         matmulBlock_hw(aa, bb, cc);
      }
      memcpy(&c[ci],cc,BSIZE*BSIZE*sizeof(elem_t));
    }
   }
}
```

# User data copy w/ wide port

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
                                                    inout([msize*msize]c) num_instances(1)

void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

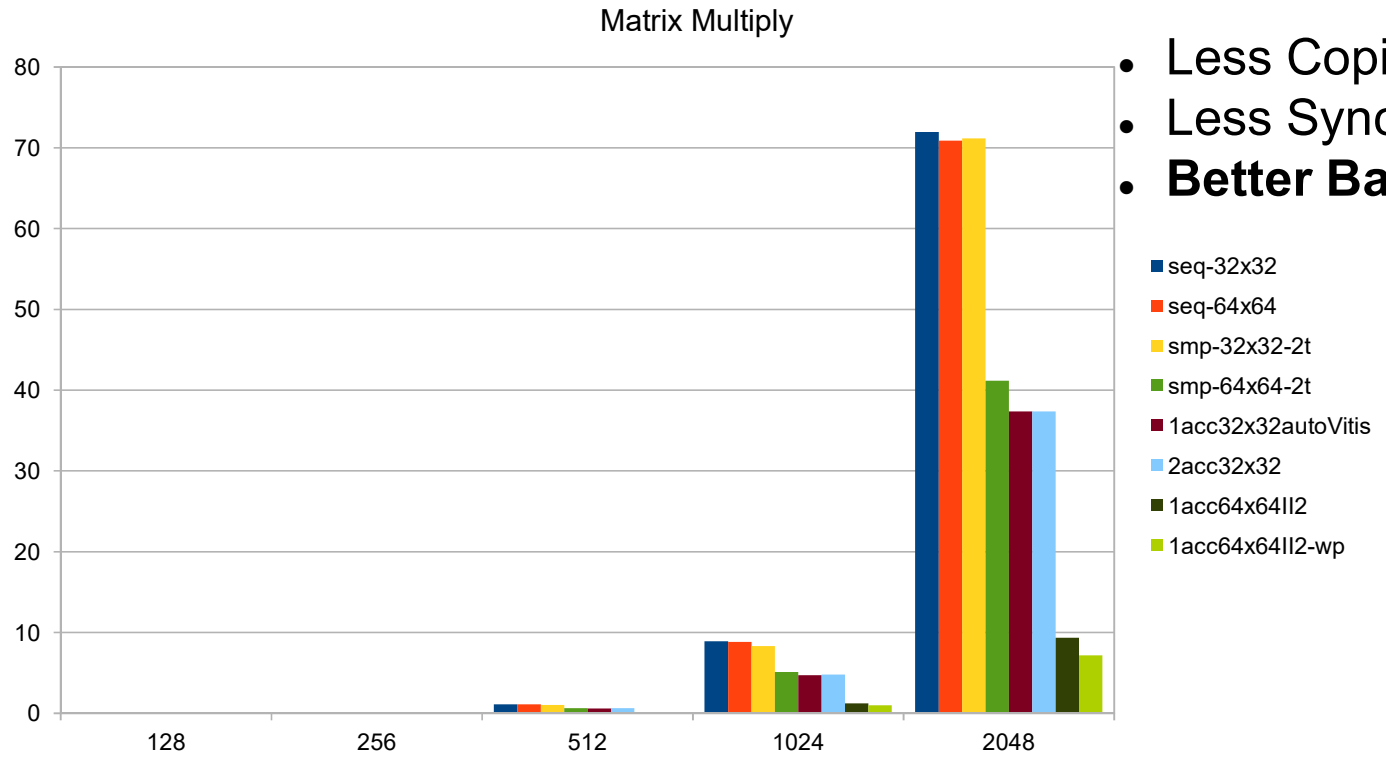
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      nanos6_fpga_memcpy_wideport_in(cc, (unsigned long long int)(&c[ci]),BSIZE*BSIZE);
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         nanos6_fpga_memcpy_wideport_in(aa,(unsigned long long int)&a[ai],BSIZE*BSIZE);
         nanos6_fpga_memcpy_wideport_in(bb,(unsigned long long int)&b[bi],BSIZE*BSIZE);
         matmulBlock_hw(aa, bb, cc);
      }
      nanos6_fpga_memcpy_wideport_out(cc,(unsigned long long int)&c[ci],BSIZE*BSIZE);
    }
   }
}
```

- Local variables
- nanos6 Wide Port Copies

```
#include <nanos6/fpga_device.h>

nanos6_fpga_memcpy_wideport_in
nanos6_fpga_memcpy_wideport_out
```

# What is the performance of the Acc version?




- Less Copies
- Less Synchronizations
- **Better Bandwidth!**

**Question:**

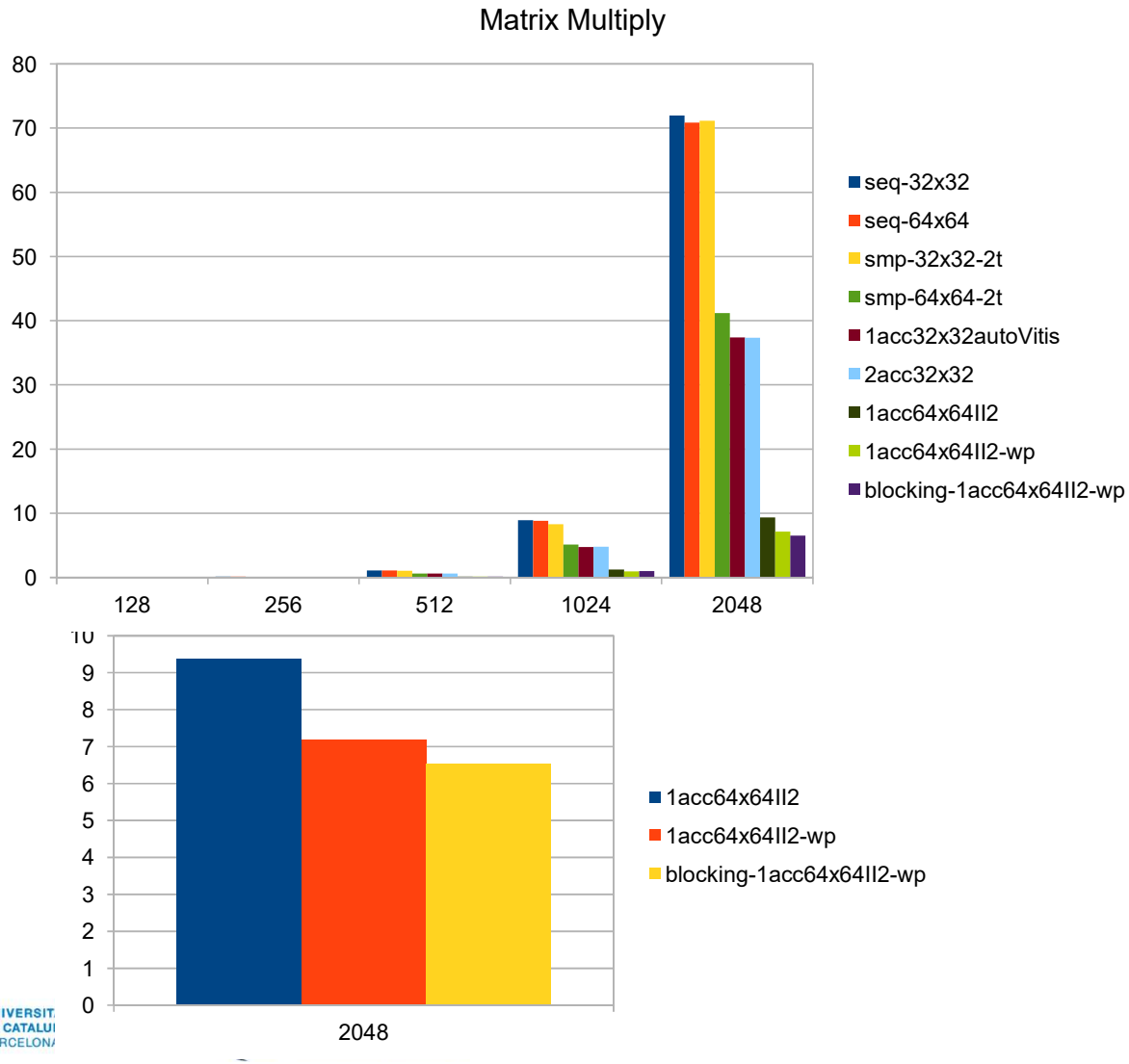
- **Can we improve it again?**

# MxM w/ blocking w/ task creation…

```
#pragma oss task device(fpga) in([CONST_BSIZE*CONST_BSIZE]a, [CONST_BSIZE*CONST_BSIZE]b)
inout([CONST_BSIZE*CONST_BSIZE]c) copy_deps num_instances(1)
void matmulBlock_hw(elem_t a[CONST_BSIZE*CONST_BSIZE], elem_t b[CONST_BSIZE*CONST_BSIZE],
elem_t c[CONST_BSIZE*CONST_BSIZE]);
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) inout([msize*msize]c)
num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;

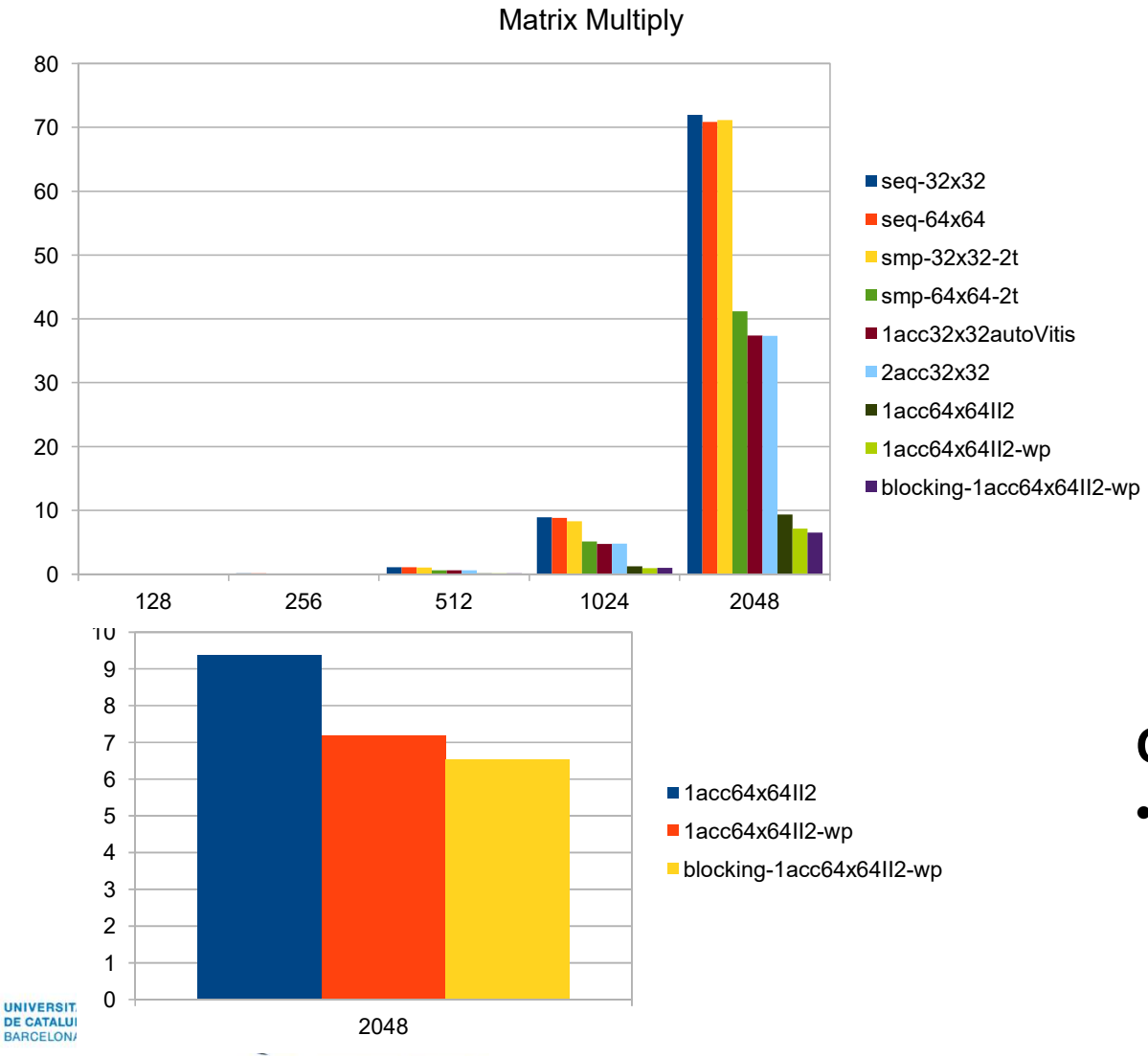
   for (unsigned int i = 0; i < msize/BSIZE; i++) {
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
            for (unsigned int j = 0; j < msize/BSIZE; j++) {
            unsigned int const ci = j*b2size + i*BSIZE*msize;
            unsigned int const bi = j*b2size + k*BSIZE*msize;
            matmulBlock_hw(a+ai, b+bi, c+ci);
   }}}
   #pragma oss taskwait
```

# What is the performance of the Acc version?




- Less Copies
- Less Synchronizations
- **Better Bandwidth**
- **Faster Creation**

# What is the performance of the Acc version?




- Less Copies
- Less Synchronizations
- Better Bandwidth
- Faster Creation

**Question:**

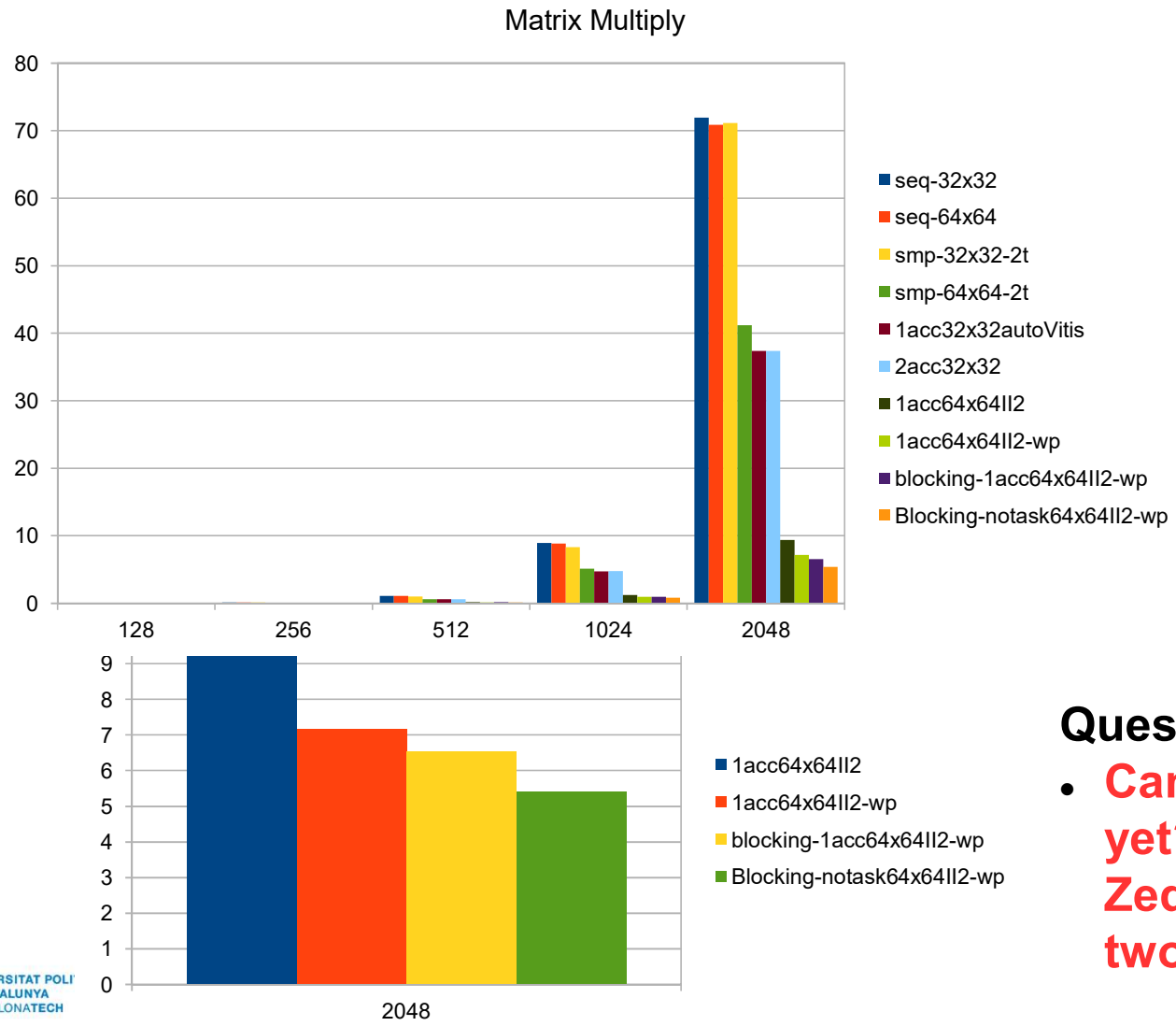
- **Can we improve it again :-)?**

# FPGA Blocking: Avoid unnecessary copies...

```
#pragma oss task device(fpga) in([msize*msize]a, [msize*msize]b) \
inout([msize*msize]c) num_instances(1)
void matmulFull_hw(int msize, elem_t *a, elem_t *b, elem_t *c) {
   unsigned int const m2size = msize*msize;
   unsigned int const b2size = BSIZE*BSIZE;
   elem_t aa[BSIZE*BSIZE];
   elem_t bb[BSIZE*BSIZE];
   elem_t cc[BSIZE*BSIZE];

   for (unsigned int i = 0; i < msize/BSIZE; i++) {
    for (unsigned int j = 0; j < msize/BSIZE; j++) {
      unsigned int const ci = j*b2size + i*BSIZE*msize;
      memcpy(cc,&c[ci],BSIZE*BSIZE*sizeof(elem_t));
      for (unsigned int k = 0; k < msize/BSIZE; k++) {
         unsigned int const ai = k*b2size + i*BSIZE*msize;
         unsigned int const bi = j*b2size + k*BSIZE*msize;
         memcpy(aa,&a[ai],BSIZE*BSIZE*sizeof(elem_t));
         memcpy(bb,&b[bi],BSIZE*BSIZE*sizeof(elem_t));
         matmulBlock_hw(aa, bb, cc);
      }
      memcpy(&c[ci],cc,BSIZE*BSIZE*sizeof(elem_t));
    }
   }
}
```

# What is the performance of the Acc version?

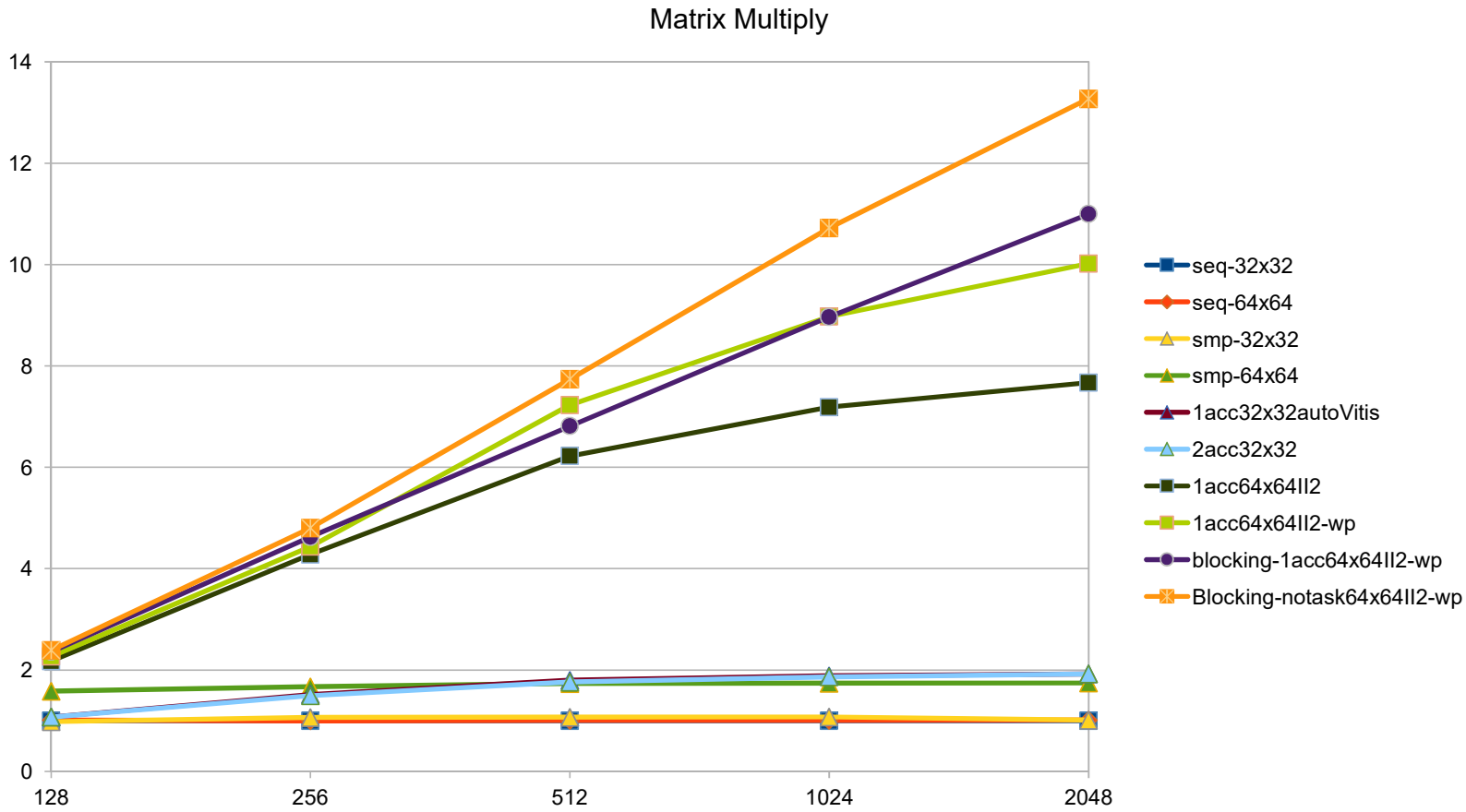



- Less Copies
- Less Synchronizations
- Better Bandwidth
- Faster Creation
- **Avoid Copies**

**Question:**

- **Can we improve it again yet? Affinity… but in Zedboard we can not fit two efficient 64x64**

# What is the performance of the Acc version?



# Vivado/Vitis HLS Optimizations

Thank you!!