SOCAO: Source-to-Source OpenCL Compiler for Intel-Altera FPGAs

Johanna Rohde[1], Marcos Martinez-Peiró[2], Rafael Gadea-Gironés[2]

Date: 7.9.2017

[1]Computer Systems Group, TU Darmstadt

[2]Department of Electronic Engineering, Universitat Politècnica de València

Overview

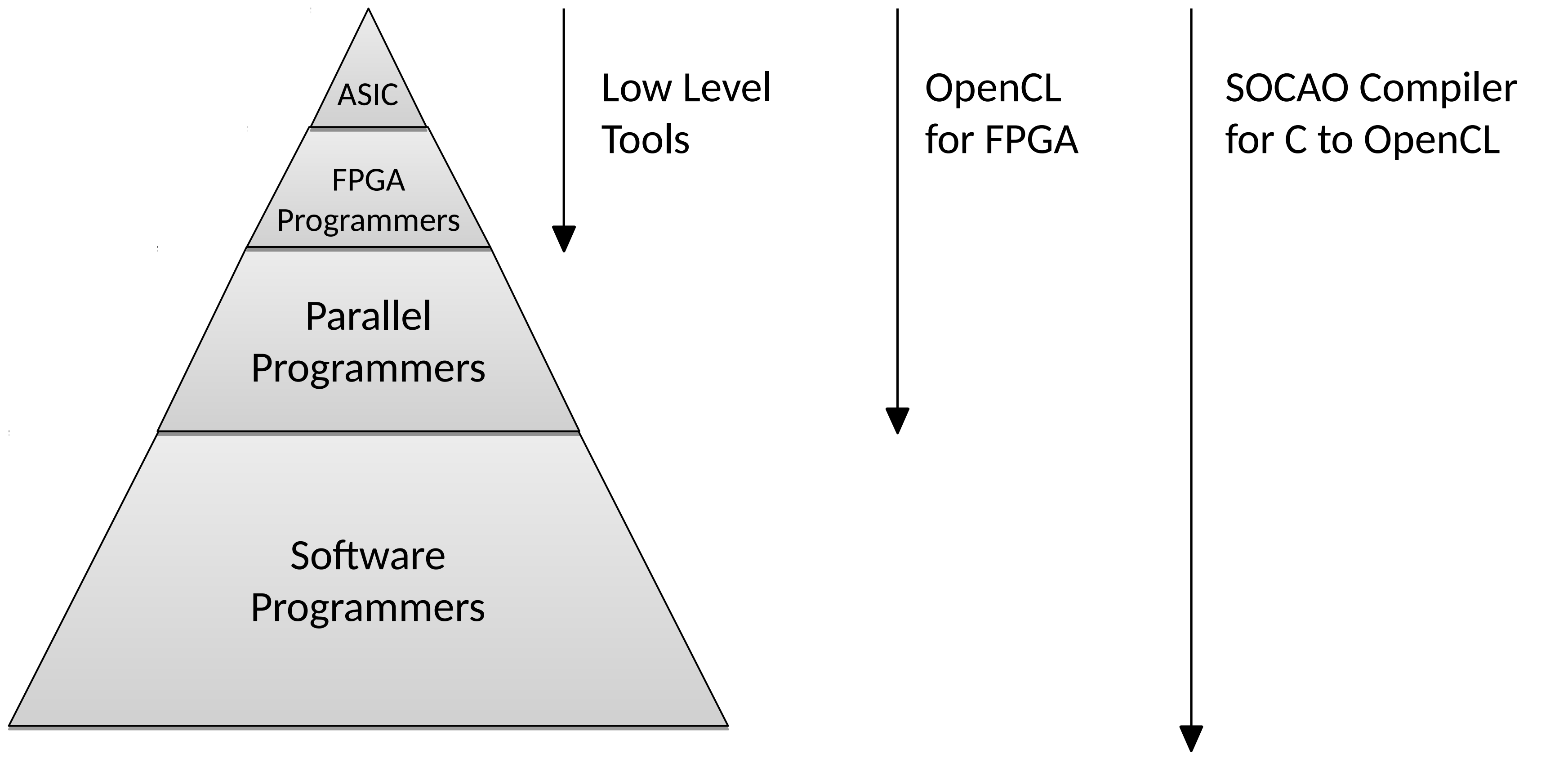
- Introduction
- Background
- Design
- Implementation
- Evaluation
- Conclusion

Introduction

Problem: Accelerate a program with an FPGA

- How do I program the FPGA?
- How do I communicate with the FPGA?
- How much time do I need to rewrite the code?

Introduction

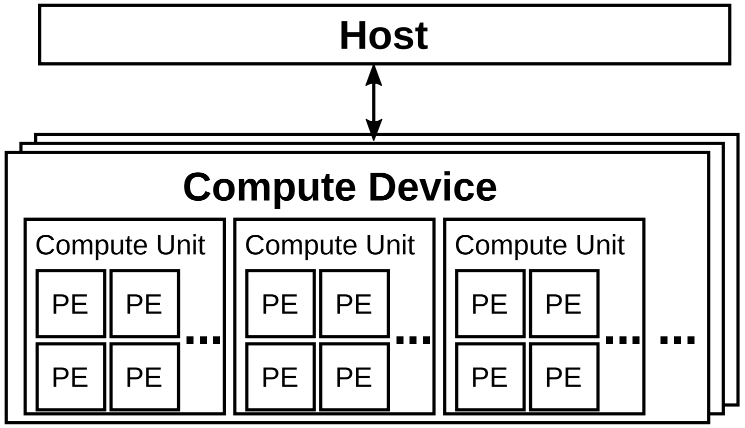


Background

- OpenCL
  - Open programming standard for heterogeneous parallel systems
  - Calculations are passed to external accelerator
  - Accelerator can be
    - CPU
    - GPU
    - FPGA
    - ...

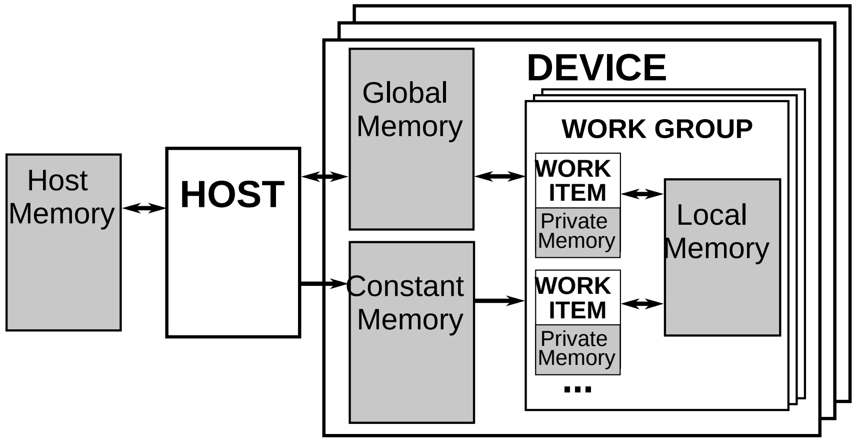
Background
OpenCL

- Platform
  - Host
    - Manages the system
    - Is connected to one or more *compute devices*
  - Compute Device
    - Executes a kernel
    - Consists of multiple *compute units*
  - Compute Unit
    - Consists of multiple *processing elements*



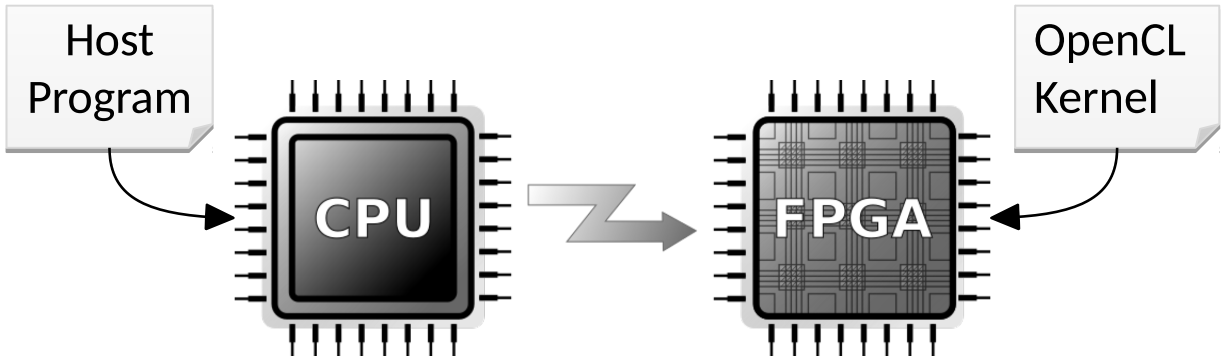
Background
OpenCL

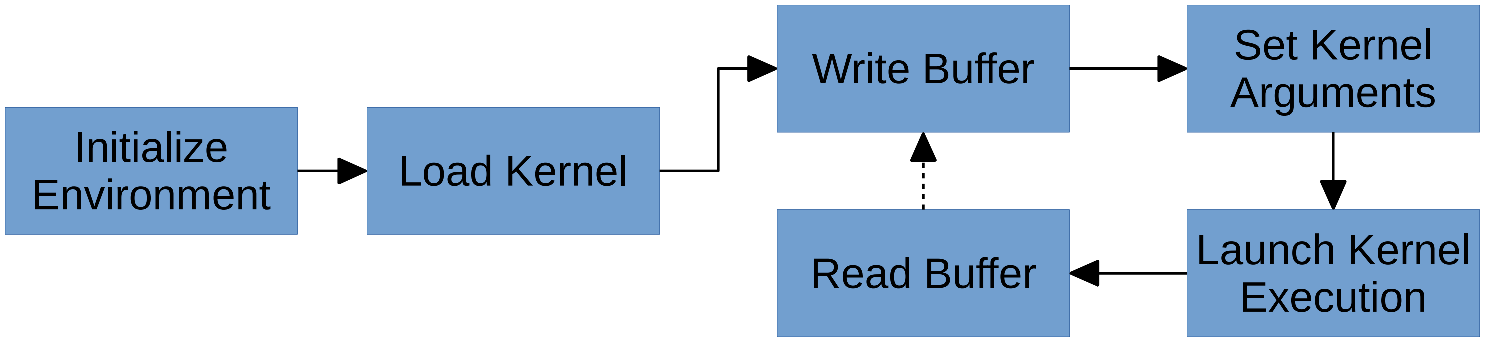
- Memory Model
  - Global Memory
    - Used to transfer data
    - Accessible by all work groups
    - Normally the slowest memory
  - Constant Memory
    - Used to save constants
  - Local Memory
    - Accessible by all work items of one work group
    - Not accessible by host
  - Private Memory
    - Accessible by one work item
    - Holds intermediate values



Background
OpenCL

- Host Program Flow

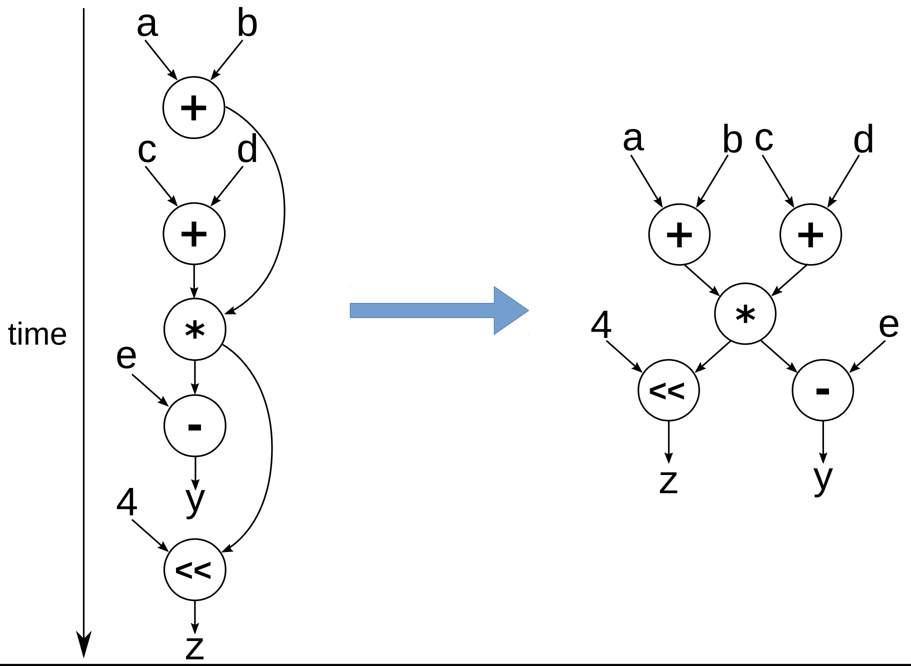




Background
OpenCL for FPGAs

- 2 additional forms of parallelism
- Instruction-level parallelism
  - Instructions that are independent of each other can be calculated at the same time

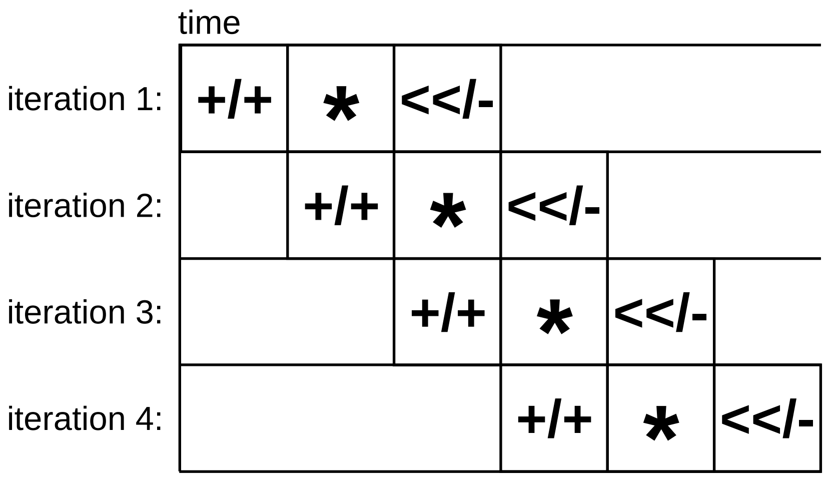
```
x = (a+b)*(c+d);
y = x - e;
z = x << 4;
```



Background
OpenCL for FPGAs

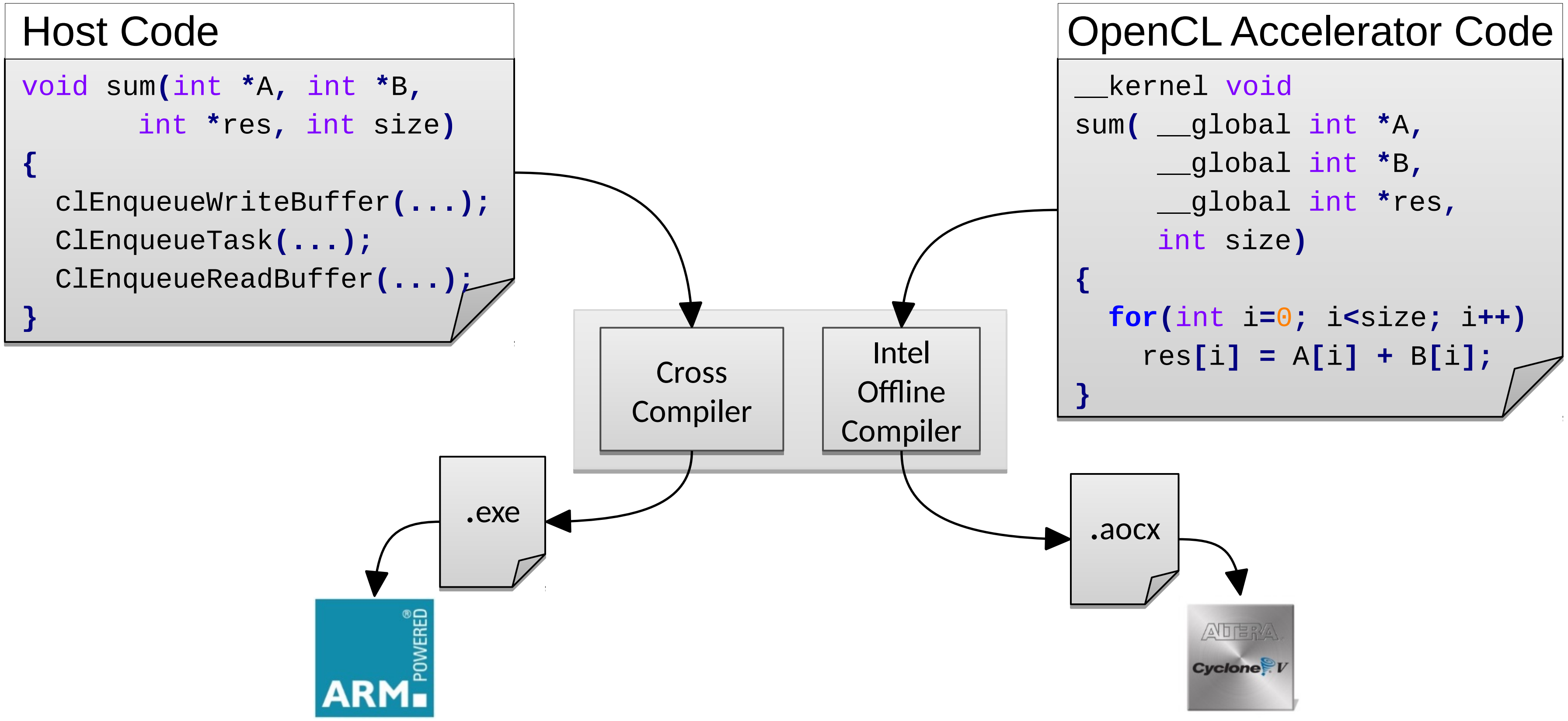
- Loop Pipelining
  - Iterations are overlapped
  - Ideal case: One Iteration per clock cycle
  - Problem when the loop has loop-carried dependencies

```
for(int a=0; a<4; a++)
{
    x = (a+b)*(c+d);
    y = x << 4;
    z = x - e;
}
```



Background
OpenCL for FPGAs

- Intel’s SDK for OpenCL



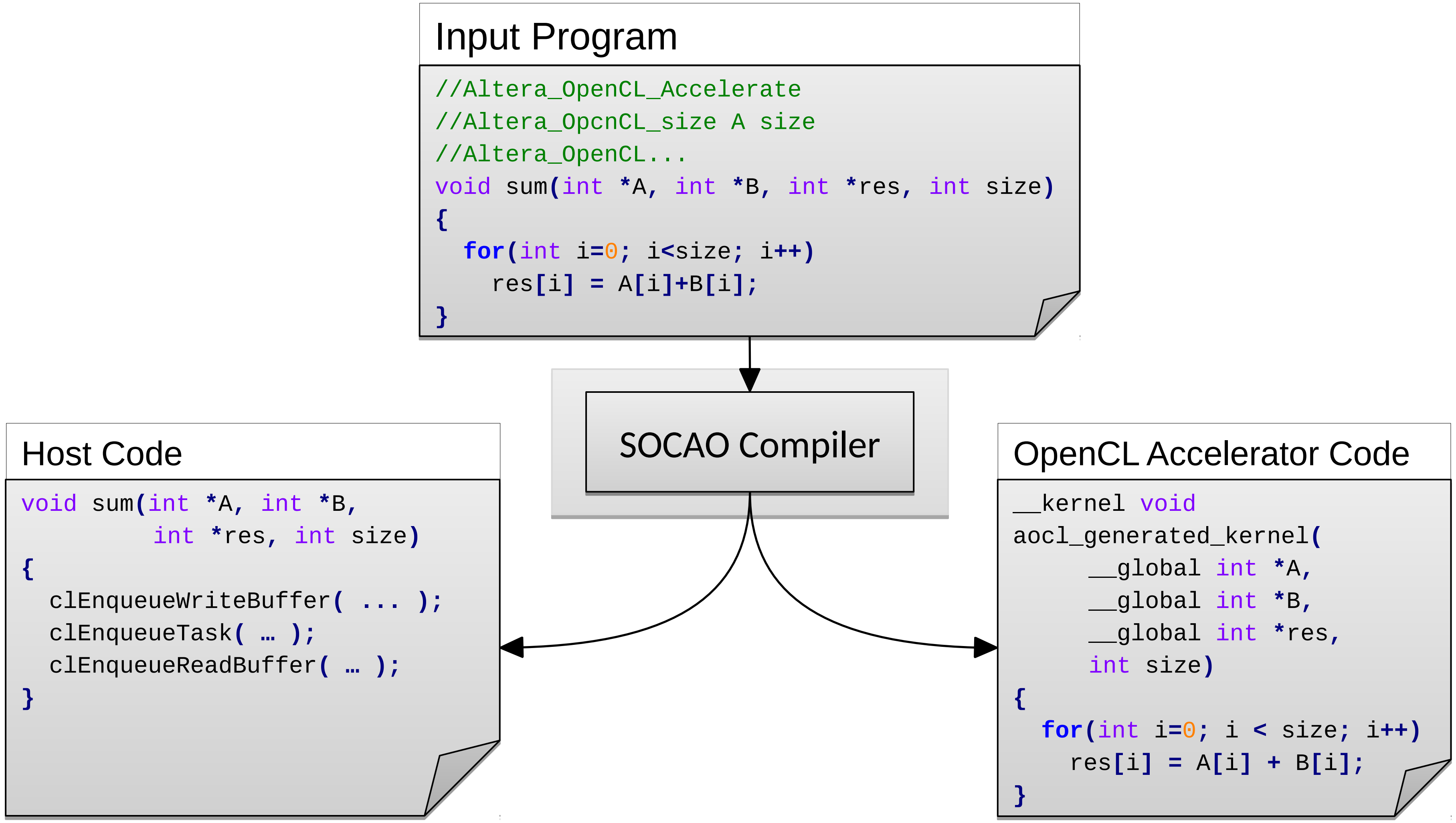
Host Code

```
void sum(int *A, int *B,
        int *res, int size)
{
  clEnqueueWriteBuffer(...);
  ClEnqueueTask(...);
  ClEnqueueReadBuffer(...);
}
```

OpenCL Accelerator Code

```
__kernel void
sum( __global int *A,
     __global int *B,
     __global int *res,
     int size)
{
  for(int i=0; i<size; i++)
    res[i] = A[i] + B[i];
}
```

Design



Input Program

```
//Altera_OpenCL_Accelerate
//Altera_OpcnCL_size A size
//Altera_OpenCL...
void sum(int *A, int *B, int *res, int size)
{
  for(int i=0; i<size; i++)
    res[i] = A[i]+B[i];
}
```

SOCAO Compiler

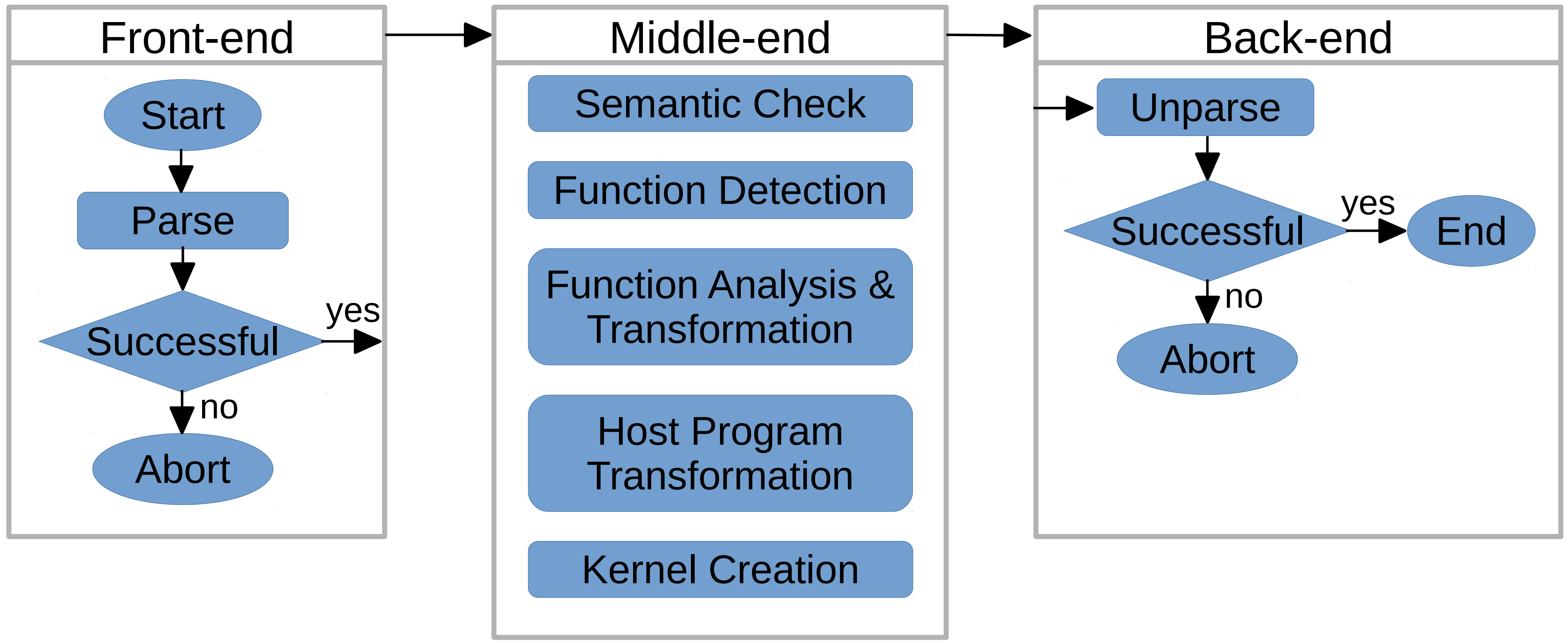
Host Code

```
void sum(int *A, int *B,
         int *res, int size)
{
  clEnqueueWriteBuffer( ... );
  clEnqueueTask( … );
  clEnqueueReadBuffer( … );
}
```

OpenCL Accelerator Code

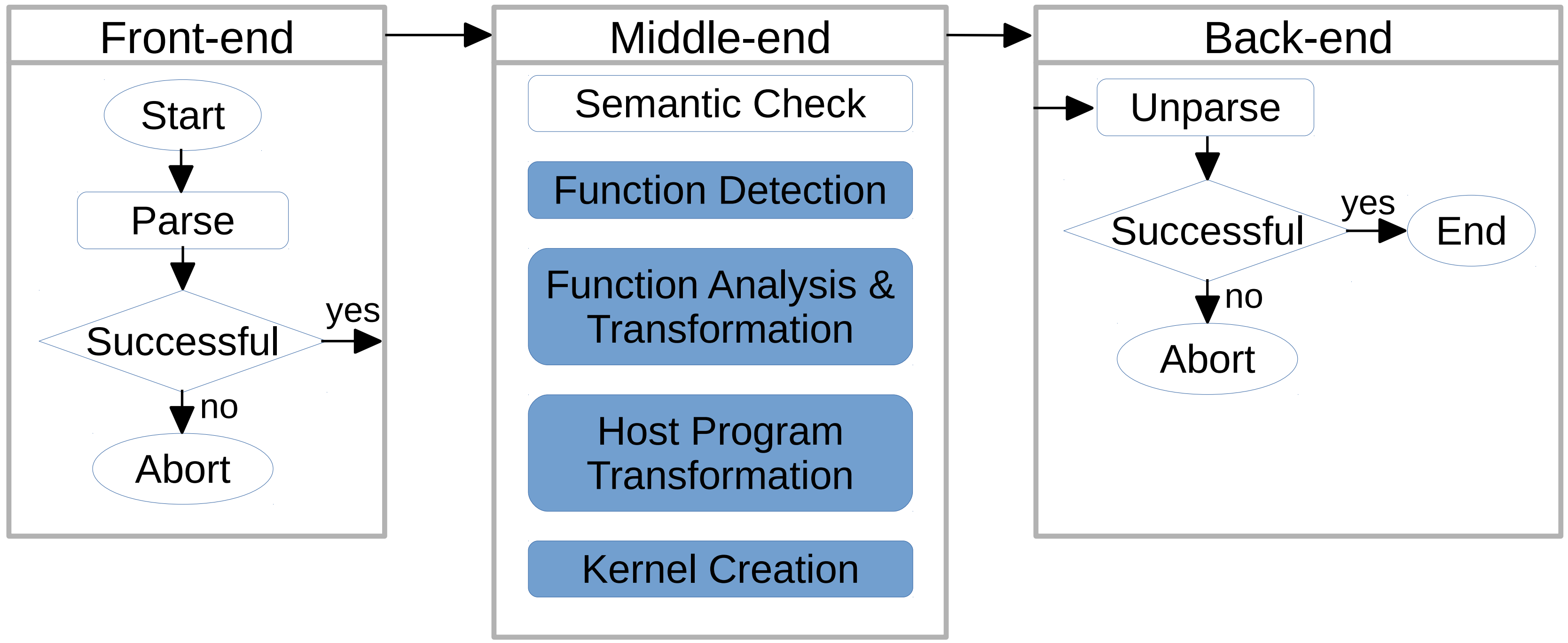
```
__kernel void
aocl_generated_kernel(
     __global int *A,
     __global int *B,
     __global int *res,
     int size)
{
  for(int i=0; i < size; i++)
    res[i] = A[i] + B[i];
}
```

Implementation



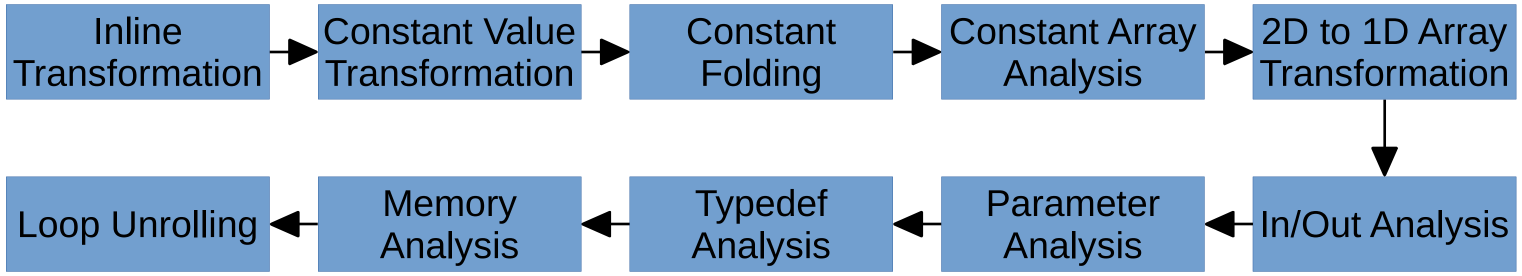
- ROSE Framework
  - Open-source compiler framework
  - Provides front-end, back-end and additional functionalities

Implementation



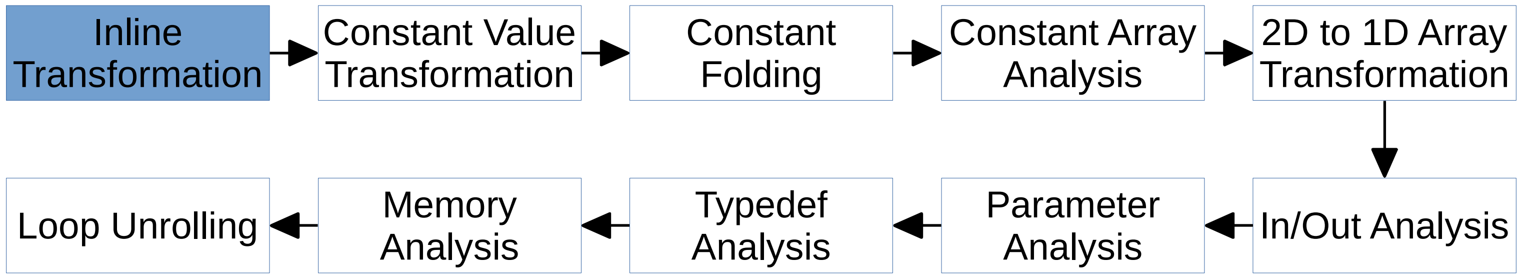
- ROSE Framework
  - Open-source compiler framework
  - Provides front-end, back-end and additional functionalities

Implementation



- The *Function Analysis & Transformation* phase is the most important
- All decisions are made during this phase
- Consists of 10 analysis/transformation steps

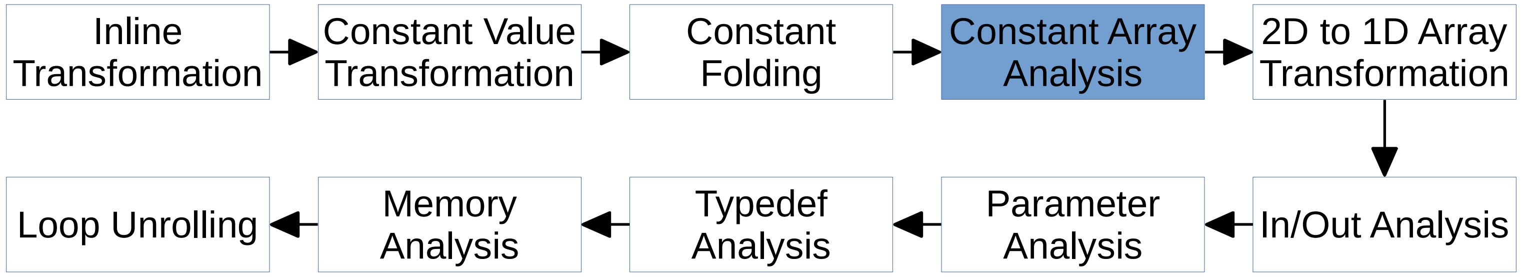
Implementation



```c
void vector_process( char *input,
                     char value)
{
    int i;
    for(i = 0; i < 64; i++)
        input[i] += value;
}

void vector_update(char *input,
                   int ilen)
{
    vector_process(input, 'c');
}
```

```c
void vector_update(char *input,
                   int ilen)
{
    {
        char value_1 = 'c';
        int i;
        for(i = 0; i < 64; i++)
            input[i] += value_1;
    }
}
```

Implementation



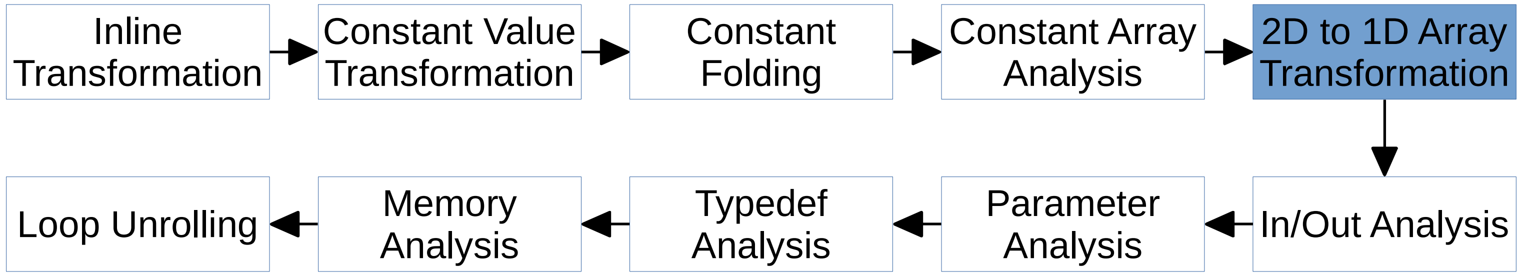
program.cpp

```
const uint32_t K[] = {0x428A2F98,
 …};

//Altera_OpenCL_Accelerate
//Altera_OpenCL_size K 64
//Altera_OpenCL_const_vec K
…
void update_accelerated( … )
```

aocl_kernel.cl

```
__constant const uint32_t K[] = {
0x428A2F98,  …};

__kernel void
aocl_generated_kernel( …. )
{
    …
}
```

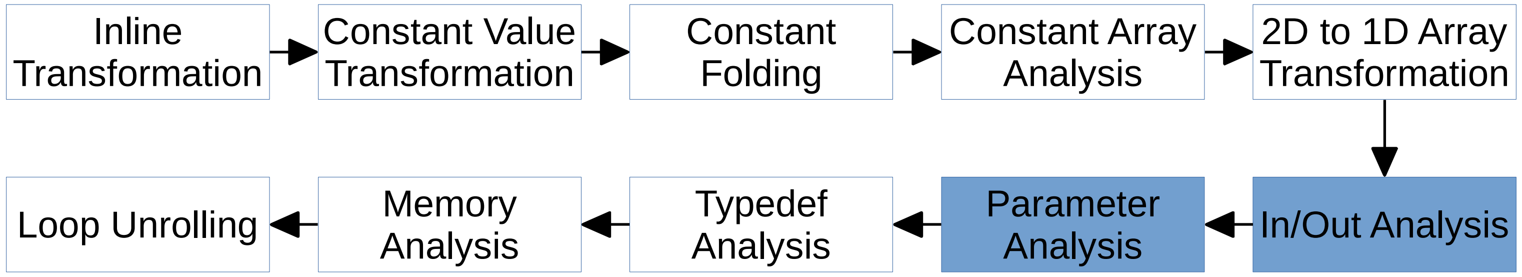
Implementation



```
#define WIDTH 512
int[5][WIDTH] A;

//Altera_OpenCL_Accelerate
//Altera_OpenCL_size A 2560
void func(...)
{
    int tmp = A[2][3];
}
```

```
#define WIDTH 512
__kernel void
aocl_generated_kernel(int
__global __restrict__ *A, ...)
{
    ...
    int tmp = A[2*WIDTH + 3];
}
```

Implementation

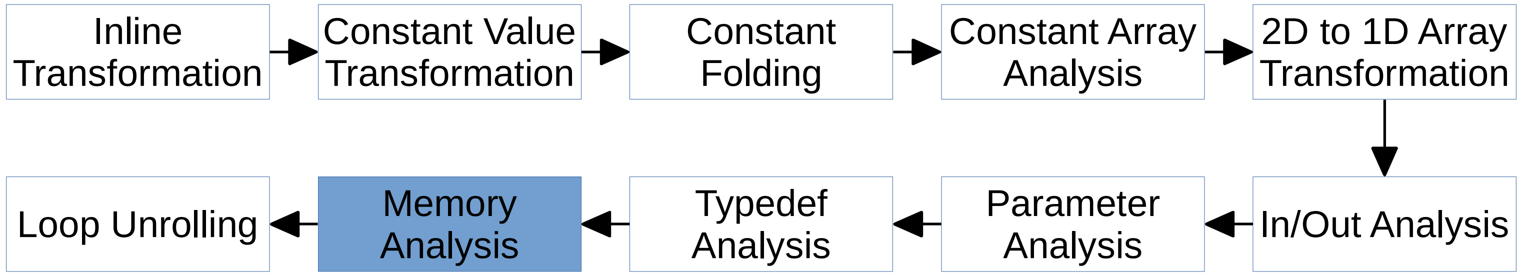


```
int *res;

//Altera_OpenCL_Accelerate
//Altera_OpenCL_size a size
//Altera_OpenCL_size res size
void vec_accumulate(int *a, int
size)
{
    for(int i = 0; i < size; i++)
        res[i] = res[i]+a[i];
}
```

Input Variables:	{i, size, res, a}
Output Variables:	{i, res}
Kernel Parameter:	{a, size, res}

```
__kernel void aocl_generated_kernel(
    int __global __restrict__ *a,
    const int size,
    int __global __restrict__ *res );
```

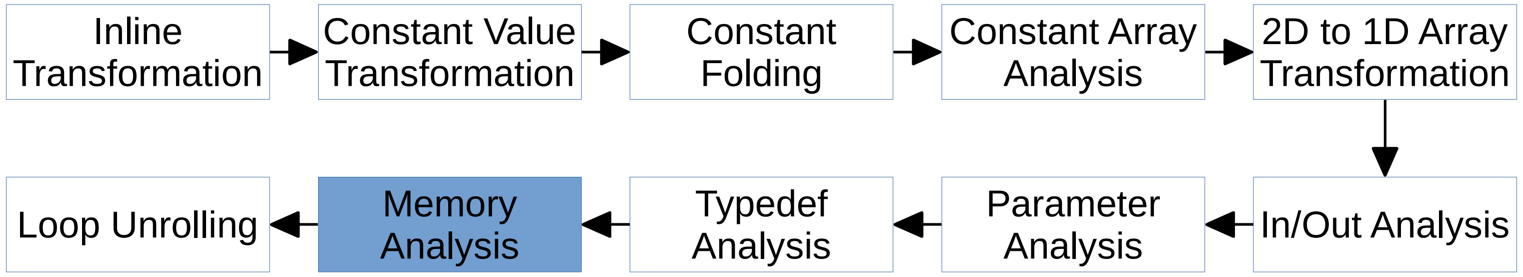
Implementation

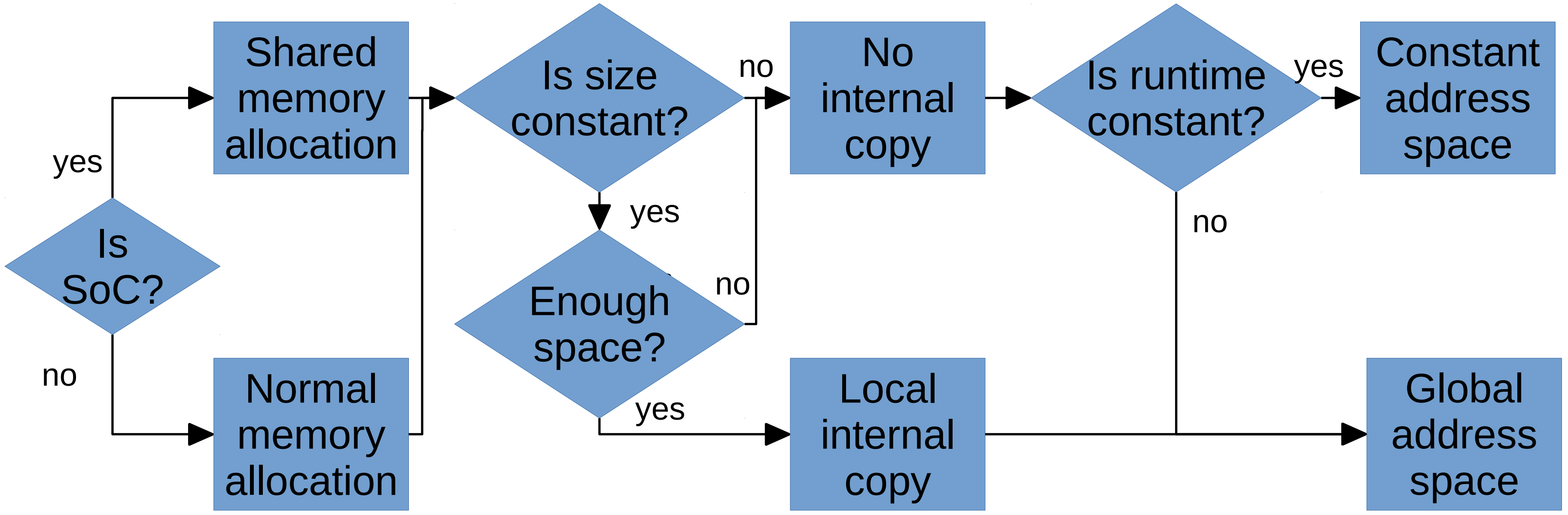


- Determine which memory buffer is used

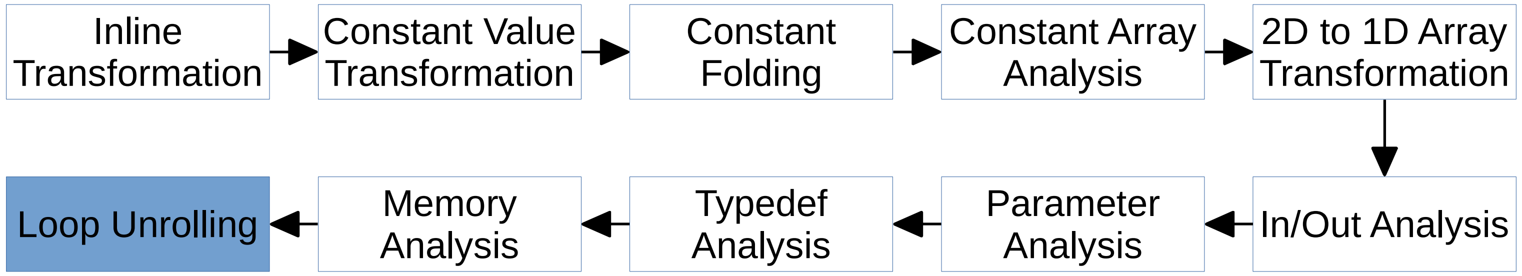
Transfer	Allocation	Internal
Constant	Normal	No copy
	Shared	No copy
Global	Normal	No copy
		Local copy
	Shared	No copy
		Local copy

Implementation





Implementation



- Insert `#pragma unroll` in front of a loop to unroll it
- Only unroll inner most loop
- Exclusion criteria
  - Number of iterations is not static
  - Number of iterations exceeds 16
  - The loop contains an operation that requires a lot of area

```
#pragma unroll
for (j = 0; j < 8; j++)
    A[j] = state[j];
```

Evaluation

- DE1SoC evaluation board
  - ARM Cortex
    - Dual Core
    - 800 MHz
  - Cyclone V FPGA

Evaluation

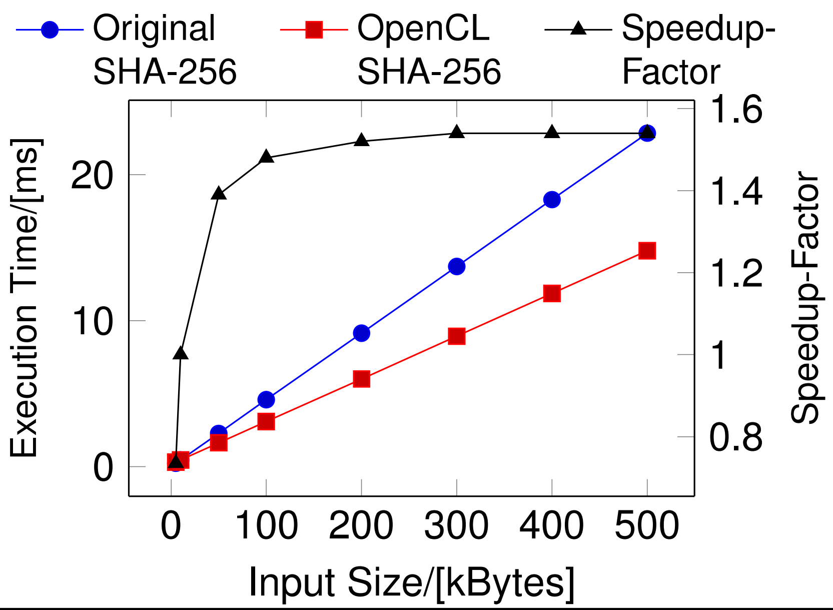
Secure Hash Algorithm (SHA-256)

- Contains two nested for loops
- Both loops have loop-carried dependencies
- Hard to parallelize
- Has a lot of instruction parallelism

```
//Altera_OpenCL_Accelerate
//Altera_OpenCL_size input len
//Altera_OpenCL_size K 64
//Altera_OpenCL_size state 8
//Altera_OpenCL_const_vec K
//Altera_OpenCL_soc
void mbedtls_sha256_update_accelerated( uint32_t *state,
                             const unsigned char *input,
                             uint64_t len )
```

Evaluation
Secure Hash Algorithm (SHA-256)

- Inner loop is pipelined perfectly
- Outer loop is not pipelined well
- Maximum speedup-factor: 1,54
- Break even point: 10kBytes



Evaluation

Advanced Encryption Standard (AES-CTR)

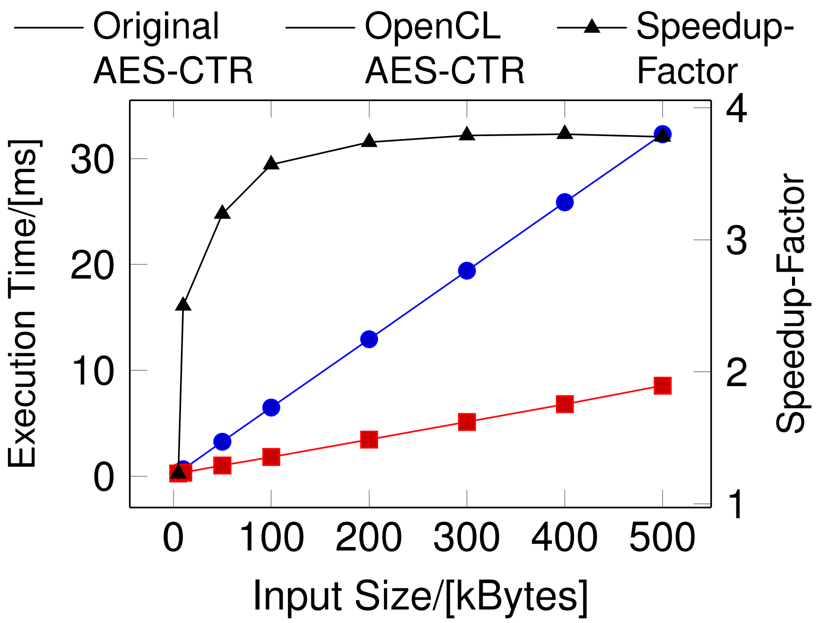
- Contains two nested loops
- Inner loop has loop-carried dependencies and is unrolled
- Uses 10 constant arrays
- Loop Unrolling

```
//Altera_OpenCL_Accelerate
//Altera_OpenCL_size input length
//Altera_OpenCL_size output length
//Altera_OpenCL_size RK 68
//Altera_OpenCL_const_vec FT3
//Altera_OpenCL_const_vec FT2
//Altera_OpenCL_const_vec FT1
//Altera_OpenCL_const_vec FT0
//Altera_OpenCL_const_vec RT3
//Altera_OpenCL_const_vec RT2
//Altera_OpenCL_const_vec RT1
//Altera_OpenCL_const_vec RT0
//Altera_OpenCL_const_vec FSb
//Altera_OpenCL_const_vec RSb
//Altera_OpenCL_soc
int mbedtls_aes_crypt_ctr_nr10(
        uint32_t *RK,
        uint64_t length,
        unsigned char nonce_counter[16],
        unsigned char stream_block[16],
        const unsigned char *input,
        unsigned char *output )
```

Evaluation

Advanced Encryption Standard (AES-CTR)

- Remaining loop can be pipelined perfectly
- Maximum speedup-factor: 3,78
- Break even point: 3,3 kBytes



Conclusion

- C code → OpenCL for FPGAs
- Two test cases
  - SHA-256
    - Speedup of 1.54
  - AES-CTR
    - Speedup 3.78