

# Support for Parallel Scan in OpenMP

E. Maicol G. Zegarra

*maicol.zegarra@students.ic.unicamp.br*

University of Campinas  
Institute of Computing

Advisor

Prof. Dr. Guido Costa Souza de Araújo

Co-advisor

Prof. Dr. Marcio Machado Pereira



24 April 2018

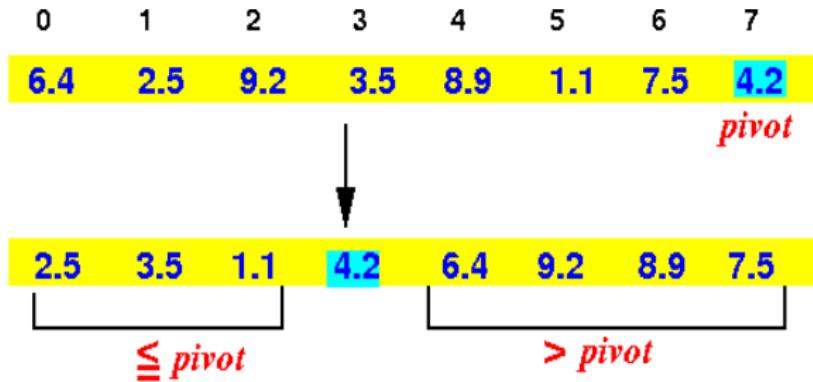
# Table of Contents

- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# Table of Contents

- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# Parallel Quicksort Algorithm



How can we do this in parallel?

# Prefix Sums Problem

In computer science, the prefix sum, cumulative sum, inclusive scan or simply scan is defined as:

- Given a **binary associative operator**  $\oplus$ ;
- An ordered set of  $n$  elements  $[a_0, a_1, \dots, a_{n-1}]$ ;
- Return the ordered set  $[a_0, a_0 \oplus a_1, \dots, a_0 \oplus a_1 \oplus \dots \oplus a_{n-1}]$ .

# Prefix Sums Problem

There are two versions of Prefix Scan: inclusive and exclusive.

[3 1 7 0 4 1 6 3]



Inclusive [3 4 11 11 15 16 22 25]

Exclusive [0 3 4 11 11 15 16 22]

# Prefix Sums Problem

There are two versions of Prefix Scan: inclusive and exclusive.

[3 1 7 0 4 1 6 3]

Inclusive [3 4 11 11 15 16 22 25]

→ Exclusive [0 3 4 11 11 15 16 22]

# Table of Contents

1 Introduction

2 The Algorithm

3 Scan Clause in AClang

- Implementation of Scan in AClang
- The Template

4 Using the Scan Operator

5 Experimental Evaluation

6 Conclusion and Future Works

# Algorithm (Proposed by Horn in 2005 )

Based in Hillis and Steele Work (1986).

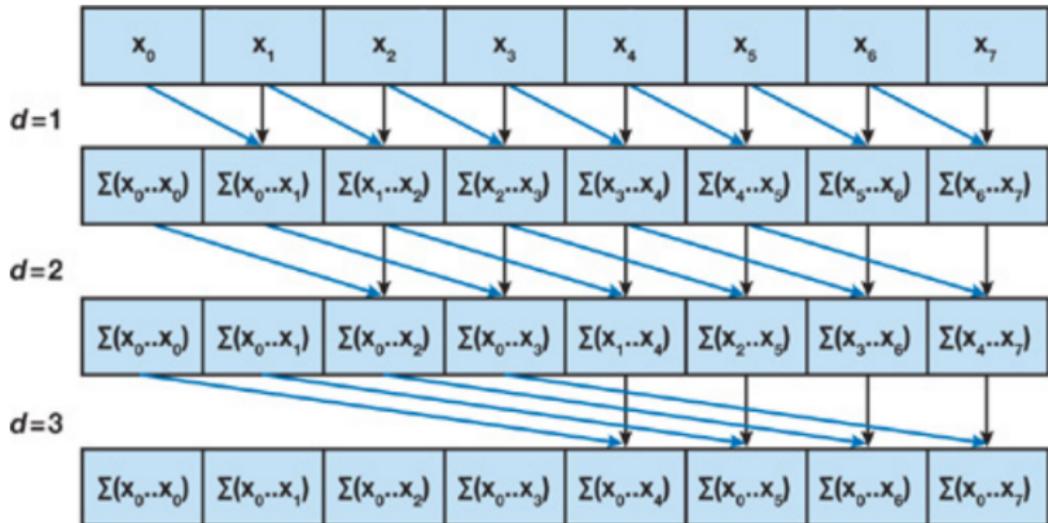


Figure 1: An illustration of the algorithm proposed by Horn

# Complexity Analysis

The algorithm performs  $O(n \log_2 n)$  addition operations.

- The tree has  $\log_2 n$  levels;
- It is performed  $O(n)$  adds for each level;

Remember that a sequential scan performs  $O(n)$  adds. Therefore, this naive implementation is not work-efficient. The factor of  $\log_2 n$  can have a large effect on performance.

# Algorithm (Proposed by Mark Harris in 2007 )

Based in Blelloch's Work (1990);

Two phases (Up-Sweep & Down-Sweep):

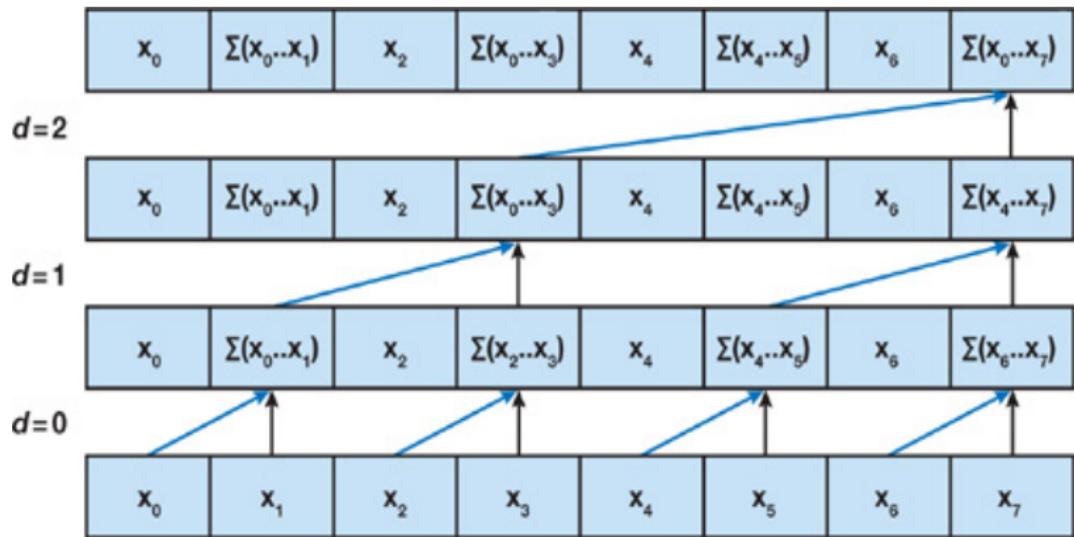
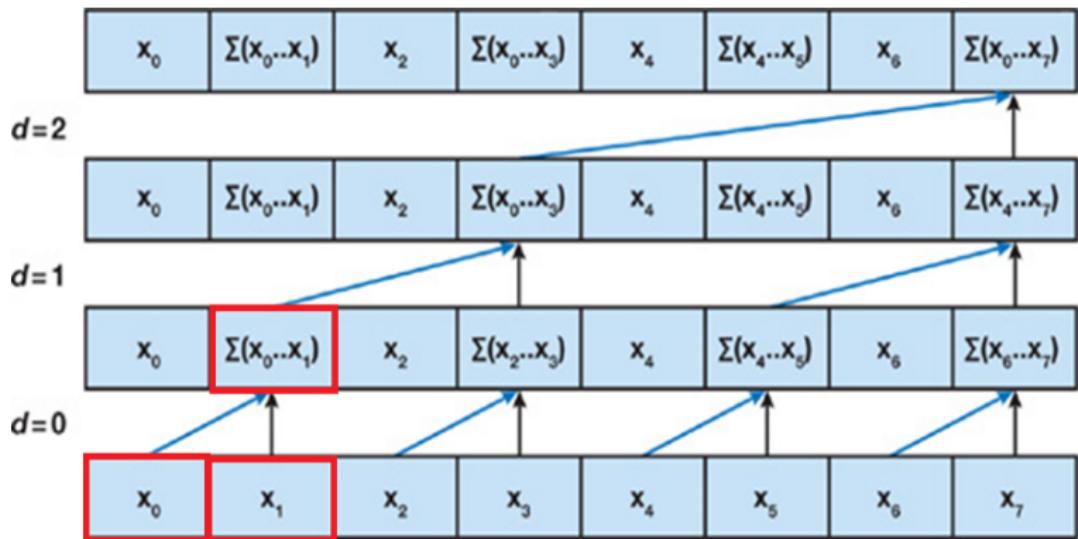
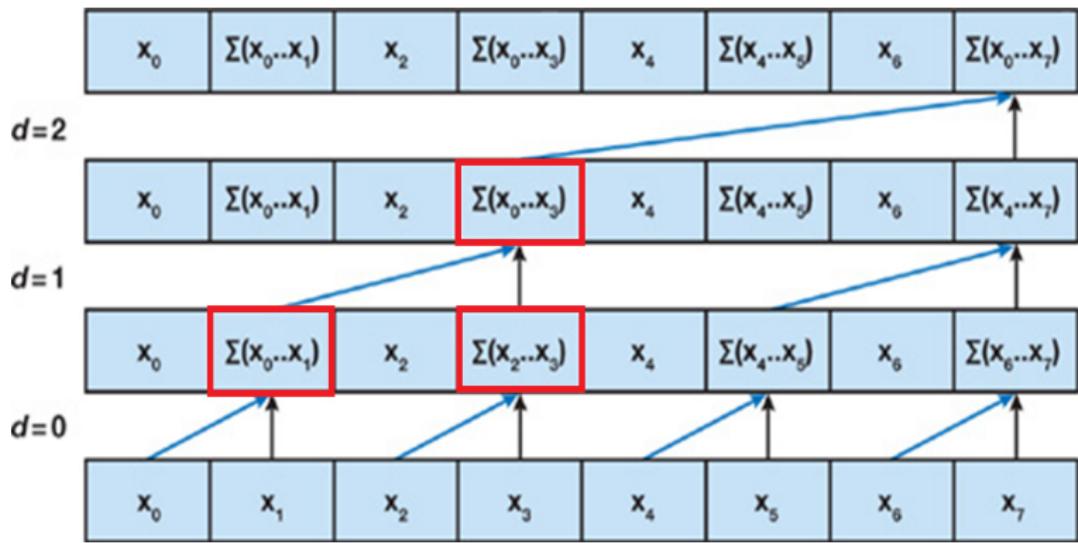


Figure 2: An illustration of the **Up-Sweep** phase

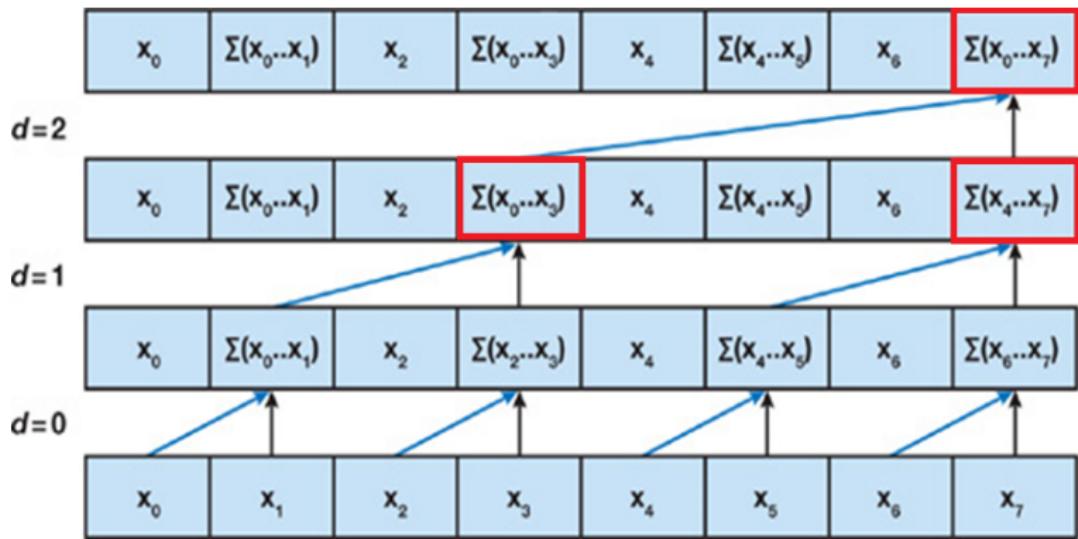
# An illustration of the Up-Sweep phase



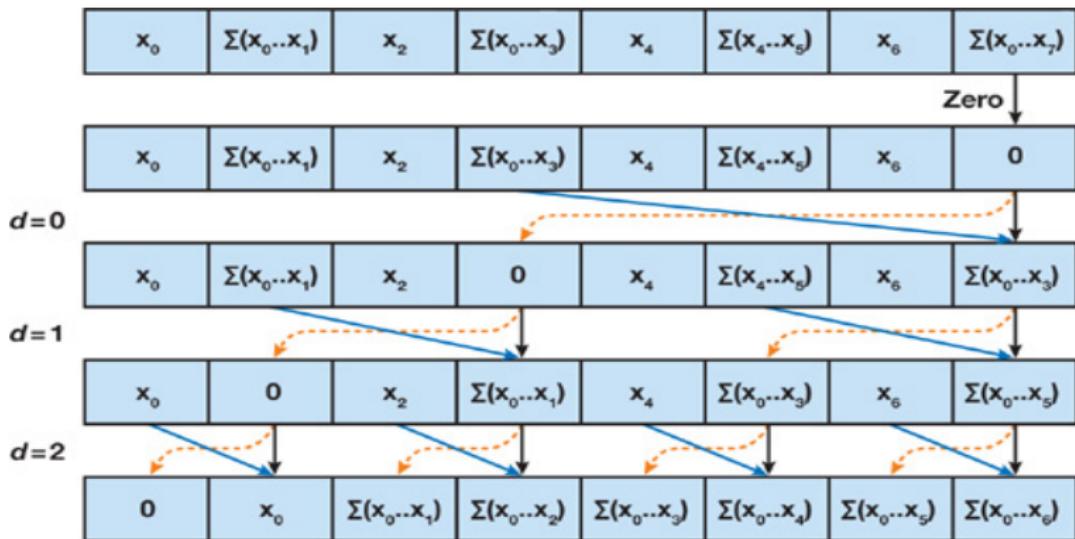
# An illustration of the Up-Sweep phase



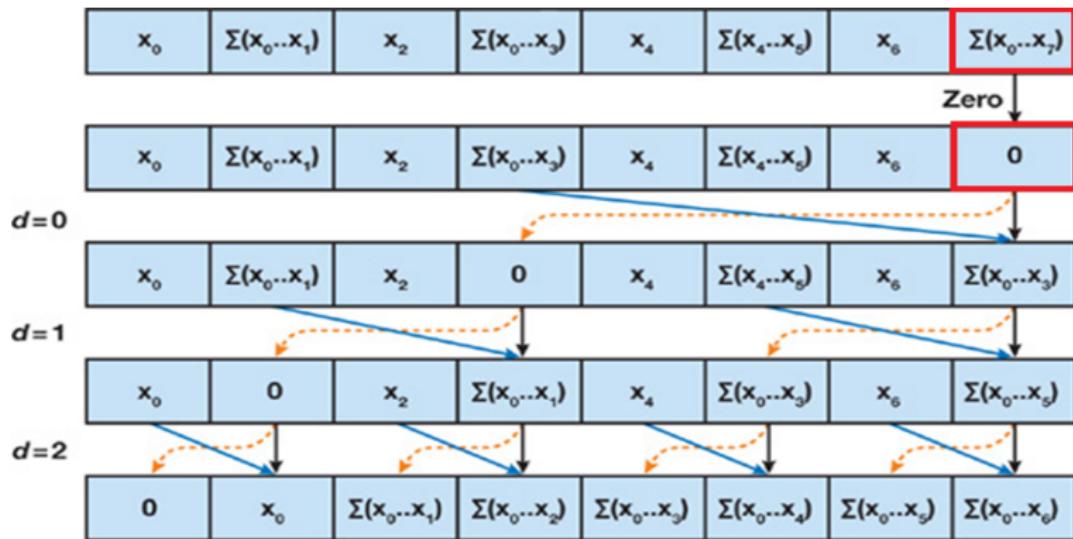
# An illustration of the Up-Sweep phase



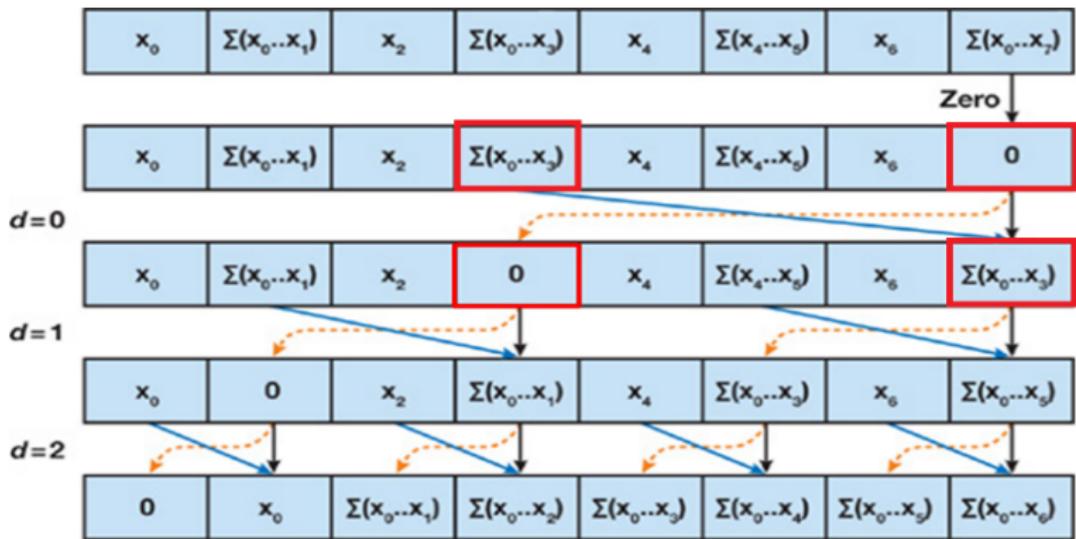
# An illustration of the Down-Sweep phase



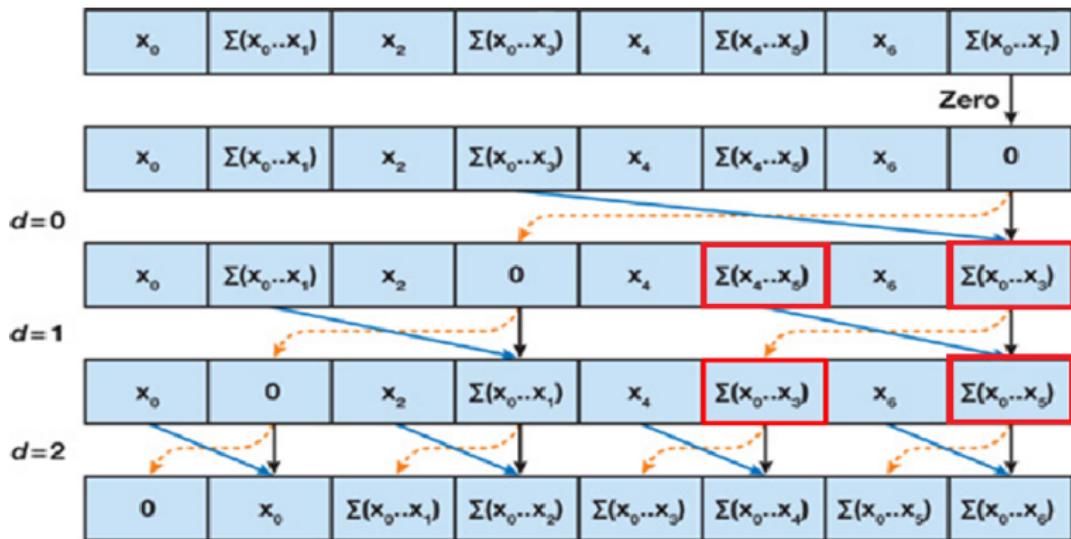
# An illustration of the Down-Sweep phase



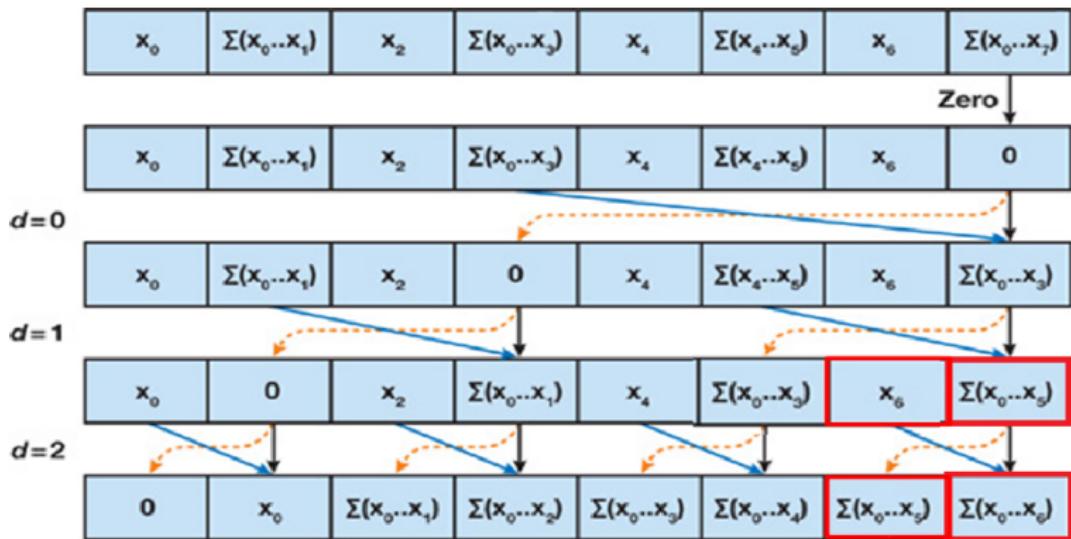
# An illustration of the Down-Sweep phase



# An illustration of the Down-Sweep phase



# An illustration of the Down-Sweep phase



# Complexity Analysis

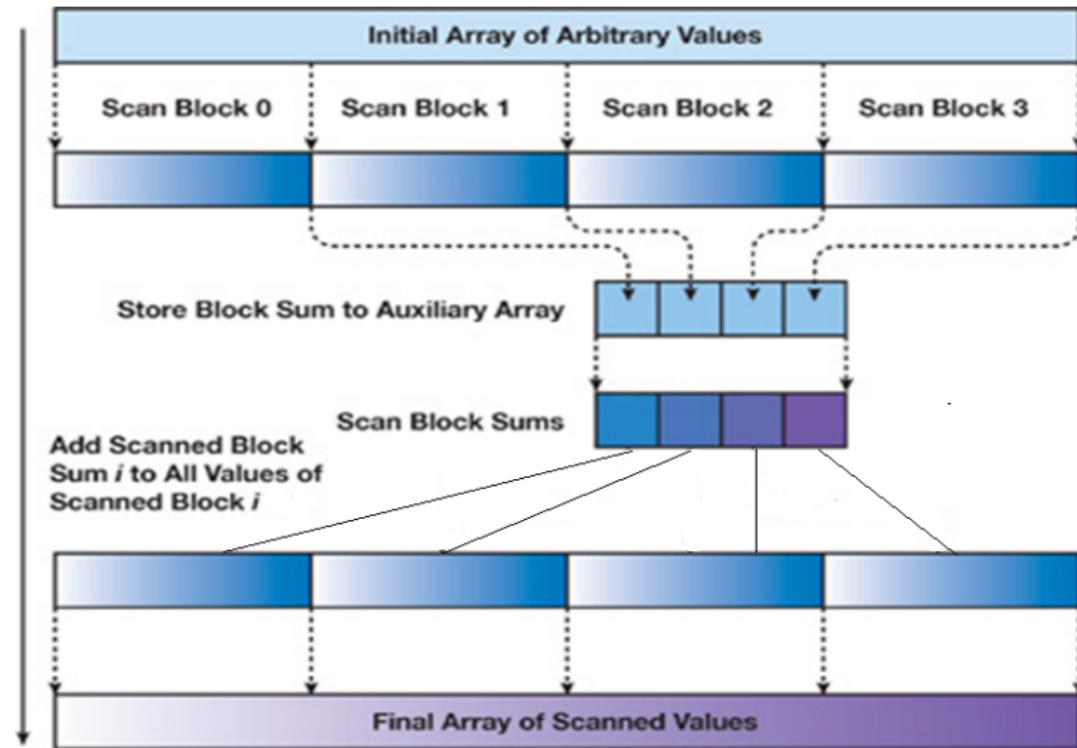
## Up-Sweep Complexity:

- A binary tree with  $n$  leaves and  $d = \log_2 n$  levels, and each level  $d$  has  $2^d$  nodes. If it is performed one add per node, it is computed  $O(n)$  adds on a single traversal of the tree.

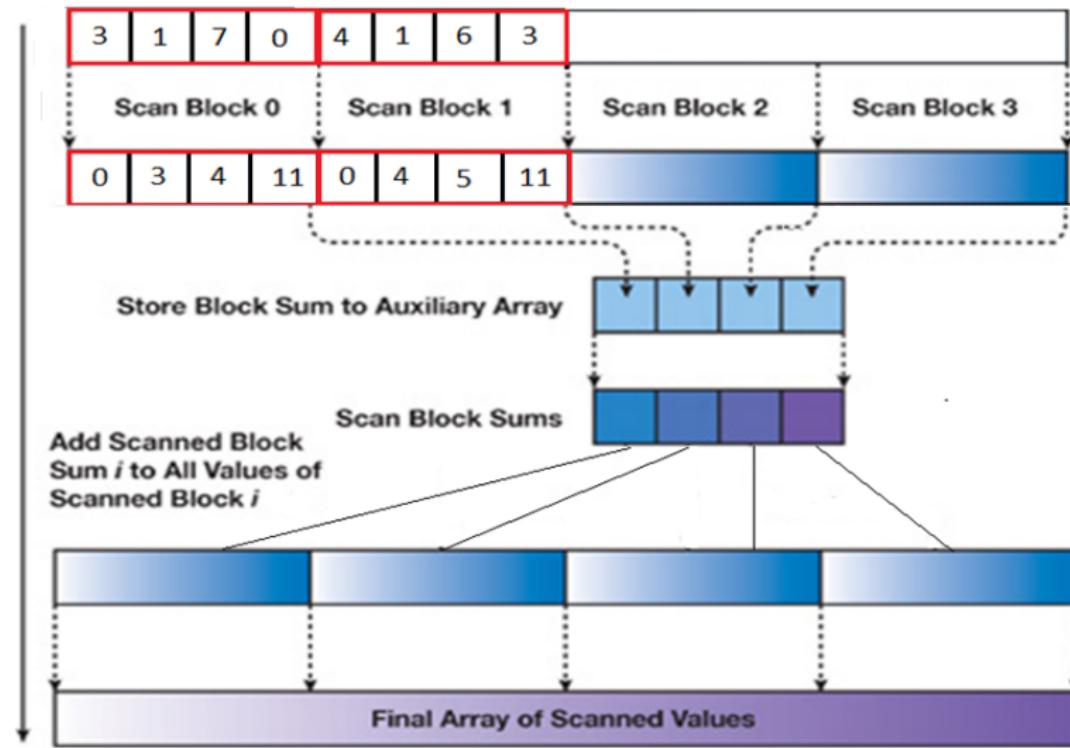
## Down-Sweep Complexity:

- Similar as Up-Sweep phase. It is performed  $O(n)$  adds and  $O(n)$  swaps.

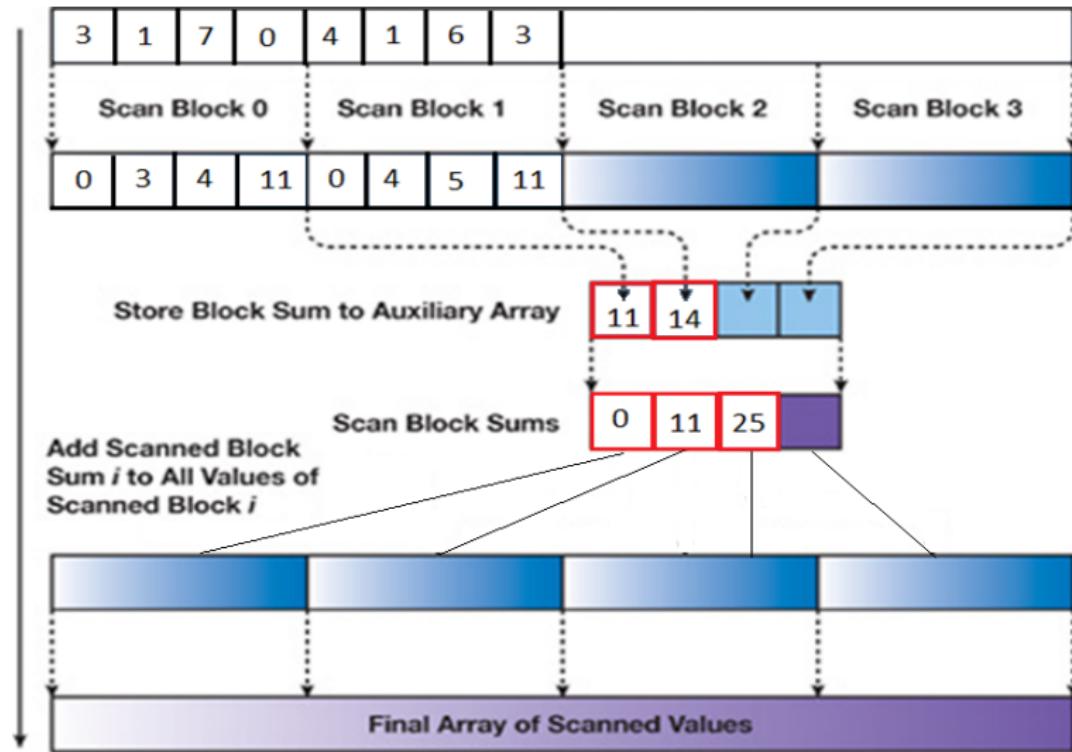
# Algorithm for large arrays



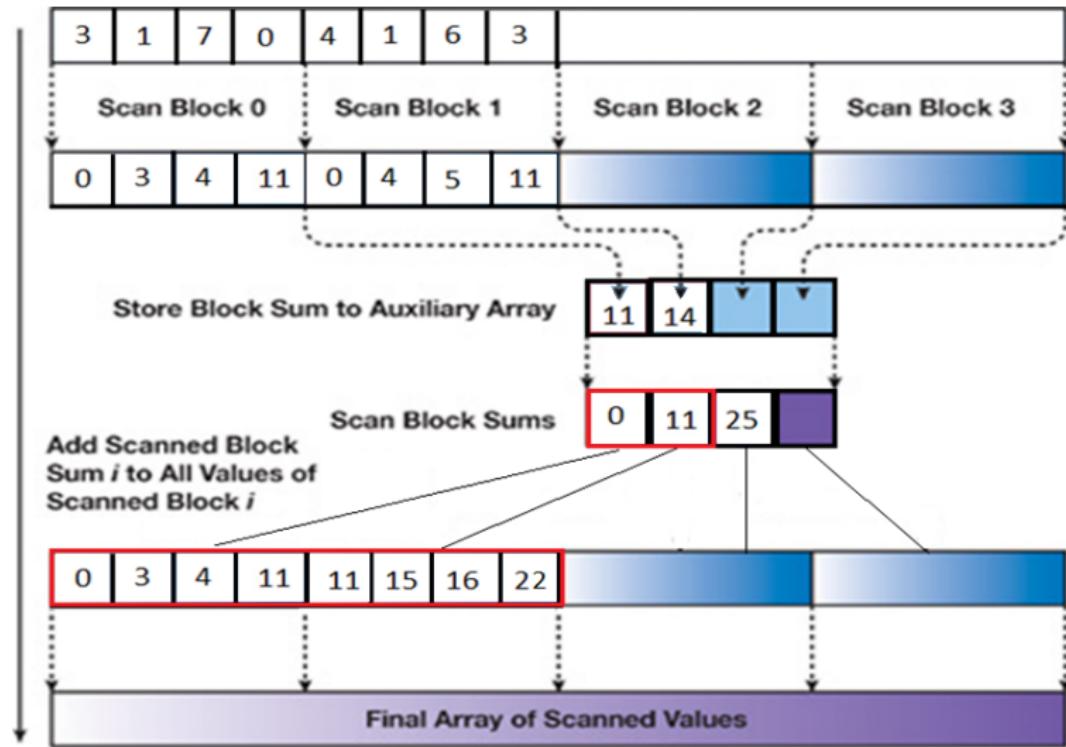
# Algorithm for large arrays



# Algorithm for large arrays



# Algorithm for large arrays



# Table of Contents

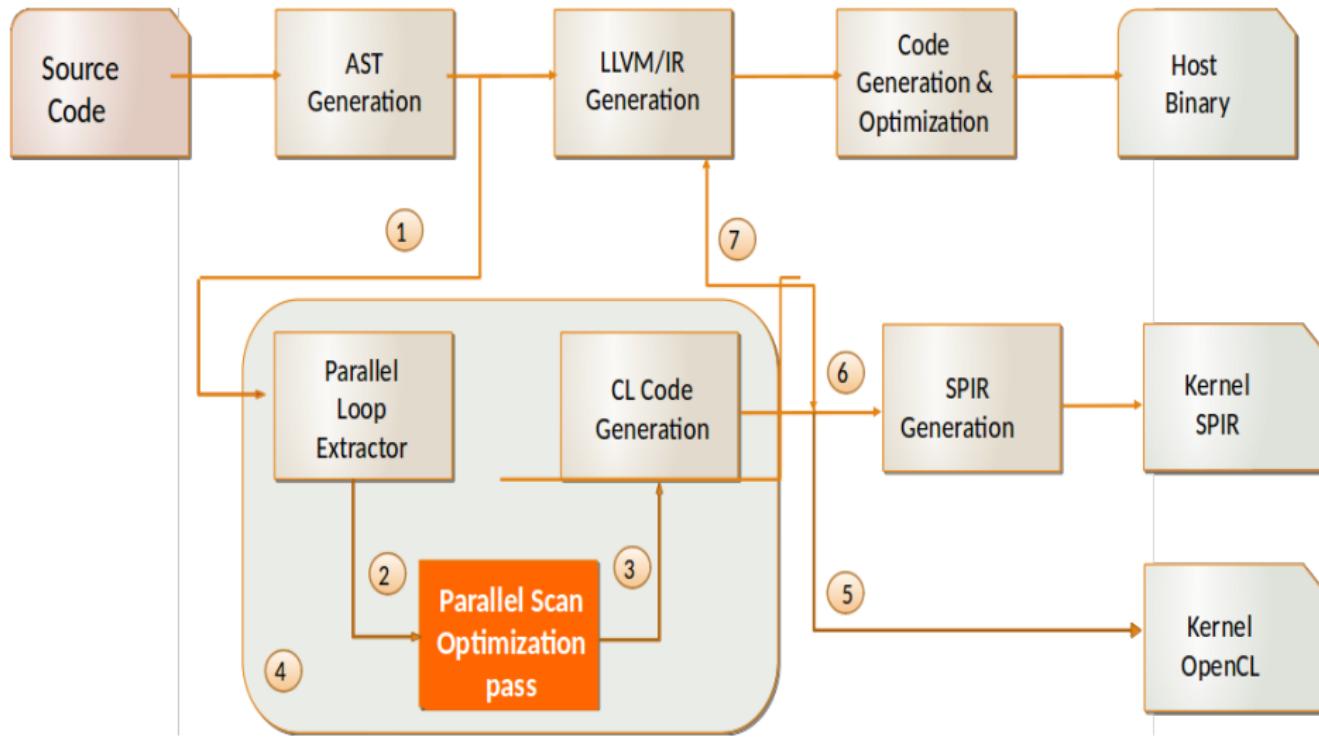
- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# What is AClang?



- AClang compiler is based in LLVM Clang 3.5;
- Designed by our group at UNICAMP;
- Supports Offloading to accelerators;
- From OMP to OpenCL or SPIR code.

# A Clang Compiler Pipeline



# A Clang Compiler

Listing 1: Sequential Implementation of Scan

```
void scan(int *x, int *y, int n) {  
    y[0] = 0;  
    for(i = 1; i < n; i++)  
        y[i] = y[i-1] + x[i-1];  
}
```

Listing 2: Parallel implementation of Scan using the new clause

```
void scan(int *x, int *y, int n) {  
    y[0] = 0;  
    #pragma omp target device (device) \  
        map(to:y[:N]) map(from:x[:N])  
    #pragma omp parallel for scan (+:y)  
    for(i = 1; i < n; i++)  
        y[i] = y[i-1] + x[i-1];  
}
```

# Table of Contents

1 Introduction

2 The Algorithm

3 Scan Clause in AClang

- Implementation of Scan in AClang
- The Template

4 Using the Scan Operator

5 Experimental Evaluation

6 Conclusion and Future Works

# Implementation of Scan in AClang

The implementation is based on the best parallel scan algorithm known today (Mark Harris).

Main steps:

- The algorithm obtains the pieces of information of the `omp scan clause` (variable type, operator, vector size);
- Compute the number of blocks and threads per block to use (Scan algorithm just work for  $2^k$  vector sizes);
- Generate code for the runtime library to call the OpenCL driver to compile the kernels and dispatch them for execution;
- Send information from `omp declare target section` (new variables types and operator overload).

# Table of Contents

1 Introduction

2 The Algorithm

3 Scan Clause in AClang

- Implementation of Scan in AClang
- The Template

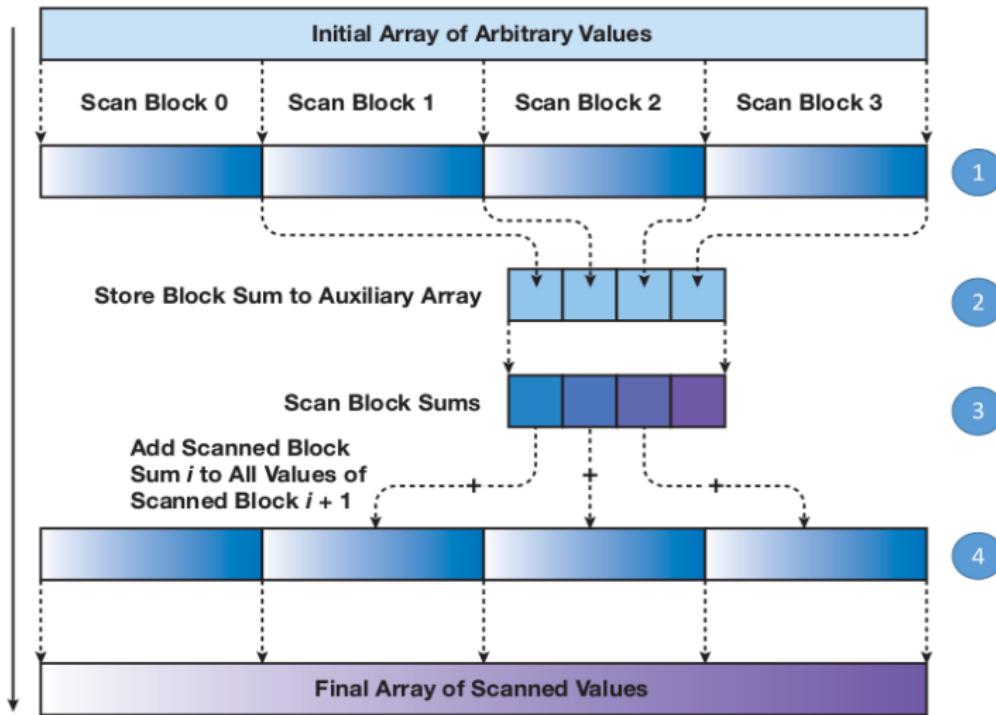
4 Using the Scan Operator

5 Experimental Evaluation

6 Conclusion and Future Works

# The Template

The template represents the algorithm explained in the following figure:



# Template's Structure

Listing 3: Template's Structure

```
//OMP_DECLARE_TARGET REGION
//END OMP_DECLARE_TARGET
#define Neutral I //Neutral value

__kernel void kernel_0 (datatype *input,\n
                      datatype *S, int n){\n
    //Performs (1) and (2)\n
}

__kernel void kernel_1(dataType *input, int n){\n
    //Performs (3)\n
}

__kernel void kernel_2(dataType *input,\n
                      dataType *output, dataType *S){\n
    //Performs (4)\n
}
```

# The Template

The algorithm has only three parameters that could be different according to the applications.

- Variable type (int, float, double and so on);
- Operator used (+, \*, &, |, *max*, *min* and so on);
- OMP declare target region (New variable type and operator).

# The Template

Ouput and input vector are the same. In this case the last kernel does not have the output vector, the result is overwritten in input.

Listing 4: Return the result into the input vector

```
int main(){
    ...
    int aux1 = input[0], aux2;
    input[0] = 0;
#pragma omp target device (GPU) map(tofrom:input[:N])
    input[0] = 0;
#pragma omp parallel for scan(+:input)
    for(i = 0 ; i < n ; i++){
        aux2 = input[i];
        input[i] = input[i-1] + aux;
        aux = aux2;
    }
    ...
}

__kernel void kernel_2(dataType *input, dataType *S)
```

# Table of Contents

- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# Example of Applications

- Random number generation
- Sequence alignment
- N-body problem
- To perform lexical analysis
- Polynomial Evaluation
- To implement Parallel Quick Sort version
- Array Filter
- Solving linear recurrences

# Example of Applications

- Random number generation
- Sequence alignment
- N-body problem
- To perform lexical analysis
- Polynomial Evaluation
- To implement Parallel Quick Sort version
- **Array Filter**
- Solving linear recurrences

## Example: Array Filter

**Input:** An array  $A[l:r]$  of integers elements, and an element  $x$  from  $A[l:r]$ .

**Output:** Rearrange the elements of  $A[l:r]$ , such that for some index  $k \in [l, r]$ , all elements in  $A[l:k]$  are smaller or equal than  $x$  and all elements in  $A[k + 1:r]$  are larger than  $x$ .

A: 

9	5	7	11	1	3	8	14	4	21
---	---	---	----	---	---	---	----	---	----

 $x = 8$

A: 

0	1	2	3	4	5	6	7	8	9
5	7	1	3	4	8	9	11	14	21

# Example: Array Filter

A: 

9	5	7	11	1	3	8	14	4	21
---	---	---	----	---	---	---	----	---	----

 $x = 8$

B: 

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

lt: 

0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	0	1	0

gt: 

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	0	1

# Example: Array Filter

A: 

9	5	7	11	1	3	8	14	4	21
---	---	---	----	---	---	---	----	---	----

 $x = 8$

B: 

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

lt: 

0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	0	1	0

lt: 

0	1	2	2	3	4	4	4	5	5
---	---	---	---	---	---	---	---	---	---

prefix sum

gt: 

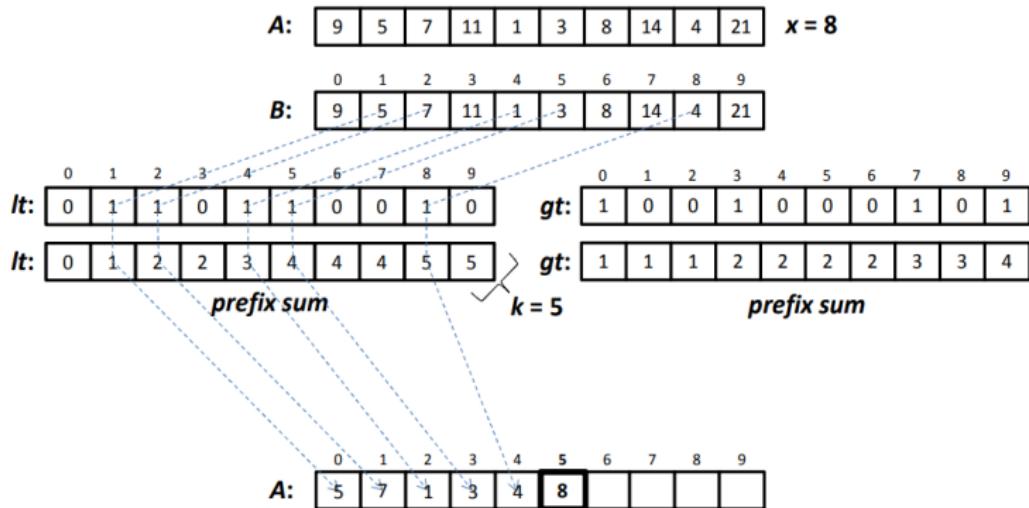
0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	0	1

gt: 

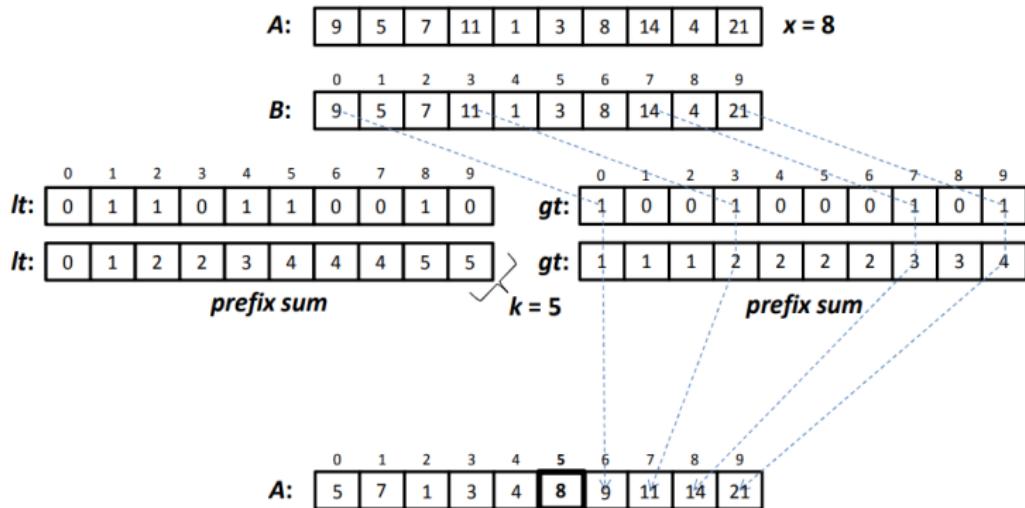
1	1	1	2	2	2	2	3	3	4
---	---	---	---	---	---	---	---	---	---

prefix sum

# Example: Array Filter



# Example: Array Filter



# Source Code: Array Filter

Listing 5: Array Filter Implementation

```
int main(){
    int *A, *lt , *gt, ltScan, gtScan, x, N;
    Reading and allocating variables

    compute(lt, x, lower_equal); //In parallel
    compute(gt, x, greater);   //In parallel

    #pragma omp target device (GPU) map(to:ltScan[:N]) map(from:lt[:N])
    ltScan[0] = 0;
    #pragma omp parallel for scan(+:ltScan)
    for(i = 1 ; i < N ; i++)
        ltScan[i] = ltScan[i-1] + lt[i-1];

    #pragma omp target device (GPU) map(to:gtScan[:N]) map(from:gt[:N])
    gtScan[0] = 0;
    #pragma omp parallel for scan(+:gtScan)
    for(i = 1 ; i < N ; i++)
        gtScan[i] = gtScan[i-1] + gt[i-1];

    Rearrange the elements from A, using lt, gt, ltScan and gtScan //In parallel
}
```

# Source Code: Array Filter

Listing 6: Array Filter Template Generated

```
#define Neutral 0

__kernel void kernel_0 (__global int *input,\n                      __global int *S, const int n)

__kernel void kernel_1 ( ... )

__kernel void kernel_2 ( ... )
```

# Example of Applications

- Random number generation
- Sequence alignment
- N-body problem
- To perform lexical analysis
- Polynomial Evaluation
- To implement Parallel Quick Sort version
- Array Filter
- Solving linear recurrences

# A realistic Example: Fibonacci Series

Let

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

And the Fibonacci numbers, defined by

$$F_0 = 0$$

$$F_1 = 1$$

$$F_{n+1} = F_n + F_{n-1}$$

Then by induction

$$A^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix}$$

And if for  $n$  the formula is true, then:

$$A^{n+1} = A \cdot A^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

$$A^{n+1} = \begin{bmatrix} F_{n+1} + F_n & F_n + F_{n-1} \\ F_{n+1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix}$$

Listing 7: Fibonacci Implementation

```
#pragma omp declare target
struct Matrix{
    long x00, x01, x10, x11;
    //Default constructor
    Matrix(){
        x00 = 1; x01 = 1;
        x10 = 1; x11 = 1;
    }
};

Matrix Operator *(Matrix A, Matrix C){
    return A*C;
}

#pragma omp declare target
```

# Source Code : Fibonacci Series

Listing 8: Fibonacci Implementation

```
int main ( ) {
    Matrix *x = new Matrix[N];
    Matrix *y = new Matrix[N];
    ...
#pragma omp target device (GPU) \
            map (to: y[:N])  map (from: x[:N])
    y[0] = Identity_Matrix;
#pragma omp parallel for scan(*:y)
    for (i = 1; i < N; i++)
        y[i] = y[i - 1] * x[i - 1];
    ...
}
```

# Source Code : Fibonacci Series

Listing 9: Fibonacci Template Generated

```
struct Matrix {
    long x00, x01, x10, x11;
};

Matrix *(Matrix A, Matrix C) {
    Mat X;
    X.x00 = A.x00 * C.x00 + A.x01 * C.x10;
    X.x01 = A.x00 * C.x01 + A.x01 * C.x11;
    X.x10 = A.x10 * C.x00 + A.x11 * C.x10;
    X.x11 = A.x10 * C.x01 + A.x11 * C.x11;
    return X;
}

#define Neutral (Matrix){ 1, 0, 0, 1 }

__kernel void kernel_0 (__global Matrix *input,
    __global Matrix *S, const int n)
__kernel void kernel_1 ( ... )
__kernel void kernel_2 ( ... )
```

# Source Code : Fibonacci Series

## Using Scan Clause

```
int main () {  
    Matrix *x = new Matrix[N];  
    Matrix *y = new Matrix[N];  
    ...  
#pragma omp target device (GPU) \  
map (tofrom: y[:N]) map (to: x[:N])  
#pragma omp parallel for scan (* : y)  
for (int i = 1; i < N; i++)  
    y[i] = y[i - 1] * x[i - 1];  
    ...  
}
```

main.c~30 lines

## OpenCL direct Programming

```
#include "main.h"  
...  
int main () {  
    cl_int err = CL_SUCCESS;  
    cl_context context = clCreateContext(0, device_id, hostCallback, hostUserCallback, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create OpenCL context!\n");  
    cl_command_queue queue = clCreateCommandQueue(context, 0, 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create command queue!\n");  
    cl_mem global_x = clCreateBuffer(context, CL_MEM_READ_WRITE, N * sizeof(float), 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create global memory for x!\n");  
    cl_mem global_y = clCreateBuffer(context, CL_MEM_READ_WRITE, N * sizeof(float), 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create global memory for y!\n");  
    cl_mem local_x = clCreateBuffer(context, CL_MEM_READ_WRITE, N * sizeof(float), 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create local memory for x!\n");  
    cl_mem local_y = clCreateBuffer(context, CL_MEM_READ_WRITE, N * sizeof(float), 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create local memory for y!\n");  
    cl_event event_x = clCreateEvent(context, 0, 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create event for x!\n");  
    cl_event event_y = clCreateEvent(context, 0, 0, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create event for y!\n");  
    cl_program program = clCreateProgramWithSource(context, 1, &kernel_code, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create OpenCL program!\n");  
    cl_int err = clBuildProgram(program, 0, NULL, NULL, NULL, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to build OpenCL program!\n");  
    cl_kernel kernel_x = clCreateKernel(program, "fibonacci_x", &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create kernel for x!\n");  
    cl_kernel kernel_y = clCreateKernel(program, "fibonacci_y", &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to create kernel for y!\n");  
    err = clEnqueueNDRangeKernel(queue, kernel_x, 1, NULL, N, 1, 1, &event_x, 0);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to enqueue kernel for x!\n");  
    err = clEnqueueNDRangeKernel(queue, kernel_y, 1, NULL, N, 1, 1, &event_y, 0);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to enqueue kernel for y!\n");  
    err = clWaitForEvents(2, &event_x, &err);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to wait for events!\n");  
    err = clReleaseEvent(event_x);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to release event for x!\n");  
    err = clReleaseEvent(event_y);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to release event for y!\n");  
    err = clReleaseProgram(program);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to release OpenCL program!\n");  
    err = clReleaseCommandQueue(queue);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to release command queue!\n");  
    err = clReleaseContext(context);  
    if (err != CL_SUCCESS)  
        printf("Error: Failed to release OpenCL context!\n");  
    ...  
}
```

main.c~92 lines

kernel.cl~140 lines

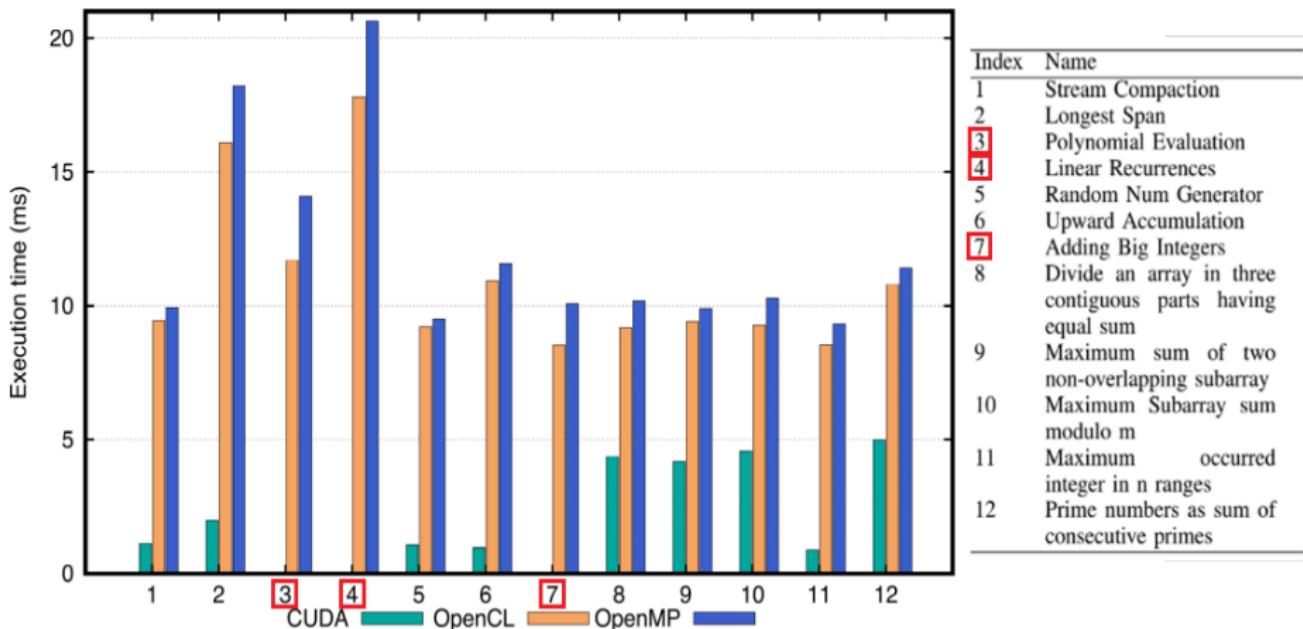
cl\_functions.c~100 lines

Total ~ 332 lines

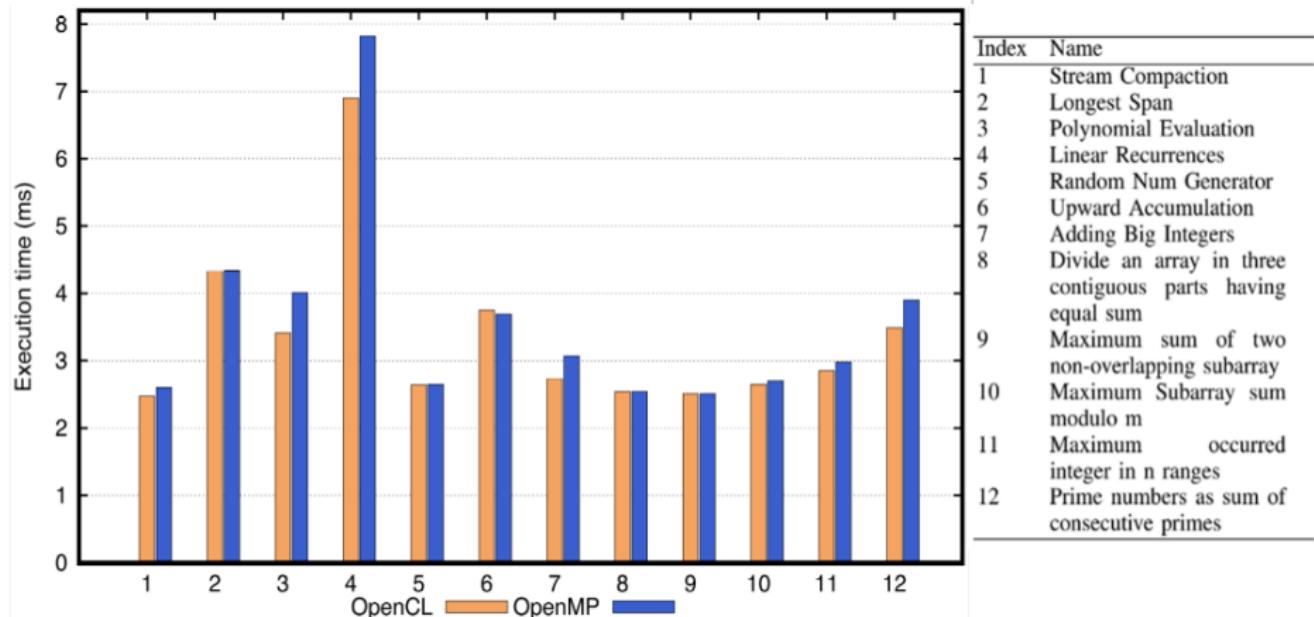
# Table of Contents

- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# a) The execution on Intel Xeon CPU E5-2620 with NVIDIA Tesla K40c



## b) The execution on Intel Core i5 with Intel Iris GPU



# Table of Contents

- 1 Introduction
- 2 The Algorithm
- 3 Scan Clause in AClang
  - Implementation of Scan in AClang
  - The Template
- 4 Using the Scan Operator
- 5 Experimental Evaluation
- 6 Conclusion and Future Works

# Conclusion and Future Works

## Conclusion:

- The scan operation is a simple and powerful parallel primitive with a broad range of applications;
- The new scan clause in OpenMP exhibits a similar performance as direct programming in OpenCL at a much smaller design effort.

# Conclusion and Future Works

## Future Works:

- Implement a template of scan algorithm for NVIDIA;
- Extend the limits of scan;
- Implement a template of reduction algorithm for AClang compiler;
- Improve the scan clause the performance by providing specific routines to handle scan operations into the AClang runtime library.

# Thanks

THANKS!

