PULP RISC-V ARA

Dr. Girish H

Professor

Department of ECE

Cambridge Institute of Technology

&

Kavinesh

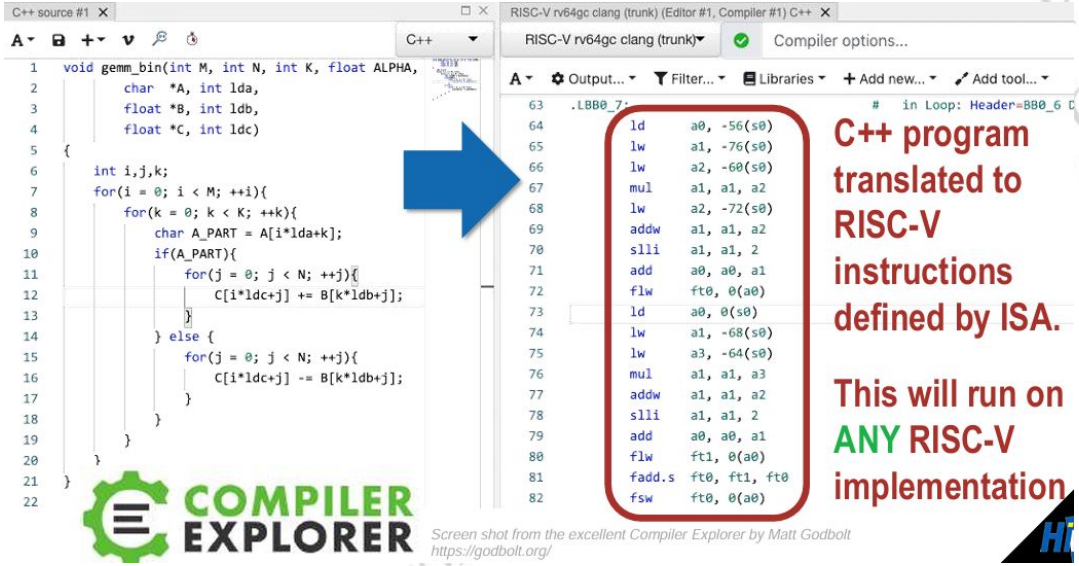
Research Staff

CCCIR

Cambridge Institute of Technology

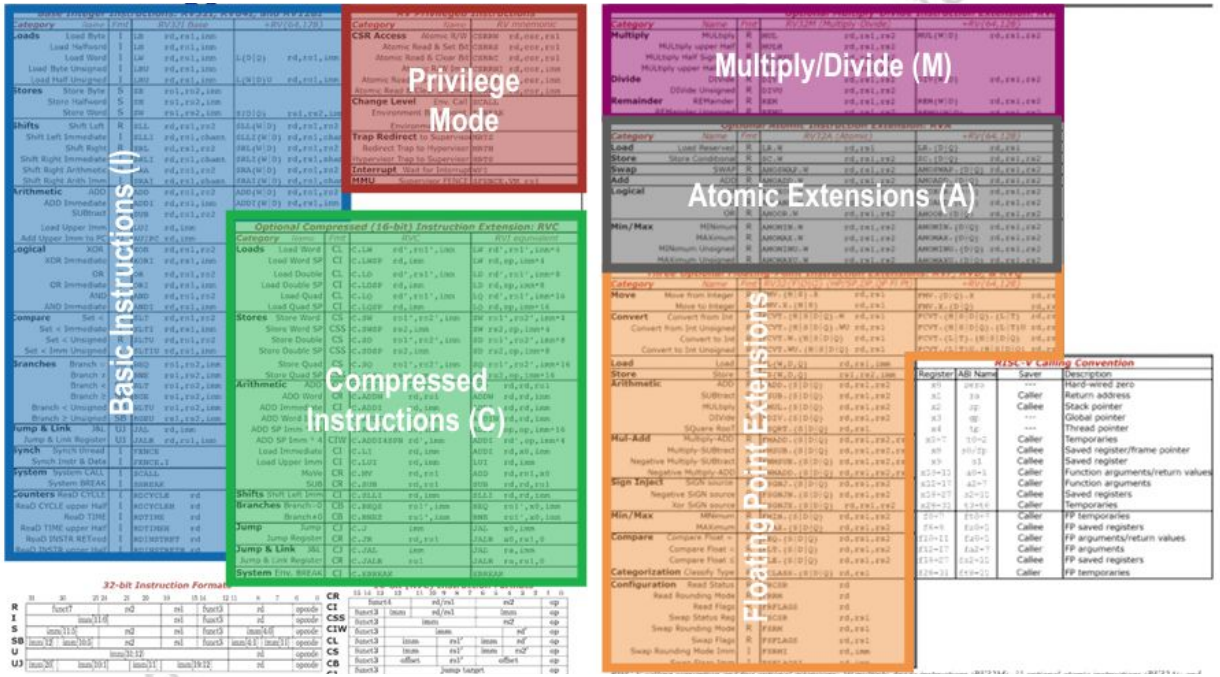
RISC-V ISA is divided into extensions

ISA defines the instructions that processor uses



I	Integer instructions (frozen)
E	Reduced number of registers
M	Multiplication and Division (frozen)
A	Atomic instructions (frozen)
F	Single-Precision Floating-Point (frozen)
D	Double-Precision Floating-Point (frozen)
C	Compressed Instructions (frozen)

Reduced Instruction Set: all in one page



PULP Parallel Ultra Low Power

- It is an open-source platform for designing energy-efficient, parallel computing systems.
- It is based on the RISC-V instruction set architecture (ISA)
- The PULP platform is designed to support efficient parallel processing.

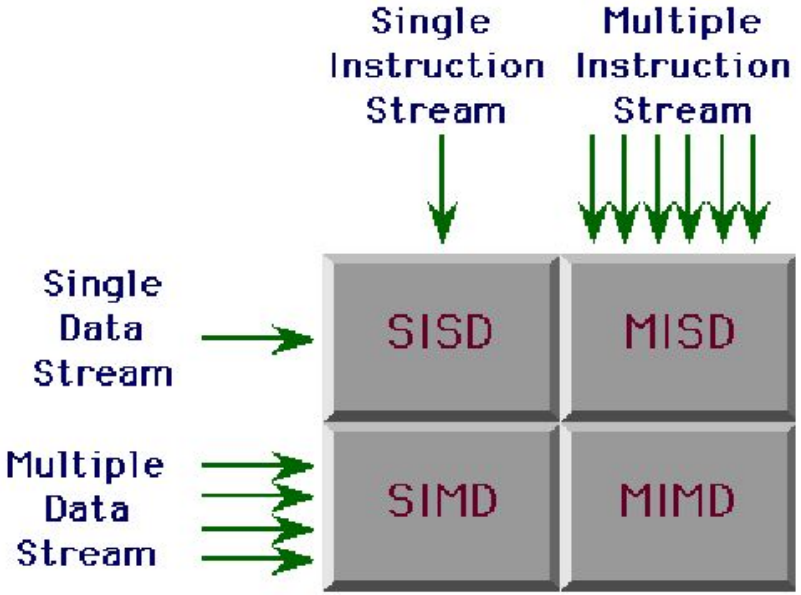
Energy

Efficient

Classification to achieve parallel computing

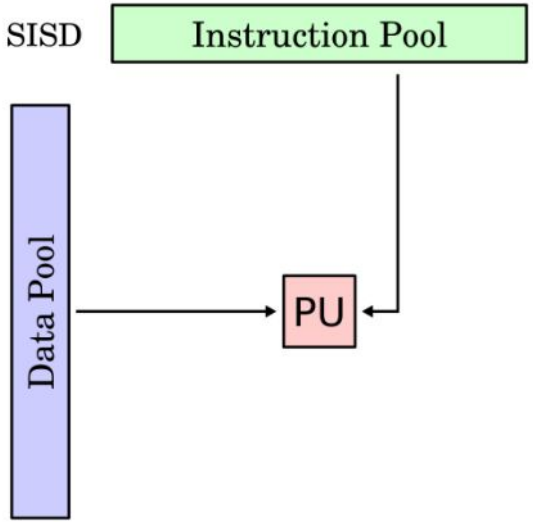
Flynn's Taxonomy

- Proposed by Michael J. Flynn in 1966
- Overview of the design space of parallel computers
- Characterization according to the data flow and the control flow



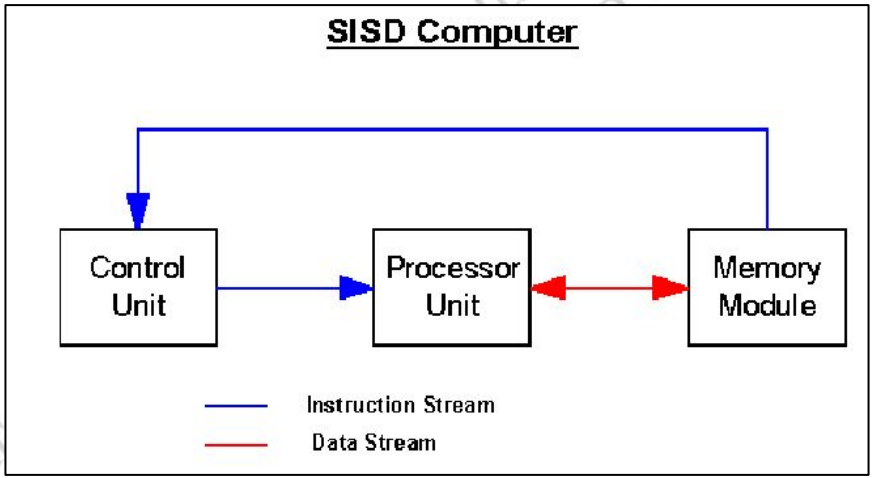
Single-Instruction, Single-Data

- One processing element
- It has access to a single program and a single data source

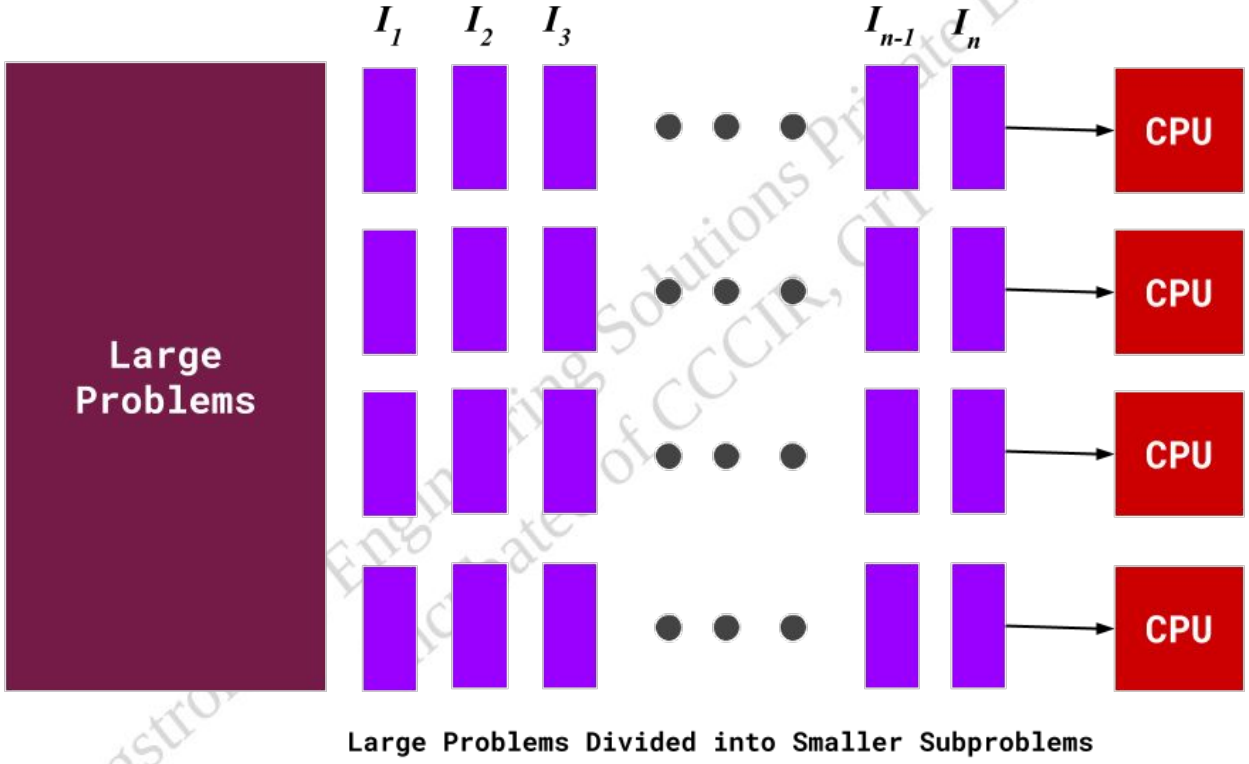


Single Instruction, Single Data Stream - SISD

- The PE fetches instructions and data from the main memory processes the data as per the instructions and sends the results to the main memory after processing has been completed

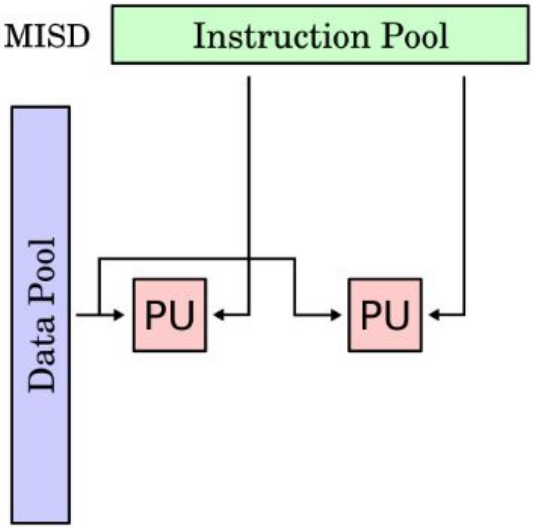


SISD is inefficient to solve for large data



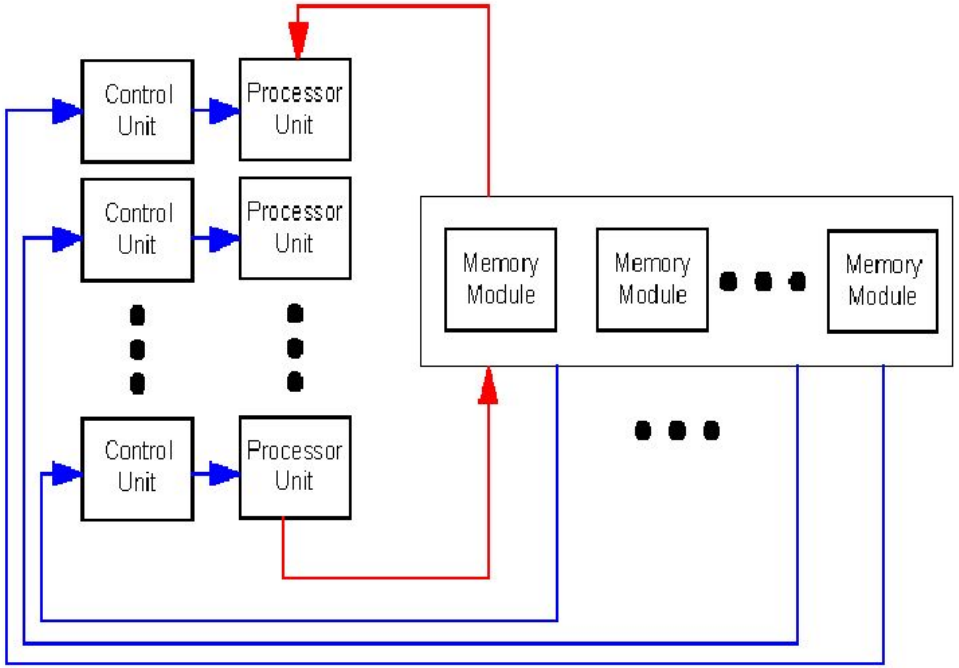
Multiple-Instructions, Single-Data

- Multiple processing elements
- Each one executes a different program on the same data



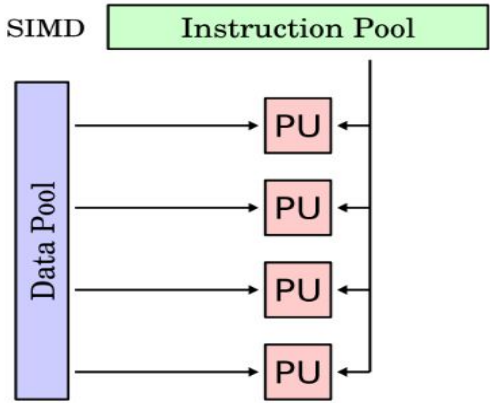
MISD computers can be useful in applications of a specialized nature:

- Robot vision
- When fault tolerance is required (military or aerospace application)
- Data can be processed by multiple machine and decision can be made on a majority principle



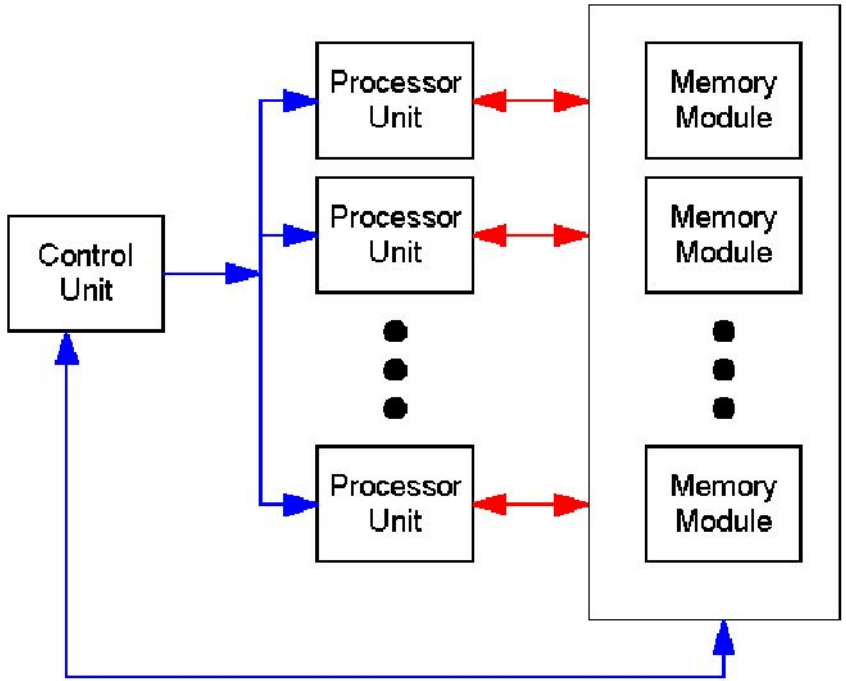
Single-Instruction, Multiple-Data

- Multiple processing elements
- Each one works on its own data ,the physical memory might be shared
- All execute the same instruction at the same time



Single Instruction, Multiple Data Stream - SIMD

- Each processing element has an associated data memory
- Each instruction is executed on a different set of data by the different processors

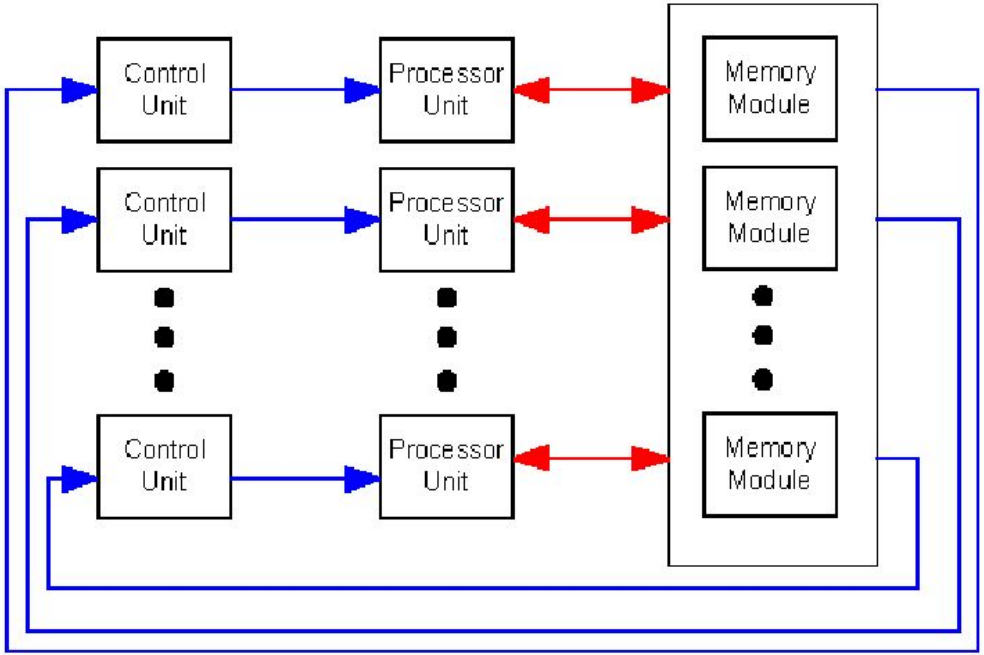


Applications:

- Image processing
- Matrix manipulations

Multiple Instruction, Multiple Data Stream- MIMD

- A set of processors simultaneously execute different instruction sequence on different data sets.
- Each processor can process all instructions necessary.

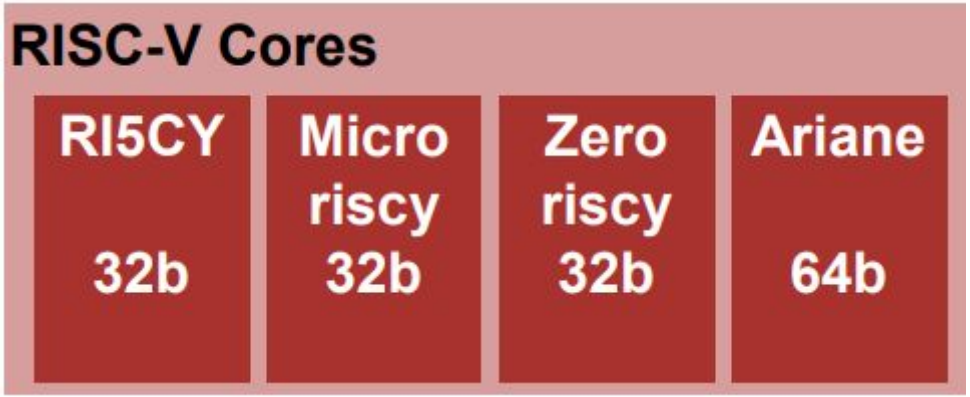


Feature	SISD	MISD	SIMD	MIMD
Instruction Stream	Single	Multiple	Single	Multiple
Data Stream	Single	Single	Multiple	Multiple
Type of Parallelism	sequential execution	Fault-tolerant	redundant	Data-level parallelism Task-level parallelism
Processing Units	Single	Multiple	Multiple	Multiple
Synchronization	Not required	High	Moderate	High
Fault Tolerance	Low	Very High	Low to Moderate	Moderate
Use Case Example	microcontrollers ; legacy CPUs	Avionics; spacecraft systems;	GPUs; DSPs; image/video processing	Multicore CPUs; Servers; distributed systems
Hardware Cost	Low	Very High	Moderate	High
Scalability	Not scalable	Poor	Good	Very Good
Performance for Parallel Tasks	Poor	Poor to Moderate	High (data-parallel workloads)	Very High (mixed/ complex workloads)

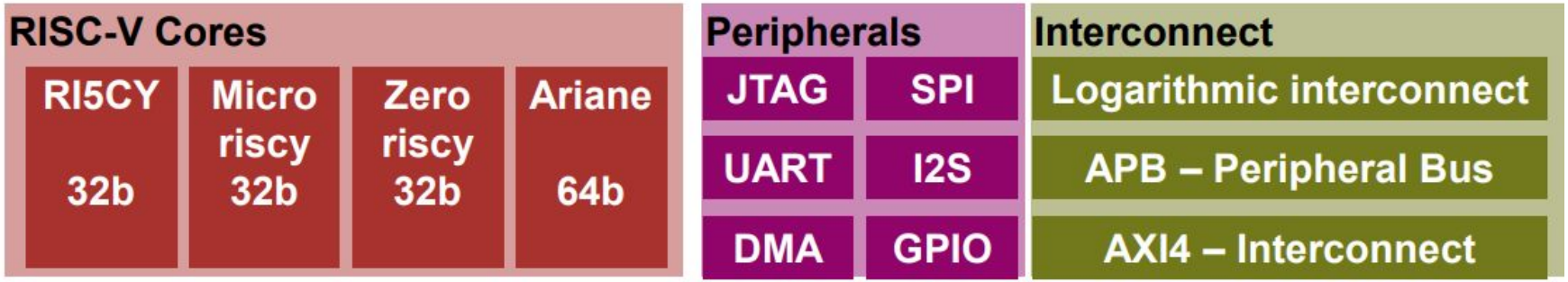
Parallel Ultra Low Power (PULP)

- Project started in 2013
- A collaboration between University of Bologna and ETH Zürich
- Large team. In total we are about 60 people, not all are working on PULP
- Key goal is
- We were able to start with a clean slate, no need to remain compatible to legacy systems.
- How to get the most BANG for the ENERGY consumed in a computing system

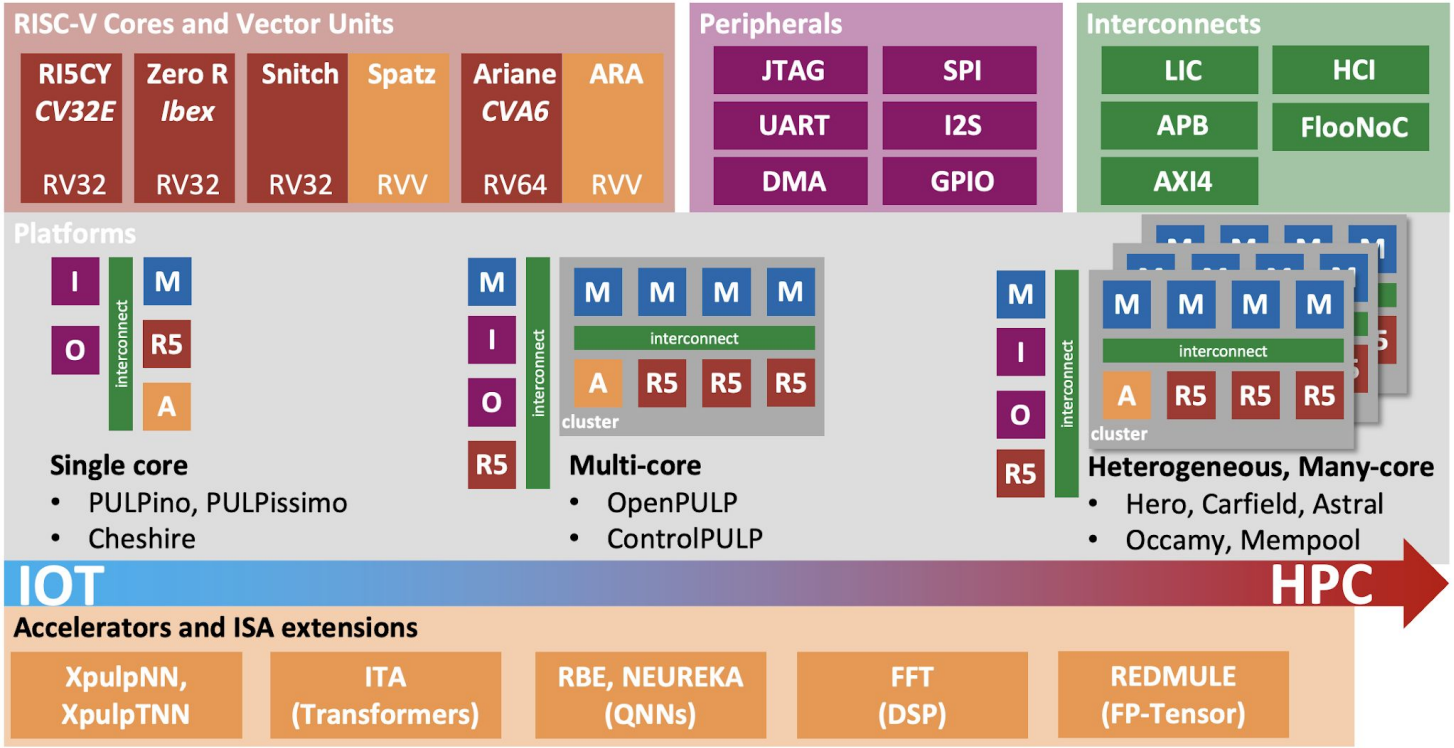
Development of several optimized RISC-V cores

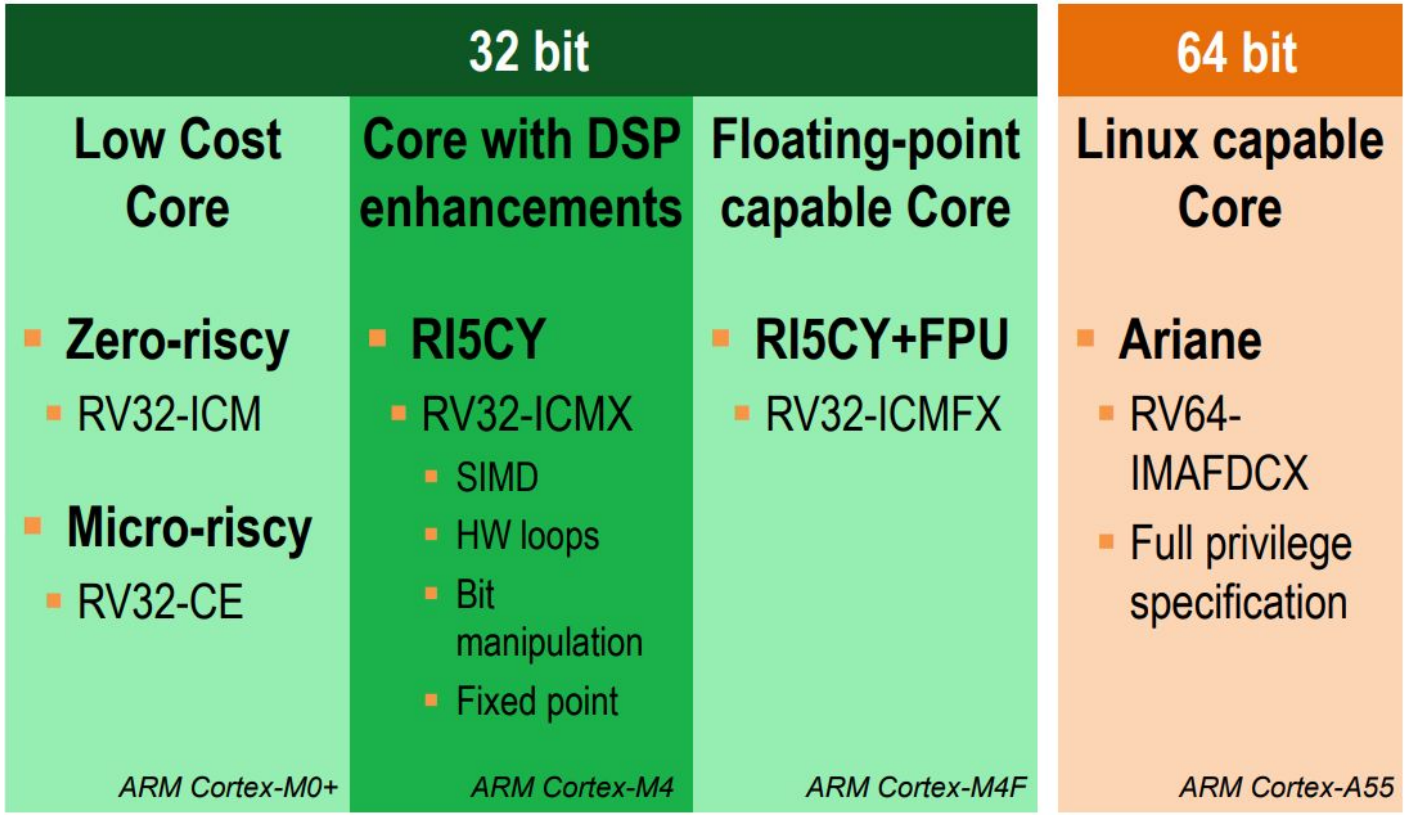


Processing cores led to requirement of Peripherals



PULP RISC-V cores





Core

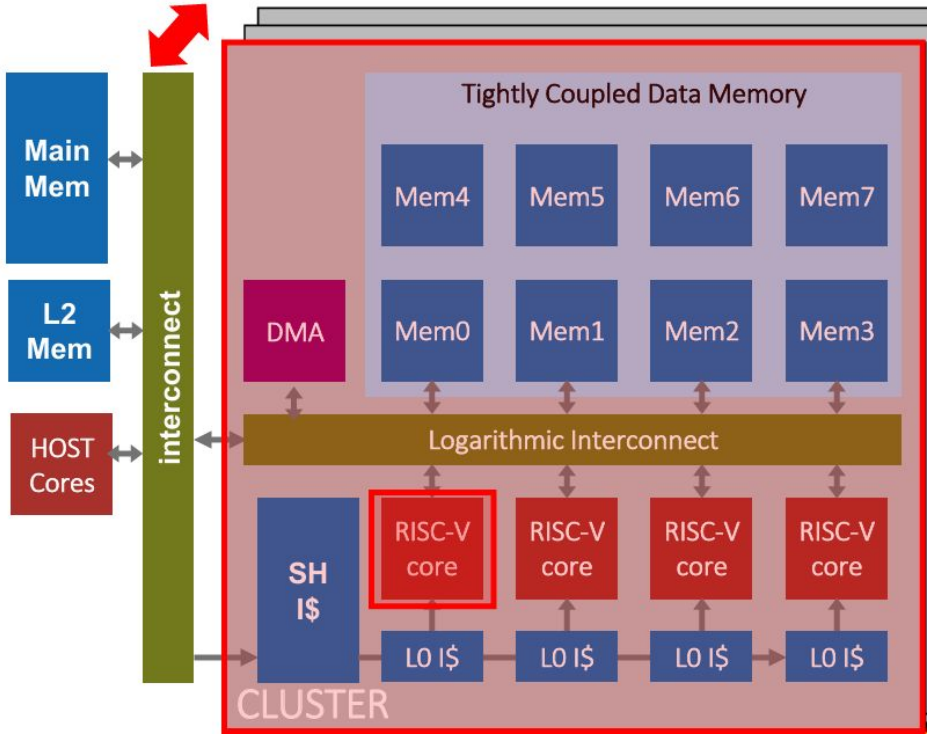
- Improving core efficiency with ISA and uAch extensions

Cluster

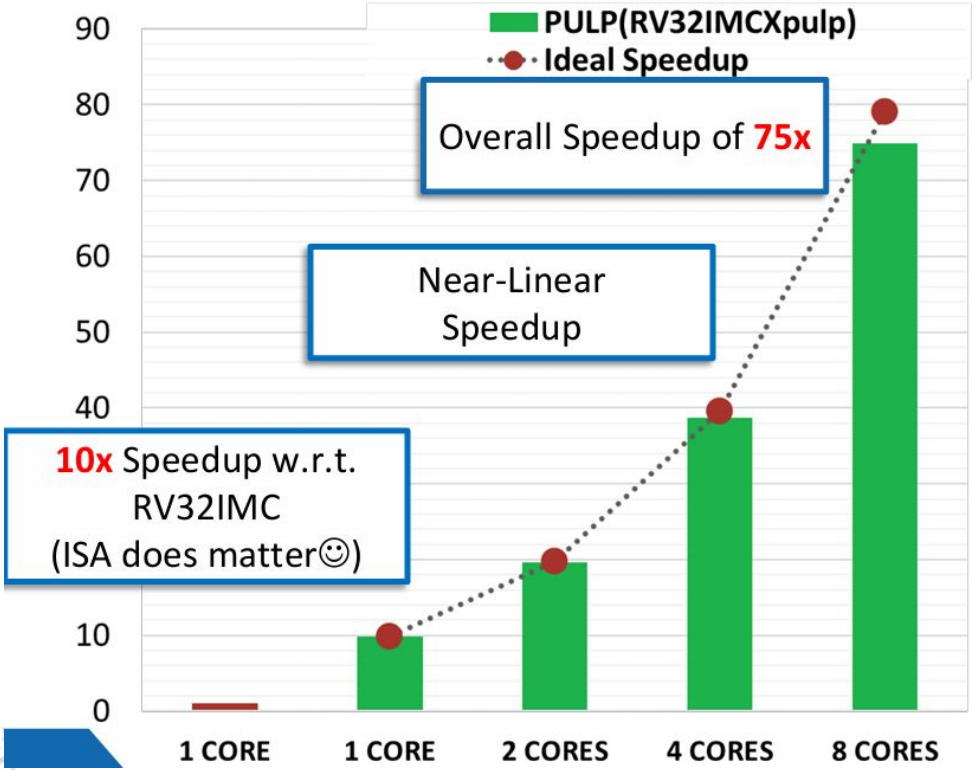
- Efficient shared-mem cluster
- From a few to thousand processing element

Full platform

- Heterogeneity: host, processor, accelerator
- One or multiple accelerators
- From chips to chiplets (2D to 3D)

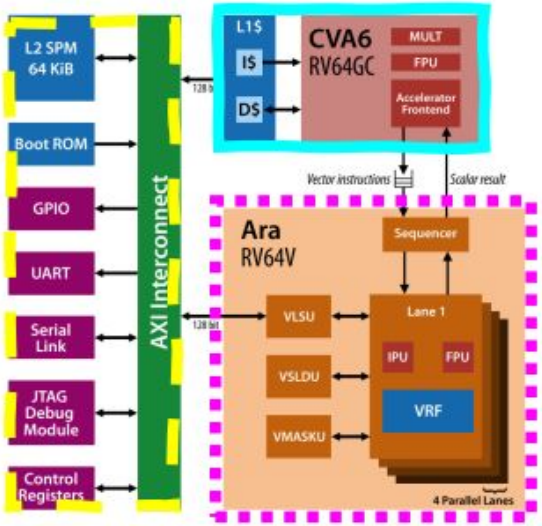


Performance

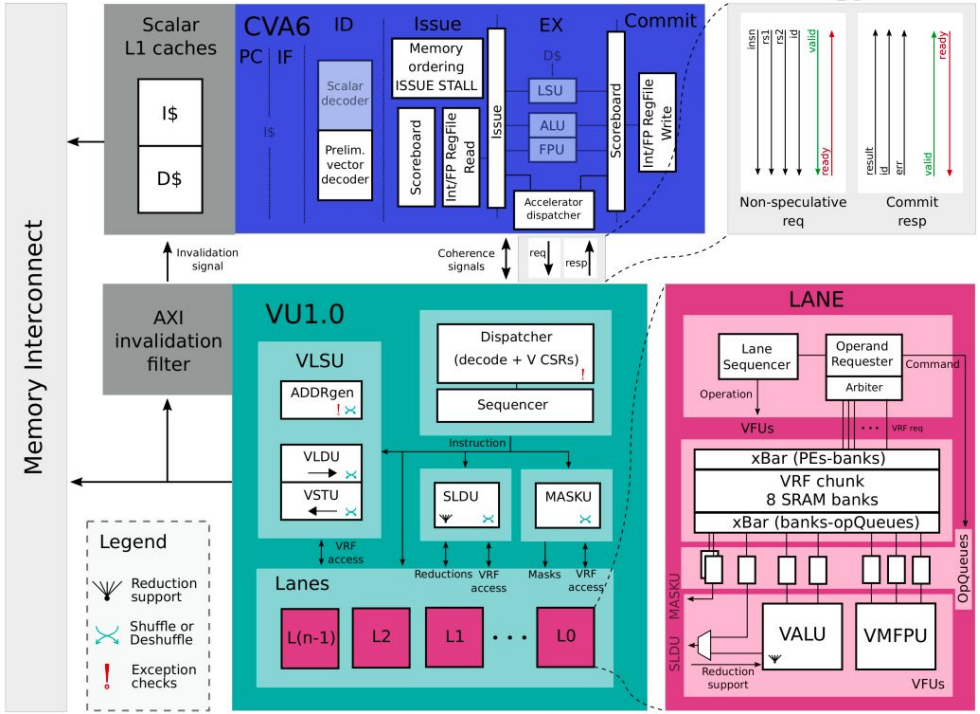


Introduction

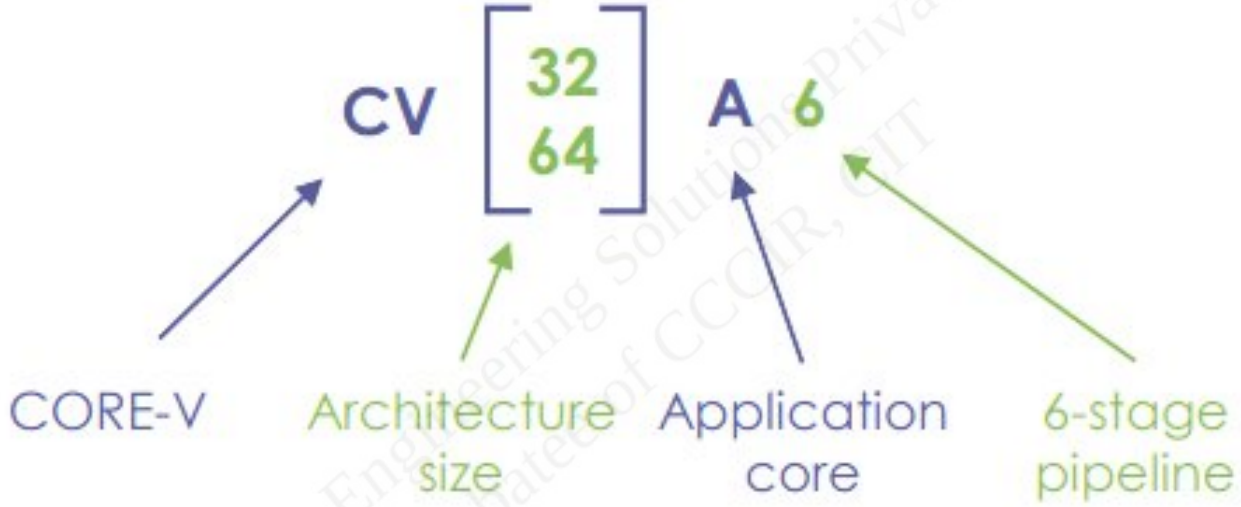
The PULP Ara is a 64-bit Vector Unit which is compatible with the RISC-V Vector extension version 1.0, working as a coprocessor to CORE-V's CVA6



PULP Ara Architecture

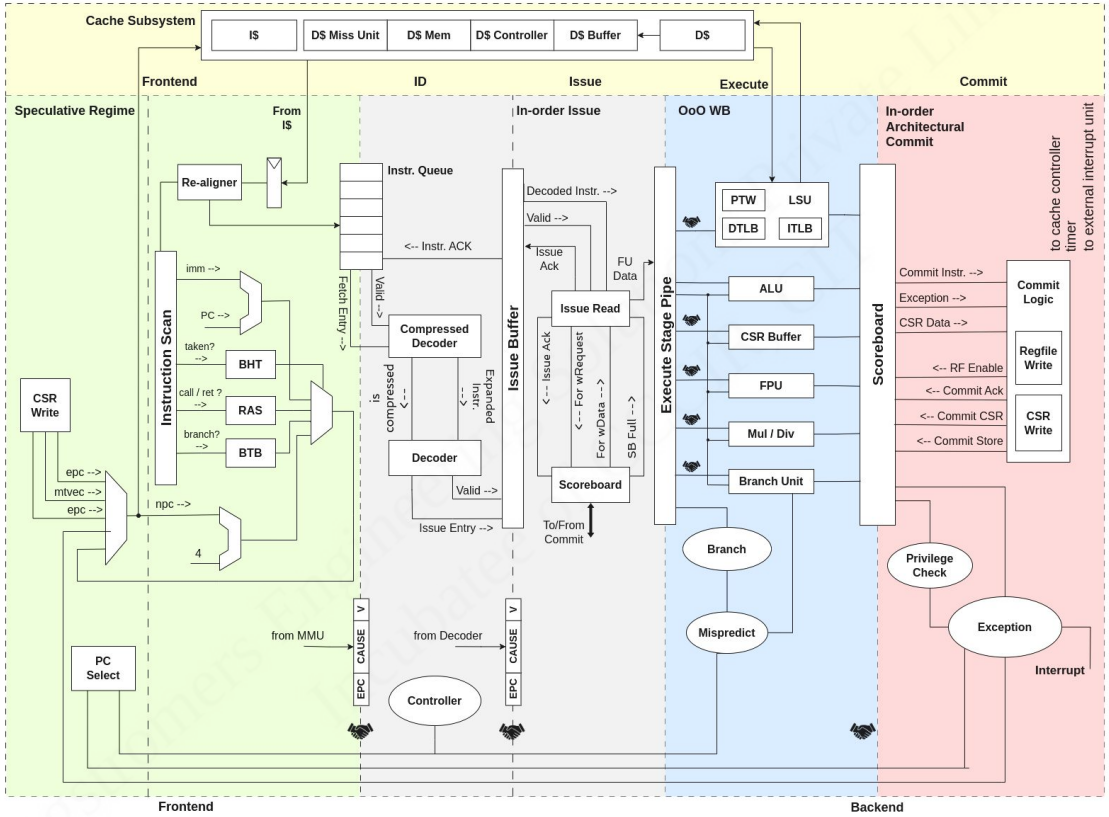


Open-source RISC-V application core

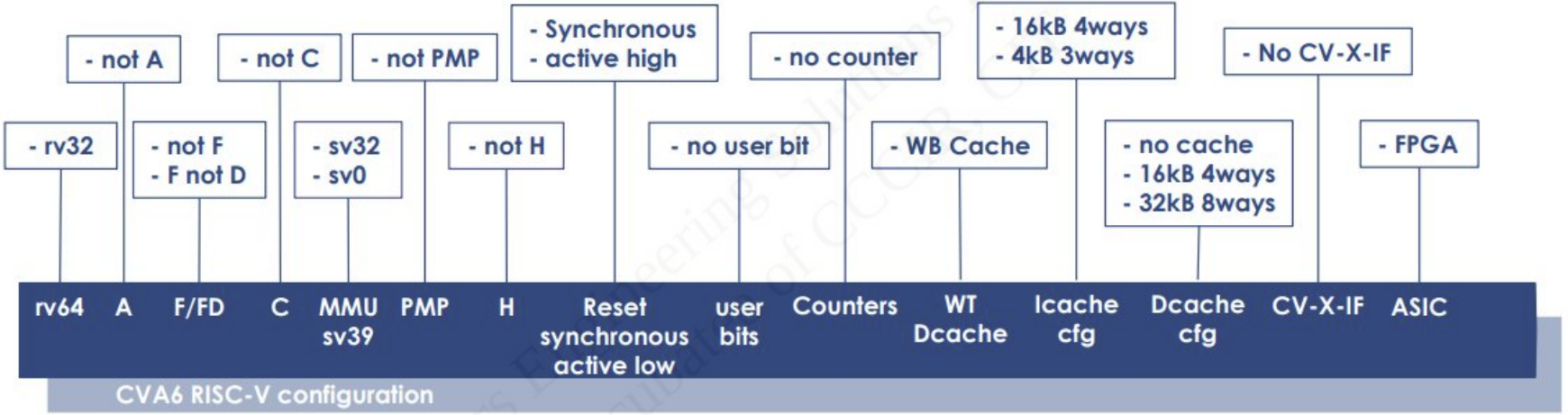


ARIANE Scalar core developed by ETH Zürich using RISC-V ISA

CVA6


Cache Subsystem
I$
D$ Miss Unit
D$ Mem
D$ Controller
D$ Buffer
D$
Frontend
ID
Issue
Execute
Commit
Speculative Regime
From I$
In-order Issue
OoO WB
In-order Architectural Commit
to cache controller
timer
to external interrupt unit
Re-aligner
Instr. Queue
Decoded Instr. -->
Valid -->
<-- Instr. ACK
Issue Ack
FU Data
PTW
LSU
DTLB
ITLB
imm -->
PC -->
taken? -->
call / ret ? -->
branch? -->
BHT
RAS
BTB
Instruction Scan
Fetch Entry -->
valid -->
Compressed Decoder
is compressed <--
Expanded instr. <--
Decoder
Valid -->
Issue Entry -->
Issue Buffer
Issue Read
<-- Issue Ack
<-- For wRequest
For wData -->
SB Full -->
Scoreboard
To/From Commit
Execute Stage Pipe
ALU
CSR Buffer
FPU
Mul / Div
Branch Unit
Branch
Mispredict
Scoreboard
Commit Instr. -->
Exception -->
CSR Data -->
Commit Logic
Regfile Write
CSR Write
<-- RF Enable
<-- Commit Ack
<-- Commit CSR
<-- Commit Store
Privilege Check
Exception
Interrupt
CSR Write
epc -->
mtvec -->
epc -->
npc -->
4
PC Select
from MMU
from Decoder
V
CAUSE
EPC
Controller
Frontend
Backend

CVA6: a highly configurable core



CVA6 core Architecture

- Fully compatible with the RISC-V open ISA
- 6-stage pipeline, single issue, branch prediction
- Soon dual-issue
- L1 data and instruction caches
- AXI4 interfaces M/S/U/H privileges
- Safe and secure features
- More upcoming (SECDED, DCLS)
- MMU and memory protection
- Ready for multi/many-core CPUs

An extendable core

- CV-X-IF interface to extend the CVA6 instruction set
- Custom extensions (cryptography, signal processing…)
- CV-X-IF specified by OpenHW Group Compiler support
- Seamless for supported standard extensions (e.g. B)
- LLVM should ease the support of custom extensions
- Inline ASM for specific processing

Software Ecosystem

Linux supported

- Available in 32 & 64 bit
- U-Boot, OpenSBI, BuildRoot Yocto
- FreeRTOS 32 & 64 bit supported
- GCC CVA6 features RISC-V extensions
- LLVM and custom extension support
- Debug: GDB, OpenOCD, Eclipse IDE

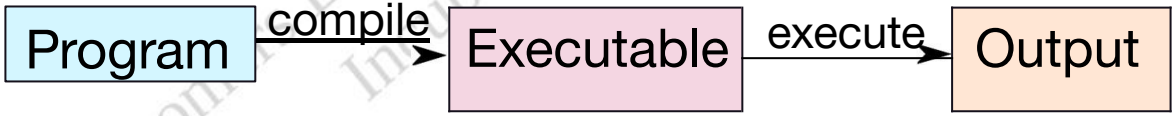
U-Boot

An academic project turning into an industrial-grade CPU core

- ARIANE donated to Open_HW Group by ETH Zürich

RISC-V Instructions

- One operation per instruction, at most one instruction per line
- Assembly instructions are related to C operations (=, +, -, *, /, &, |, etc.)
  - Must be, since C code decomposes into assembly!
  - A single line of C may break up into several lines of RISC-V
- In Assembly Language, registers have no type, simply stores 0s and 1s; operation determines how register contents are treated



Assembly Instructions

- In assembly language, each statement (called an Instruction), executes exactly one of a short list of simple commands
- Unlike in C (and most other High Level Languages), each line of assembly code contains at most 1 instruction.
- Instructions are related to operations (=, +, -, *, /) in C

RISC-V Instructions

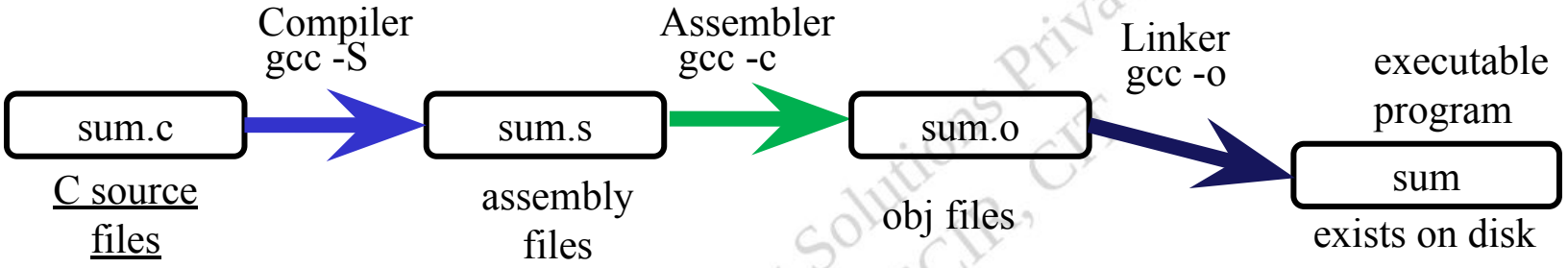
- Instruction Syntax is rigid:

  op dst, src1, src2

  – 1 operator, 3 operands
    - op = operation name (“operator”)
    - dst = register getting result (“destination”)
    - src1 = first register for operation (“source 1”)
    - src2 = second register for operation (“source 2”)

- Keep hardware simple via regularity

Software Design Flow



When most people say "compile" they mean the entire process:
compile + assemble + link

"It's alive!"

Executing
in
Memory
process

C variables vs. registers

- In C (and most High Level Languages) variables declared first and given a type.
  E.g., int fahr, celsius;
  char a, b, c, d, e;
- Each variable can ONLY represent a value of the type it was declared as (cannot mix and match int and char variables).
- In Assembly Language, the registers have no data type
  - Operation determines how register contents are treated

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
float mpg = get_mpg(c);
free(c);
```

Assembly language:

```
get_mpg(car*):
        lw      a5,0(a0)
        lw      a4,4(a0)
        divw    a5,a5,a4
        fcvt.s.w        fa0,a5
        ret
```

Machine code:

```
01110100000011000
10001101000001000000010
1000100111000010
11000001111111010000111111
```

Computer system:

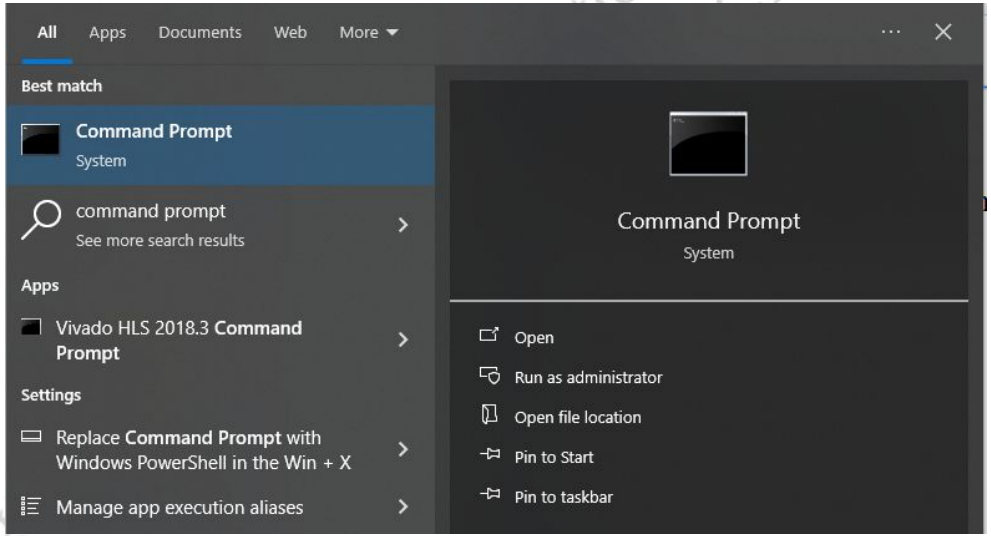
OS:

- Memory & data
- Arrays & structs
- Integers & floats
- **RISC V assembly**
- Procedures & stacks
- Executables
- Memory & caches
- Processor Pipeline
- Performance
- Parallelism

RISC-V Programming

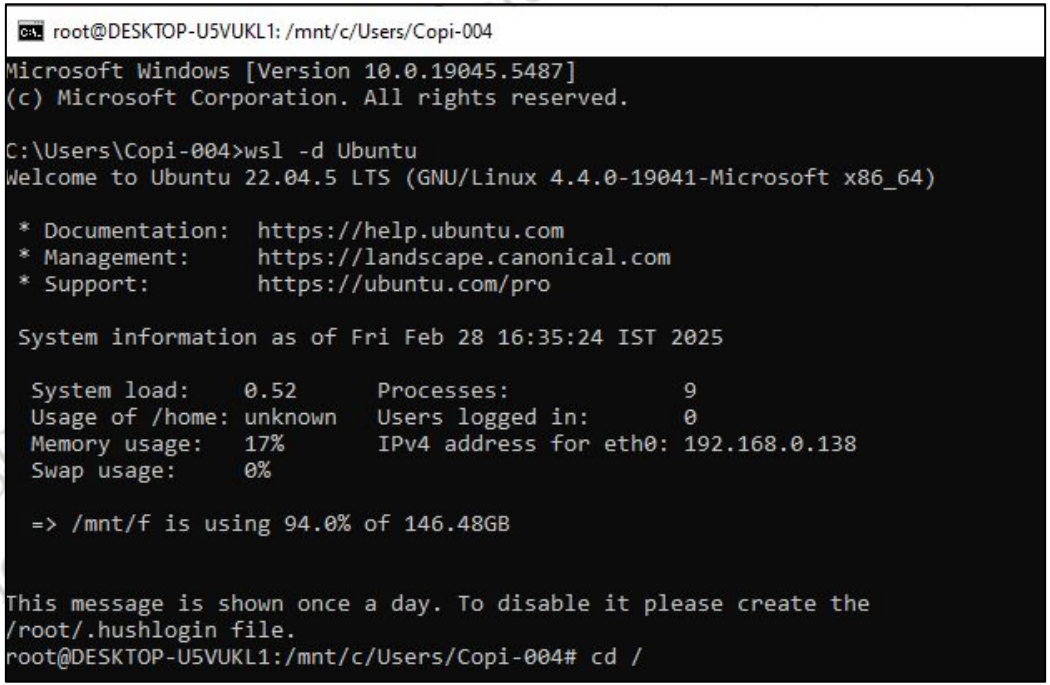
Launch the WSL software

- Search for command prompt in windows and type the following command



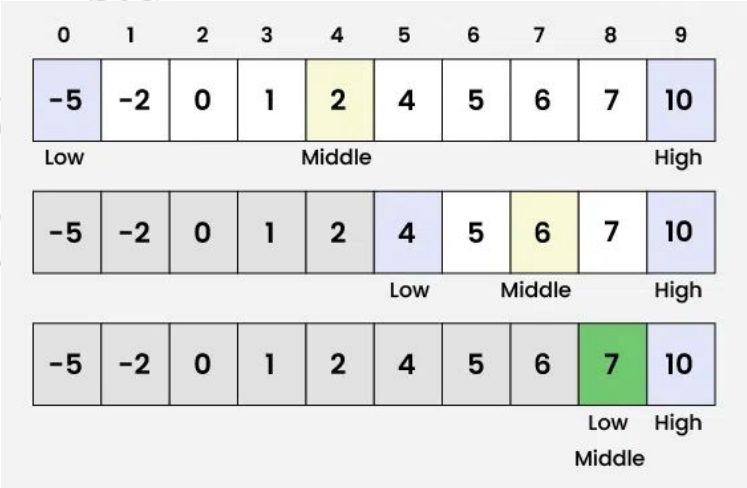
Initial Configuration

- Launch command prompt using wsl -d Ubuntu
- To change from existing windows directory cd / to the main working directory in Ubuntu



Binary Search

- Binary search is used to find a target in a sorted array.
- It works by repeatedly halving the search range.
- The middle element is compared with the target.
- If the middle element matches the target, the search stops.
- If the target is smaller, search continues in the left half.



Code Compilation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be stored.
- Example : folder_name = Program_name = binary_search

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

- To create a new file to write the code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Code Execution

```
copi001-OptiPlex-7050:~/ara$ make -C apps bin/binary_search
ering directory '/home/copi-001/ara/apps'
_search && if [ -d script ]; then python3 script/gen_data.py
i-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvf
vm -scalable-vectorization=on -mllvm -riscv-v-vector-bits-min
me/copi-001/ara/apps/common -std=gnu99 -O3 -ffast-math -fno-c
nused-command-line-argument  -ffunction-sections -fdata-secti
/home/copi-001/ara/apps/common/script/align_sections.sh
me/copi-001/ara/apps/common/link.ld && cp /home/copi-001/ara/
i-001/ara/apps/common/script/align_sections.sh 4 /home/copi-0
i-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvf
vm -scalable-vectorization=on -mllvm -riscv-v-vector-bits-min
me/copi-001/ara/apps/common -std=gnu99 -O3 -ffast-math -fno-c
nused-command-line-argument  -ffunction-sections -fdata-secti
i-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvf
vm -scalable-vectorization=on -mllvm -riscv-v-vector-bits-min
me/copi-001/ara/apps/common -std=gnu99 -O3 -ffast-math -fno-c
```

```c
#include <stdio.h>
#include <printf.h>
int binarySearch(int array[], int x, int low, int high) {
  // Repeat until the pointers low and high meet each other
  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (array[mid] == x)
      return mid;

    if (array[mid] < x)
      low = mid + 1;

    else
      high = mid - 1;
  }

  return -1;
}

int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d\n", result);
  return 0;
}
```

Code Execution

```
#include <stdio.h>
#include <printf.h>
int binarySearch(int array[], int x,
int low, int high) {
  while (low <= high) {
    int mid = low + (high - low) / 2;
    if (array[mid] == x)
      return mid;
if (array[mid] < x)
      low = mid + 1;
else
      high = mid - 1;
    }
  return -1;
}
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
   //printf("Not found");
  // else
  printf("Element is found at index %d\n", result);
  return 0;
}
```

Simulation on ARA

For simulation command : make -C hardware simv app=folder_name

Where, folder_name is the program to be executed.

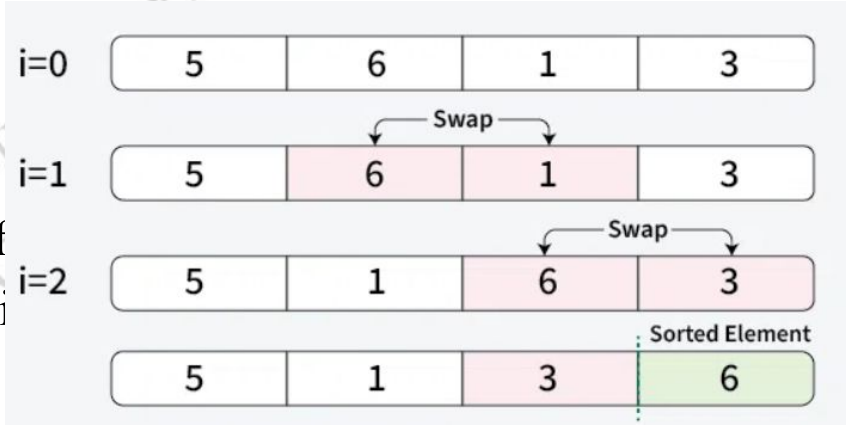
folder_name = binary_search

```
~/ara$ make -C hardware simv app=folder_name
```

```
Simulation of Ara
=================

Tracing can be toggled by sending SIGUSR1 to this process:
$ kill -USR1 29e0

Simulation running, end by pressing CTRL-c.
Element is found at index 1
[hw-cycles]:           0
[5298] -Info: ara_tb_verilator.sv:49: Assertion failed in TOP.
- /home/copi-001/ara/hardware/tb/ara_tb_verilator.sv:52: Veril
Received $finish() from Verilog, shutting down simulation.

Simulation statistics
=====================
Executed cycles:  a59
Wallclock time:   1.104 s
Simulation speed: 2399.46 cycles/s (2.39946 kHz)
```

Bubble Sort

- Bubble Sort is used, to repeatedly swaps adjacent elements if they are in the wrong order.
- It sorts in multiple passes, moving the
position in each pass.
- In each pass, only the unsorted portion of
- Elements are compared and swapped unti



Code Compilation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be stored.
- Example : folder_name = Program_name = bubble_sort

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Code Execution

```
#include <stdio.h>
#include <printf.h>
void swap(int* arr, int i, int j)
{
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}
void bubbleSort(int arr[], int n)
{
        int i, j;
        for (i = 0; i < n - 1; i++)
                for (j = 0; j < n - i - 1; j++)
                        if (arr[j] > arr[j + 1])
                                swap(arr, j, j + 1);
}
```

```
void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}
int main()
{
        int arr[] = { 5, 1, 4, 2, 8 };
        int N = sizeof(arr) / sizeof(arr[0]);
        bubbleSort(arr, N);
        printf("Sorted array: ");
        printArray(arr, N);
        return 0;
}
```

Code Execution

```
~/ara$ make -C apps bin/bubble_sort
:/copi-001/ara/apps'
)t ]; then python3 script/gen_data.py  > data.S ; else touch data.S; fi
tv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -menable-experimental-extensior
)tion=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -target-feature -Xclar
:ommon -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -DNR_LANE
)ument  -ffunction-sections -fdata-sections -c bubble_sort/data.S -o bubble_
)s/common/script/align_sections.sh
:ommon/link.ld && cp /home/copi-001/ara/apps/common/arch.link.ld /home/copi-
'script/align_sections.sh 4 /home/copi-001/ara/apps/common/link.ld
tv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -menable-experimental-extensior
)tion=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -target-feature -Xclar
:ommon -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -DNR_LANE
)ument  -ffunction-sections -fdata-sections -c bubble_sort/main.c -o bubble_
tv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -menable-experimental-extensior
)tion=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -target-feature -Xclar
:ommon -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -DNR_LANE
)ument  -ffunction-sections -fdata-sections -c common/crt0.S -o common/crt0-
tv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -menable-experimental-extensior
)tion=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -target-feature -Xclar
```

```c
#include <stdio.h>
#include <printf.h>
// Swap function
void swap(int* arr, int i, int j)
{
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
        int i, j;
        for (i = 0; i < n - 1; i++)

                // Last i elements are already
                // in place
                for (j = 0; j < n - i - 1; j++)
                        if (arr[j] > arr[j + 1])
                                swap(arr, j, j + 1);
}
```

Simulation on ARA

For simulation command : make -C hardware simv app=folder_name

Where, folder_name is the program to be executed.

folder_name = bubble_sort

```
~/ara$ make -C hardware simv app=folder_name
```

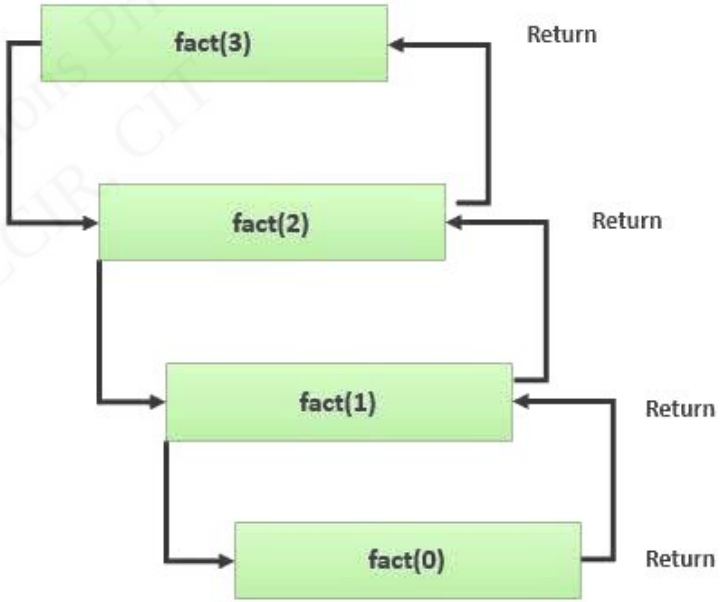
```
Simulation of Ara
=================

Tracing can be toggled by sending SIGUSR1 to this pro
$ kill -USR1 3a9a

Simulation running, end by pressing CTRL-c.
Sorted array: 1 2 4 5 8
[hw-cycles]:          0
[9230] -Info: ara_tb_verilator.sv:49: Assertion faile
- /home/copi-001/ara/hardware/tb/ara_tb_verilator.sv:
Received $finish() from Verilog, shutting down simula

Simulation statistics
=====================
Executed cycles:  1207
Wallclock time:   1.997 s
Simulation speed: 2310.97 cycles/s (2.31097 kHz)
make: Leaving directory '/home/copi-001/ara/hardware'
```

Factorial

The factorial of a number, denoted as 'n!', is the product of all positive integers up to n.

Essential in permutations, combinations, and probability.

It simplifies complex calculations, like determining possible seating arrangements



Code Compilation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store. Example : folder_name = Program_name = factorial

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Code Execution

```
#include<stdio.h>
#include <printf.h>

unsigned int factorial(unsigned int N)
{
int fact = 1,i;

for(i =1; i <= N; i++)
{
fact *= i;
}
return fact;
}
```

```
int main()
{
int N = 5;
int fact = factorial(N);
printf("Factorial of %d is %d", N, fact);
return 0;
}
```

Code Execution

```
1@copi001-OptiPlex-7050:~$ cd ara
1@copi001-OptiPlex-7050:~/ara$ make -C apps bin/factorial
ntering directory '/home/copi-001/ara/apps'
orial && if [ -d script ]; then python3 script/gen_data.py  > data.
n data.S; fi
opi-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1
erimental-extensions -mabi=lp64d -mno-relax -fuse-ld=lld -fvectoriz
lable-vectorization=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang
ure -Xclang +no-optimized-zero-stride-load -mcmodel=medany -I/home/
apps/common -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-pri
ES=4 -DVLEN=4096 -Wunused-variable -Wall -Wextra -Wno-unused-comman
nt  -ffunction-sections -fdata-sections -c factorial/data.S -o fact
o
x /home/copi-001/ara/apps/common/script/align_sections.sh
home/copi-001/ara/apps/common/link.ld && cp /home/copi-001/ara/apps
link.ld /home/copi-001/ara/apps/common/link.ld
opi-001/ara/apps/common/script/align_sections.sh 4 /home/copi-001/a
n/link.ld
opi-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1
erimental-extensions -mabi=lp64d -mno-relax -fuse-ld=lld -fvectoriz
lable-vectorization=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang
```

```c
#include<stdio.h>
#include <printf.h>

unsigned int factorial(unsigned int N)
{
int fact = 1,i;

for(i =1; i <= N; i++)
{
fact *= i;
}
return fact;
}

int main()
{
int N = 5;
int fact = factorial(N);
printf("Factorial of %d is %d", N, fact);
return 0;
}
```

Simulation on ARA

For simulation command : make -C hardware simv app=folder_name

Where, folder_name is the program.

folder_name = factorial

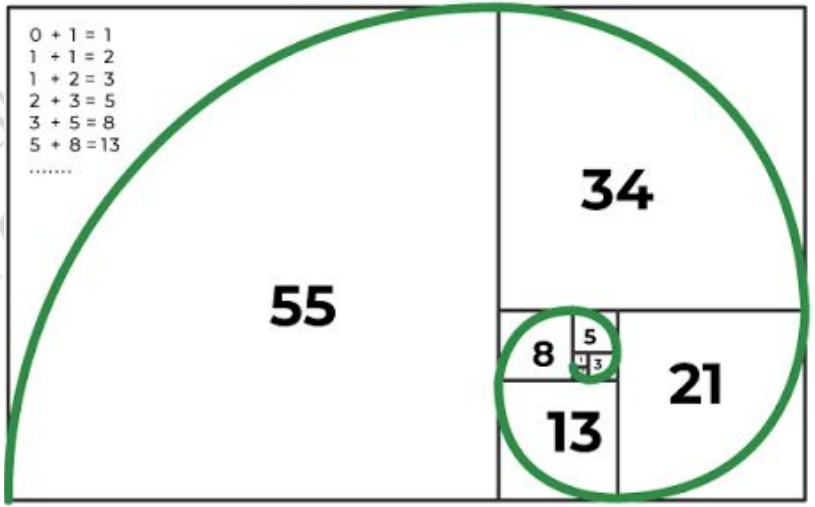
```
ara$ make -C hardware simv app=factorial
```

```
Simulation of Ara
=================

Tracing can be toggled by sending SIGUSR1 to this pro
$ kill -USR1 2208

Simulation running, end by pressing CTRL-c.
Factorial of 5 is 120[hw-cycles]:           0
[5616] -Info: ara_tb_verilator.sv:49: Assertion faile
Core Test *** SUCCESS *** (tohost = 0)
- /home/copi-001/ara/hardware/tb/ara_tb_verilator.sv:
Received $finish() from Verilog, shutting down simula

Simulation statistics
=====================
Executed cycles:  af8
Wallclock time:   1.16 s
Simulation speed: 2420.69 cycles/s (2.42069 kHz)
make: Leaving directory '/home/copi-001/ara/hardware'
```

Fibonacci Series

The Fibonacci series is the sequence where each number is the sum of the previous two numbers of the sequence.

The first two numbers are 0 and 1 which are used to generate the whole series.



Code Compilation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store. Example : folder_name = Program_name = fibonacci

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Code Execution

```
#include <printf.h>
#include <stdio.h>
int main() {
  int i, n;
  // initialize first and second terms
  int t1 = 0, t2 = 1;
  // initialize the next term (3rd term)
  int nextTerm = t1 + t2;
  // print the first two terms t1 and t2
  printf("Fibonacci Series: %d, %d, ", t1, t2);
  // print 3rd to nth terms
  for (i = 3; i <= n; ++i) {
     printf("%d, ", nextTerm);
     t1 = t2;
     t2 = nextTerm;
     nextTerm = t1 + t2;
  }

   return 0;
}
```

Code Execution

```
: Leaving directory '/home/copi-001/ara/hardware'
-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/fibonacci
: Entering directory '/home/copi-001/ara/apps'
ibonacci && if [ -d script ]; then python3 script/gen_data.py  > data.S ; el
ouch data.S; fi
e/copi-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -mena
experimental-extensions -mabi=lp64d -mno-relax -fuse-ld=lld -fvectorize -mll
scalable-vectorization=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -targ
eature -Xclang +no-optimized-zero-stride-load -mcmodel=medany -I/home/copi-0
ra/apps/common -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -
LANES=4 -DVLEN=4096 -Wunused-variable -Wall -Wextra -Wno-unused-command-line
ument  -ffunction-sections -fdata-sections -c fibonacci/data.S -o fibonacci/
.S.o
d +x /home/copi-001/ara/apps/common/script/align_sections.sh
f /home/copi-001/ara/apps/common/link.ld && cp /home/copi-001/ara/apps/commo
```

```c
#include <printf.h>
#include <stdio.h>
int main() {

  int i, n;


  // initialize first and second terms
  int t1 = 0, t2 = 1;

  // initialize the next term (3rd term)
  int nextTerm = t1 + t2;

  // print the first two terms t1 and t2
  printf("Fibonacci Series: %d, %d, ", t1, t2);

  // print 3rd to nth terms
  for (i = 3; i <= n; ++i) {
    printf("%d, ", nextTerm);
    t1 = t2;
    t2 = nextTerm;
    nextTerm = t1 + t2;
  }

  return 0;
}
```

Simulation on ARA

For simulation command : make -C hardware simv app=folder_name

Where, folder_name is the program.

folder_name = fibonacci

```
ara$ make -C hardware simv app=factorial
```

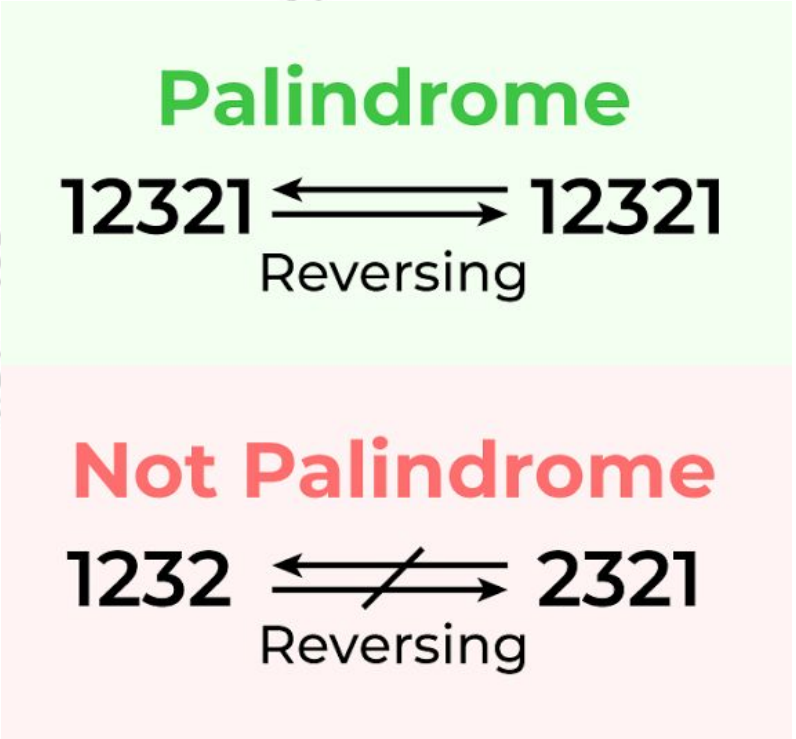
```
Simulation of Ara
=================

Tracing can be toggled by sending SIGUSR1 to this pro
$ kill -USR1 2208

Simulation running, end by pressing CTRL-c.
Factorial of 5 is 120[hw-cycles]:           0
[5616] -Info: ara_tb_verilator.sv:49: Assertion faile
Core Test *** SUCCESS *** (tohost = 0)
- /home/copi-001/ara/hardware/tb/ara_tb_verilator.sv:
Received $finish() from Verilog, shutting down simula

Simulation statistics
=====================
Executed cycles:  af8
Wallclock time:   1.16 s
Simulation speed: 2420.69 cycles/s (2.42069 kHz)
make: Leaving directory '/home/copi-001/ara/hardware'
```

Palindrome

An integer is a palindrome if the reverse of that number is equal to the original number.

A simple method is to first reverse all the digits of a given number using arithmetic operations.

Then compare the reverse of the number with a given number.



Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = palindrome

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Code Execution

```
#include <stdio.h>
#include <printf.h>
int main() {
  int n = 121, reversed = 0, remainder, original;
    printf("Enter integer: %d\n", n);
    original = n;
    // reversed integer is stored in reversed
variable
    while (n != 0) {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
    }
```

```
// palindrome if orignal and reversed are equal
    if (original == reversed)
        printf("%d is a palindrome.\n", original);
    else
        printf("%d is not a palindrome.", original);

    return 0;
}
```

Code Execution

```
File  Edit  View  Search  Terminal  Help
copi-001@copi001-OptiPlex-7050:~$ cd ara
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/palindrome
make: Entering directory '/home/copi-001/ara/apps'
cd palindrome && if [ -d script ]; then python3 script/gen_data.py  > data.S ; e
lse touch data.S; fi
/home/copi-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -mena
ble-experimental-extensions -mabi=lp64d -mno-relax -fuse-ld=lld -fvectorize -mll
vm -scalable-vectorization=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -targ
et-feature -Xclang +no-optimized-zero-stride-load -mcmodel=medany -I/home/copi-0
01/ara/apps/common -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -
DNR_LANES=4 -DVLEN=4096 -Wunused-variable -Wall -Wextra -Wno-unused-command-line
-argument  -ffunction-sections -fdata-sections -c palindrome/data.S -o palindrom
e/data.S.o
chmod +x /home/copi-001/ara/apps/common/script/align_sections.sh
rm -f /home/copi-001/ara/apps/common/link.ld && cp /home/copi-001/ara/apps/commo
n/arch.link.ld /home/copi-001/ara/apps/common/link.ld
/home/copi-001/ara/apps/common/script/align_sections.sh 4 /home/copi-001/ara/app
s/common/link.ld
/home/copi-001/ara/install/riscv-llvm/bin/clang -march=rv64gcv_zfh_zvfh0p1 -mena
ble-experimental-extensions -mabi=lp64d -mno-relax -fuse-ld=lld -fvectorize -mll
vm -scalable-vectorization=on -mllvm -riscv-v-vector-bits-min=2048 -Xclang -targ
et-feature -Xclang +no-optimized-zero-stride-load -mcmodel=medany -I/home/copi-0
01/ara/apps/common -std=gnu99 -O3 -ffast-math -fno-common -fno-builtin-printf  -
```

```c
#include <stdio.h>
int main() {
  int n, reversed = 0, remainder, original;
    printf("Enter an integer: ");
    scanf("%d", &n);
    original = n;

    // reversed integer is stored in reversed variable
    while (n != 0) {
        remainder = n % 10;
        reversed = reversed * 10 + remainder;
        n /= 10;
    }

    // palindrome if orignal and reversed are equal
    if (original == reversed)
        printf("%d is a palindrome.\n", original);
    else
        printf("%d is not a palindrome.", original);

    return 0;
}
```

Simulation on ARA

For simulation command :

make -C hardware simv app=folder_name

Where, folder_name is the program.

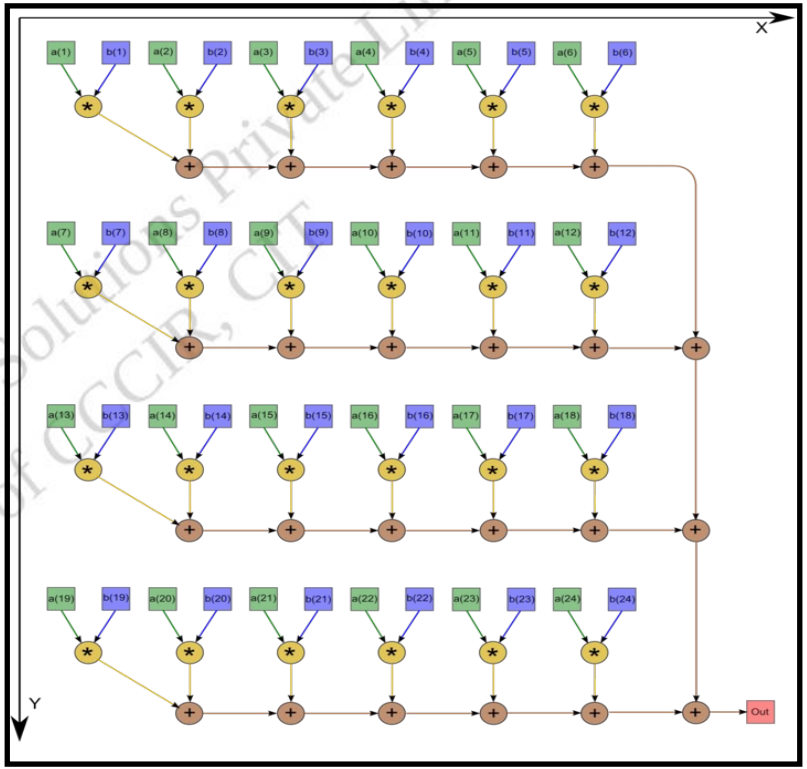
folder_name = palindrome

```
~/ara$ make -C hardware simv app=palindrome
```

```
Simulation of Ara
=================

Tracing can be toggled by sending SIGUSR1 to this pro
$ kill -USR1 12c2

Simulation running, end by pressing CTRL-c.
Enter integer: 121
121 is a palindrome.
[hw-cycles]:           0
[7002] -Info: ara_tb_verilator.sv:49: Assertion faile
Core Test *** SUCCESS *** (tohost = 0)
- /home/copi-001/ara/hardware/tb/ara_tb_verilator.sv:
Received $finish() from Verilog, shutting down simula

Simulation statistics
=====================
Executed cycles:  dad
Wallclock time:   1.524 s
Simulation speed: 2297.24 cycles/s (2.29724 kHz)
make: Leaving directory '/home/copi-001/ara/hardware'
```

Dot Product

A dot product is a combination of multiplication and addition between two vectors. Given two vectors
A = [*aa bb cc*] and D = [*dd ee ff*]
The dot product A · D is defined by ,A · D = ad + be + cf



Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = dot_product

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Dot product

```
#include <stdio.h>

int main()
{
   int a[3] = {1, 2, 3};
   int b[3] = {4, 5, 6};
   int sum = 0;
   for (int i = 0; i < 3; ++i)
   {
      sum += a[i] * b[i];
   }
   printf("The inner product of two vectors is %d", sum);
   return 0;
}
```

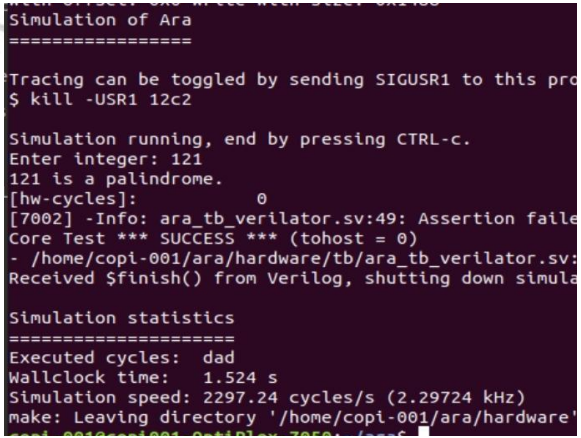
Simulation on ARA

For simulation command :

make -C hardware simv app=folder_name

Where, folder_name is the program.

folder_name = dot_product



Length of last string

```
#include <stdio.h>
#include <string.h>
int main() {

const char* str1 = "Hello World";
const char* str2 = "Length of the last word is ";

        printf("%s\n", str1);
        int len = strlen(str1);
        int count = 0;
        int i = len - 1;

        while (i >= 0 && str1[i] == ' ') {
        i--;
        }

        while (i >= 0 && str1[i] != ' ') {
        count++;
        i--;
        }

        printf("%s%d\n", str2, count);

        return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = string

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

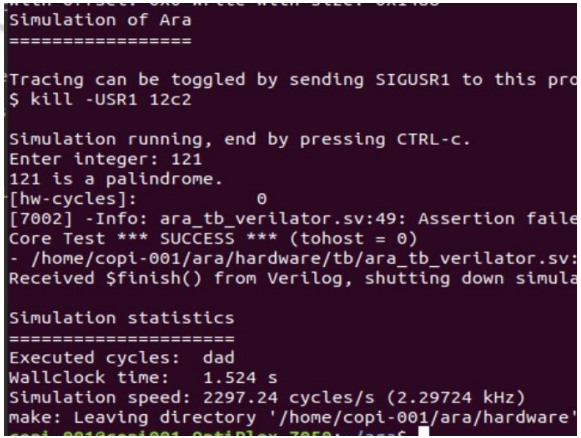
Simulation on ARA

For simulation command :

make -C hardware simv app=folder_name

Where, folder_name is the program.

folder_name = string



Matrix Multiplication

```
#include <stdio.h>
#define N 3  // Matrix size (N x N)
void matrix_multiply(int A[N][N],
int B[N][N], int C[N][N]) {
   for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
              C[i][j] = 0;
              for (int k = 0; k < N; k++) {
              C[i][j] += A[i][k] * B[k][j];
              } } } }
int main() {
   int A[N][N] = {{1, 2, 3},
              {4, 5, 6},
              {7, 8, 9}};
int B[N][N] = {{9, 8, 7},
              {6, 5, 4},
              {3, 2, 1}};
   int C[N][N] = {0};
   matrix_multiply(A, B, C);
   printf("Resultant Matrix C:\n");
   for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
              printf("%d ", C[i][j]);
        }
        printf("\n");
   }
   return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = matrix

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

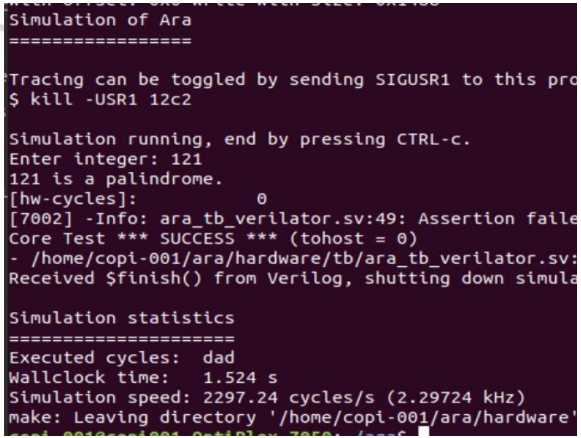
Simulation on ARA

For simulation command :

make -C hardware simv app=folder_name

Where, folder_name is the program.

folder_name = matrix



Matrix Transpose

```
#include <stdio.h>
#define N 4
 void transpose(int A[][N], int B[][N])
{
        int i, j;
        for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
        // Assigns the transpose of
element A[j][i] to
        // B[i][j]
        B[i][j] = A[j][i];
}
int main()
{
        int A[N][N] = { { 1, 1, 1, 1 },
                { 2, 2, 2, 2 },
                { 3, 3, 3, 3 },
                { 4, 4, 4, 4 } };
        int B[N][N], i, j;
        transpose(A, B);
        printf("Result matrix is \n");
        for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++)
        printf("%d ", B[i][j]);
        printf("\n");
        }
        return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = transpose

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Matrix Addition

```
#include <stdio.h>
#define ROWS 2
#define COLUMNS 3
void addMatrices(int
A[ROWS][COLUMNS], int
B[ROWS][COLUMNS], int
C[ROWS][COLUMNS]) {
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        C[i][j] = A[i][j] + B[i][j];
        }
        }
}
int main() {
        int A[ROWS][COLUMNS] = {
        {1, 2, 3},
        {4, 5, 6}
        };
int B[ROWS][COLUMNS] = {
        {6, 5, 4},
        {3, 2, 1}
        };
        int C[ROWS][COLUMNS];
        addMatrices(A, B, C);
        printf("Resultant Matrix (A + B):\n");
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        printf("%d ", C[i][j]);
        }
        printf("\n");
        }
        return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = addition

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Matrix Subtraction

```
#include <stdio.h>
#define ROWS 2
#define COLUMNS 3
void subtractMatrices(int
A[ROWS][COLUMNS], int
B[ROWS][COLUMNS], int
C[ROWS][COLUMNS]) {
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        C[i][j] = A[i][j] - B[i][j];
        }
        }
}
int main() {
        int A[ROWS][COLUMNS] = {
        {10, 20, 30},
        {40, 50, 60}
        };
int B[ROWS][COLUMNS] = {
        {1, 2, 3},
        {4, 5, 6}
        };
        int C[ROWS][COLUMNS];
        subtractMatrices(A, B, C);
        printf("Resultant Matrix (A - B):\n");
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        printf("%d ", C[i][j]);
        }
        printf("\n");
        }
        return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = subtraction

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```

Matrix Division

```
#include <stdio.h>
#define ROWS 2
#define COLUMNS 3
void divideMatrices(int A[ROWS][COLUMNS], int
B[ROWS][COLUMNS], float
C[ROWS][COLUMNS]) {
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        if (B[i][j] != 0)
                C[i][j] = (float)A[i][j] / B[i][j];
        else
                C[i][j] = 0.0; // or handle error
        } } }
int main() {
        int A[ROWS][COLUMNS] = {
        {10, 20, 30},
        {40, 50, 60}
        };
        int B[ROWS][COLUMNS] = {
        {2, 4, 5},
        {8, 10, 15}
        };
        float C[ROWS][COLUMNS];
        divideMatrices(A, B, C);
        printf("Resultant Matrix (A / B):\n");
        for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLUMNS; j++) {
        printf("%.2f ", C[i][j]);
        }
        printf("\n");
        }
        return 0;
}
```

Source file creation

- To create a new folder in the directory type the command mkdir followed by the folder name, in which the saved program will be store.
- Example : folder_name = Program_name = division

```
copi-001@copi001-OptiPlex-7050:~/ara$ cd apps
copi-001@copi001-OptiPlex-7050:~/ara/apps$ mkdir folder_name
copi-001@copi001-OptiPlex-7050:~/ara/apps$ cd folder_name/
```

Code Compilation

- To create a new file to write the c code using editor type command nano followed by .c extension.

```
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ nano main.c
copi-001@copi001-OptiPlex-7050:~/ara/apps/folder_name$ cd ../..
copi-001@copi001-OptiPlex-7050:~/ara$ make -C apps bin/folder_name
```