THEIA GPU

Open Source multicore programmable GPU

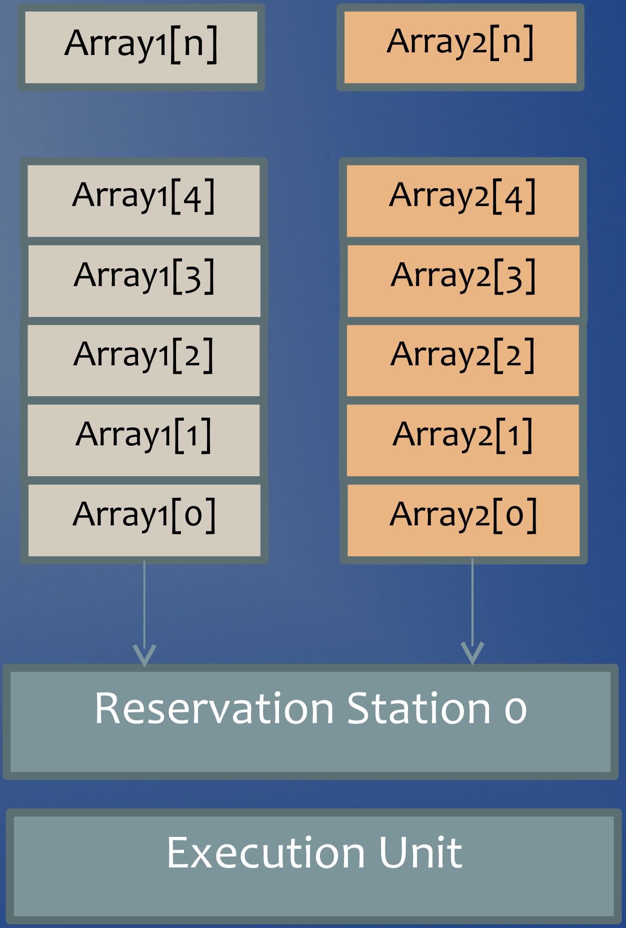
Problem Statement

- Develop an open source 3D Graphic Processor (GPU).
- Develop a high level language to program the GPU.
- Provide all of the necessary tools, test-bench and regressions.
- Should be different from current state-of-the-art (at least a little different).

What kind of GPU?

- Vector Processing.
- Multiple hardware threads.
- Multiple cores.
- Out-of-order execution.
- And many other funky stuff...

VECTOR PROCESSING ADDS DATA LEVEL PARALELISM

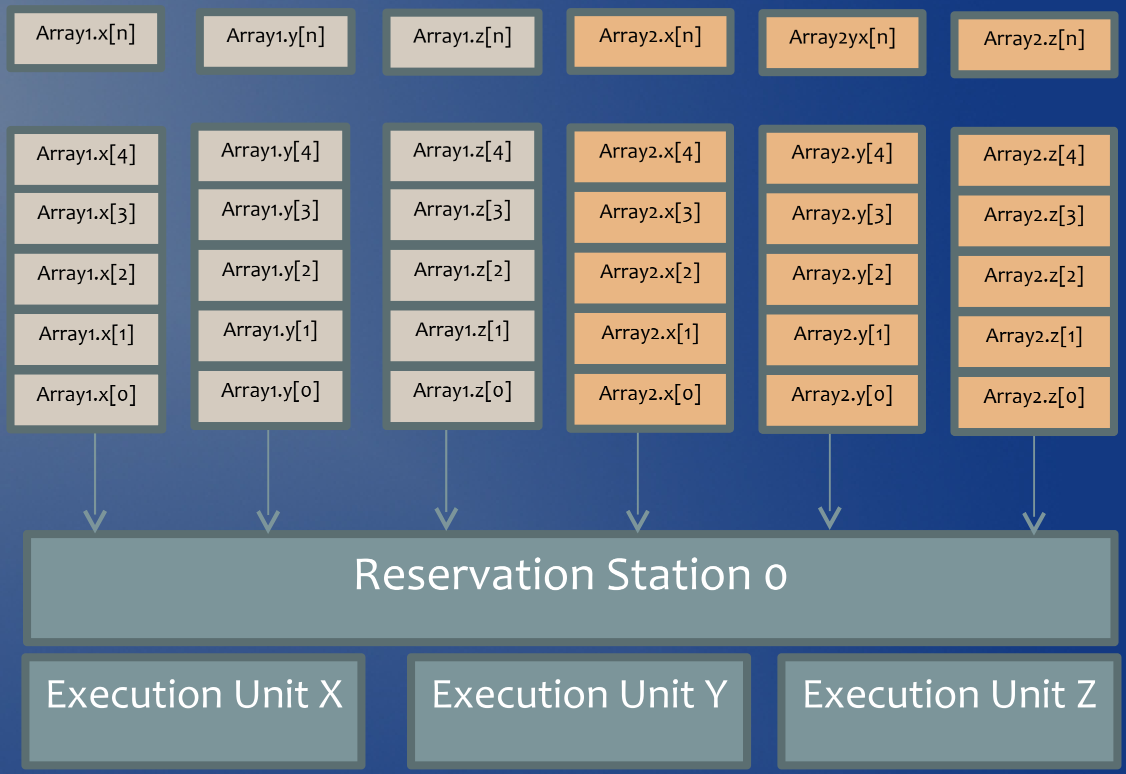


- Instructions operates on “Ranges” of registers instead of operating on single registers.
- Example
- R3[50:10] = R1[50:10] + R2[50:10]

3 Data LANES adds further parallelism to vector operations

Each Execution unit is replicated three times for parallel execution .

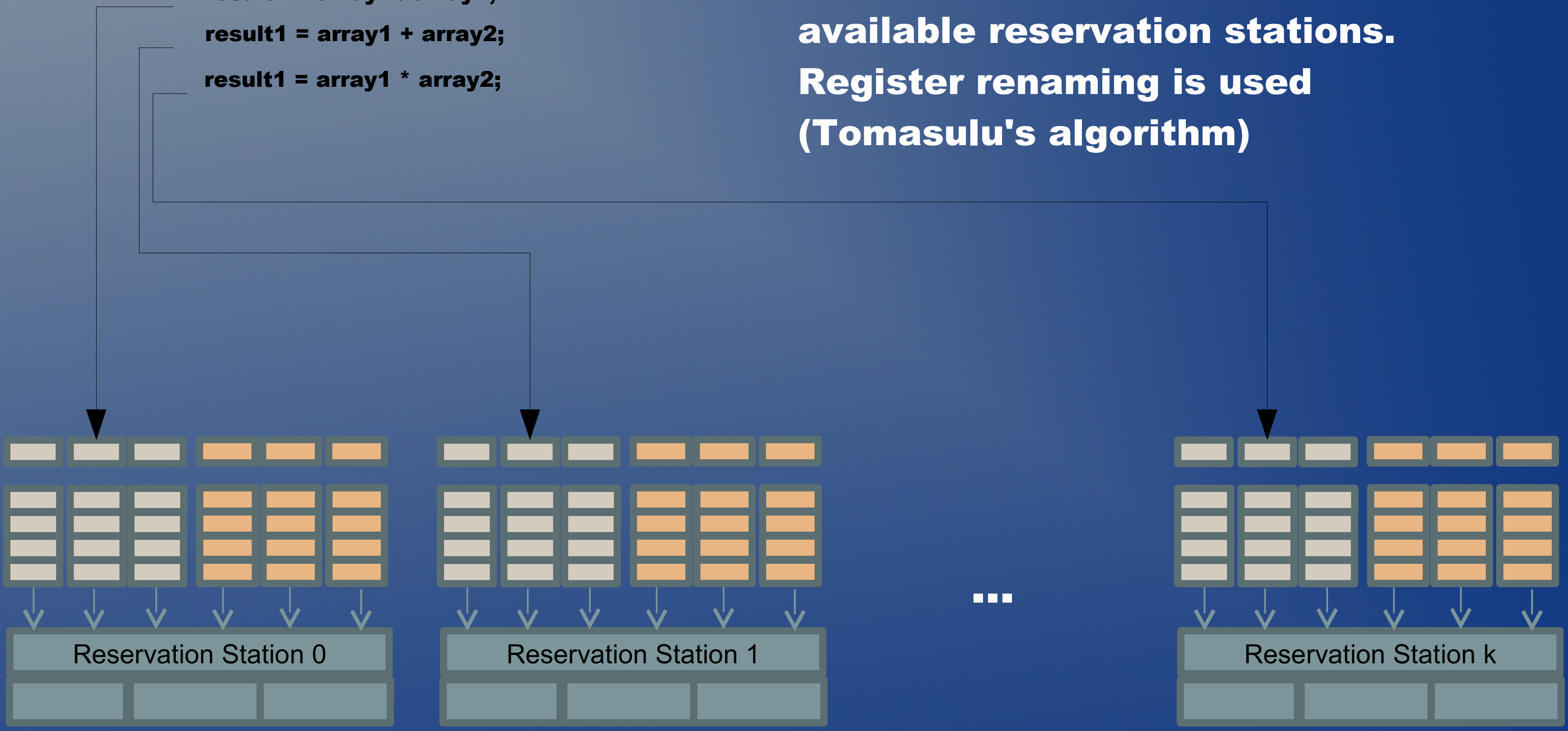
Memory locations are logically divided into x, y and z components (32 bits each)



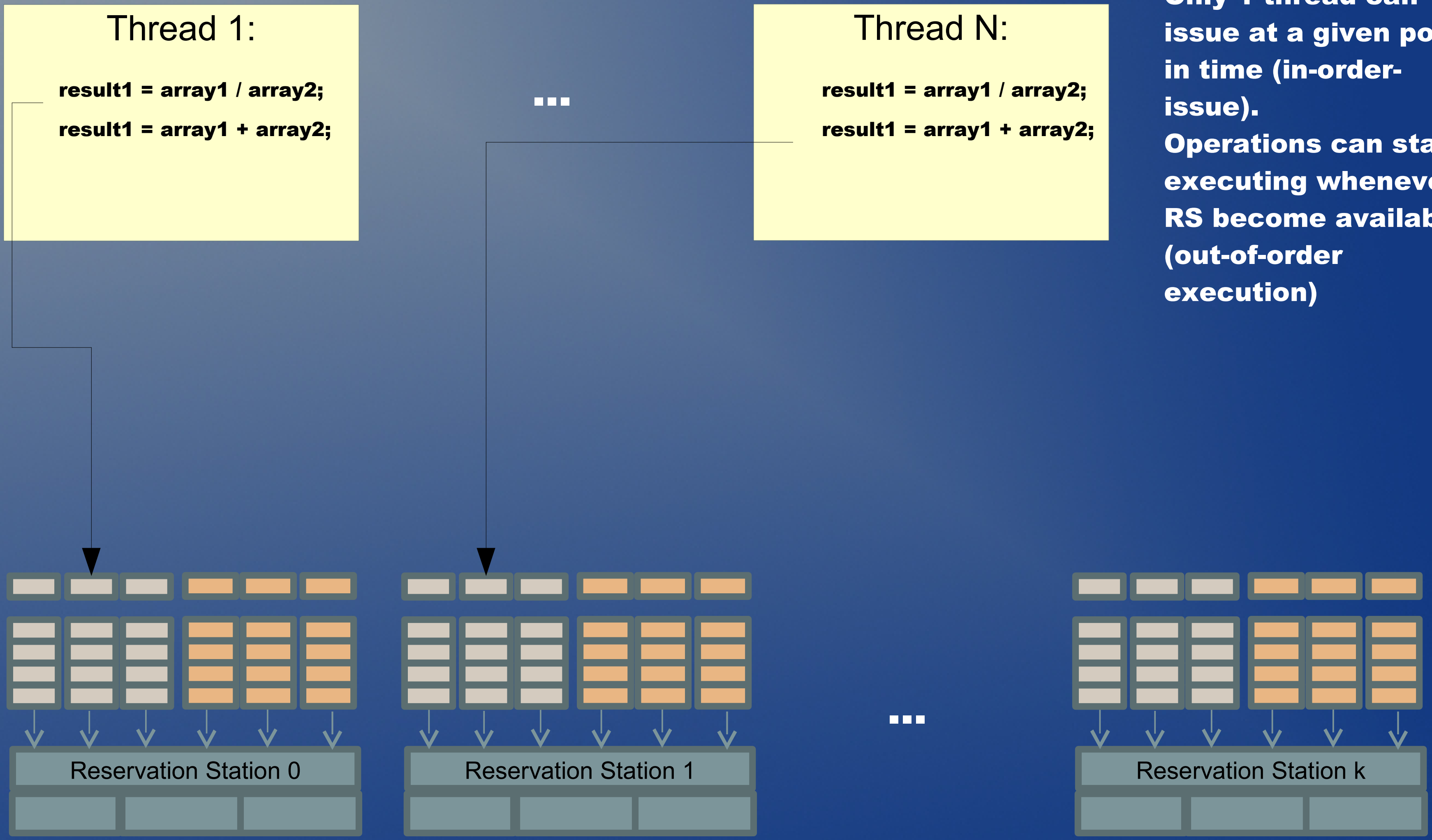
More parallelism: Out of order execution of the vector operations

```
vector array1[10],array2[10];

vector result[10],result[10],result3[10];

result1 = array1 / array2;
result1 = array1 + array2;
result1 = array1 * array2;
```

Vectors operations can be executed out of order as long as as there are available reservation stations. Register renaming is used (Tomasulu's algorithm)



Simultaneous multi-threading (SMT)

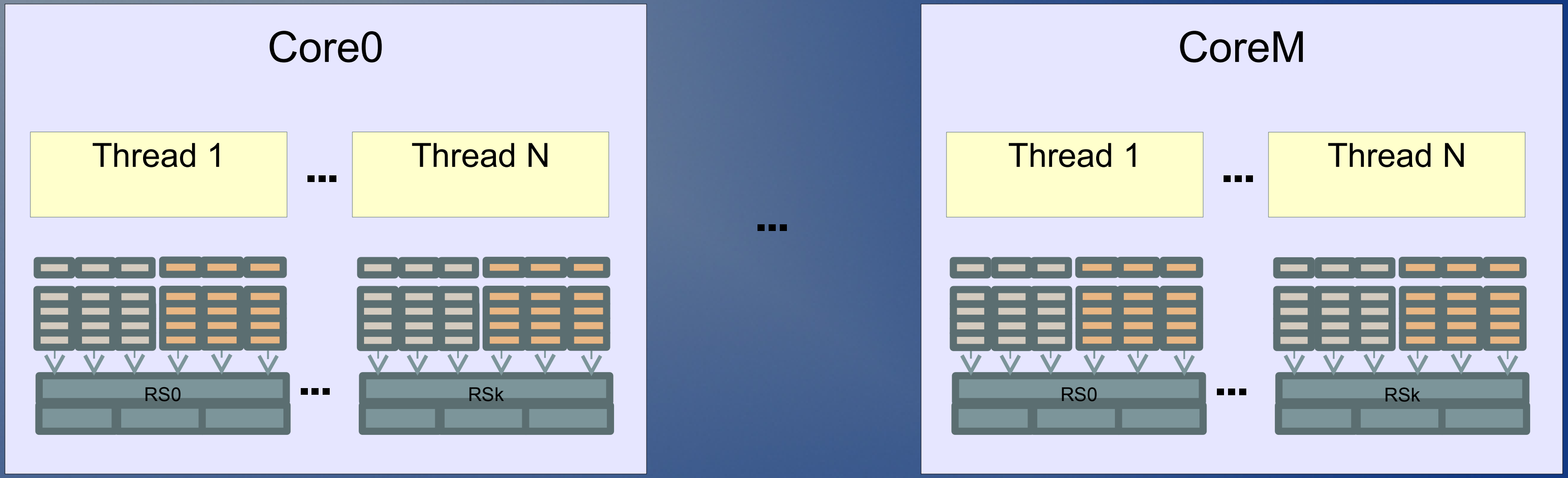


```
result1 = array1 / array2;
result1 = array1 + array2;
```

```
result1 = array1 / array2;
result1 = array1 + array2;
```

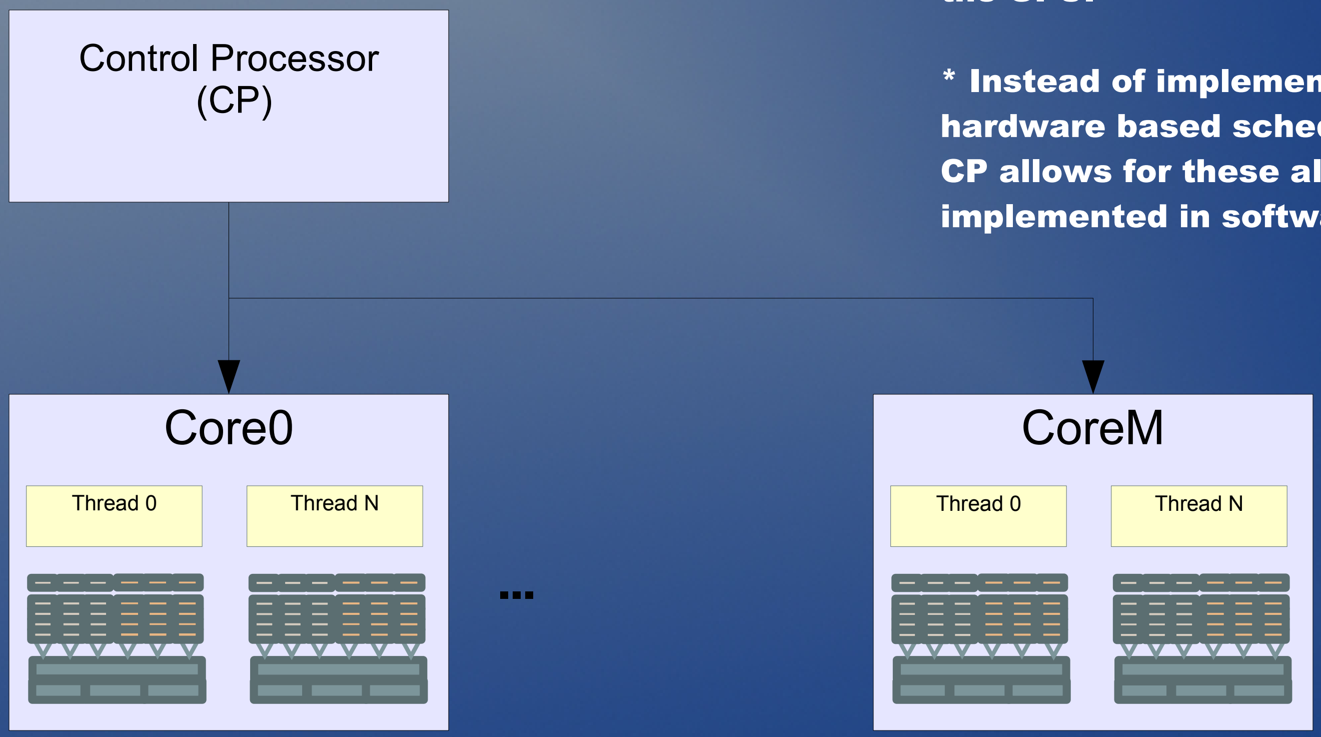
Only 1 thread can issue at a given point in time (in-order-issue).
Operations can start executing whenever RS become available (out-of-order execution)

Multiple Vector processing Cores



Multiple vector processing cores operate in parallel. Each core vector processing core executes multiple threads in parallel.

Control processor handles Load and resource distribution of the system



* **The CP allows the user to programmatically control the resource allocation and the workload distribution of the GPU.**

* **Instead of implementing complex dynamic hardware based scheduling algorithms, the CP allows for these algorithms to be implemented in software.**

The control processor

The CP controls the global execution of the system

The CP does not process data, it only schedules the data processing that will occur in the VPS

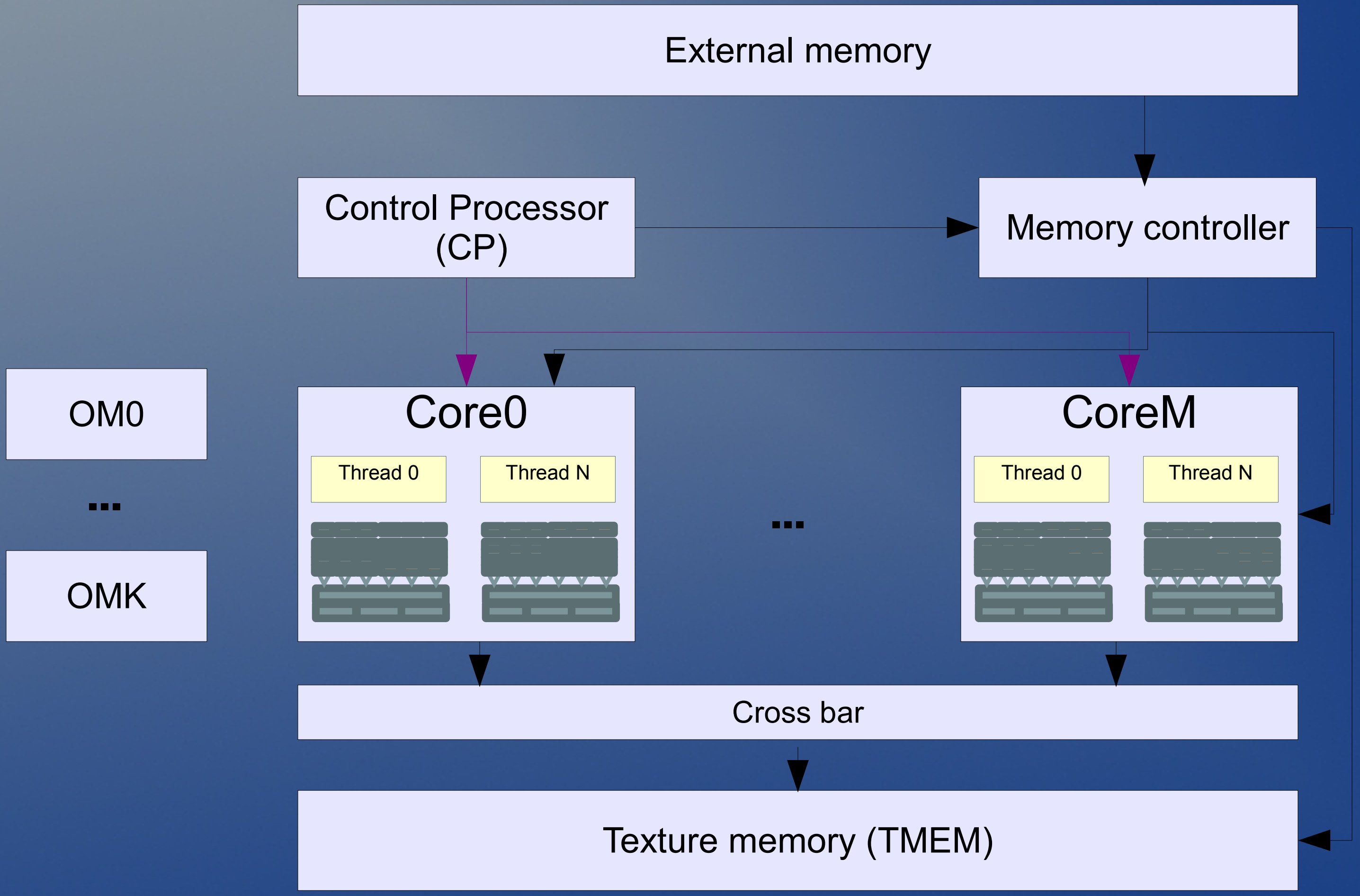
The CP is a simple but fully programmable processor.

A special extension of the high level language has been developed specifically for the CP.

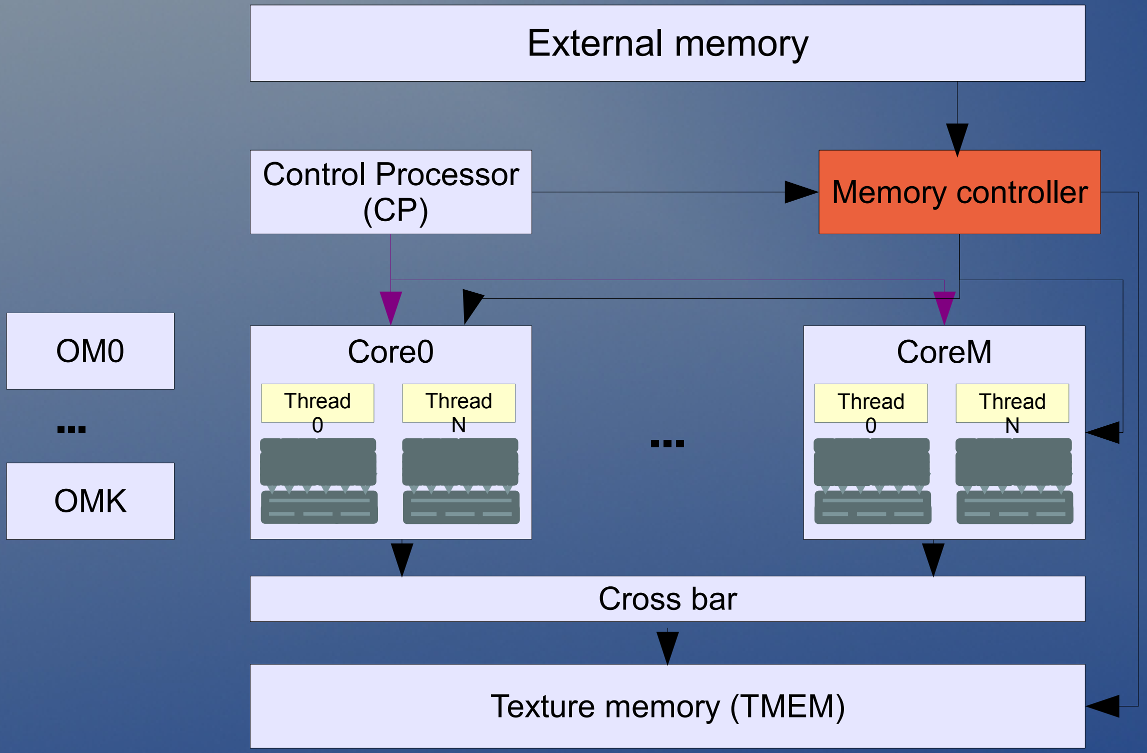
The CP controls the interface between the VP cores and the GPU memory

```
#include "theia.thh"
#include "code_block_header.thh"


scalar DstOffsetAndLen,SrcOffset,CoredId;
//First send the data into cores

SrcOffset = 0;
DstOffsetAndLen = (0x0 | (CORE_INPUT_AREA_SIZE << 20)  );

 while (CoredId <= THEIA_CAPABILTIES_MAX_CORES)
 {
               copy_data_block< CoredId, DstOffsetAndLen, SrcOffset>;
               SrcOffset += INPUT_DATA_LEN;
               CoredId++;
}

 //wait until enqueued block transfers are complete
  while ( block_transfer_in_progress ) {}


  SrcOffset = SIMPLE_RENDER_OFFSET;
  DstOffsetAndLen = (0x0 | SIMPLE_RENDER_SIZE | VP_DST_CODE_MEM );
  copy_data_block < ALLCORES , DstOffsetAndLen  ,SrcOffset>;

  start <ALLCORES>;

  exit ;
```

Memories and the memory controller



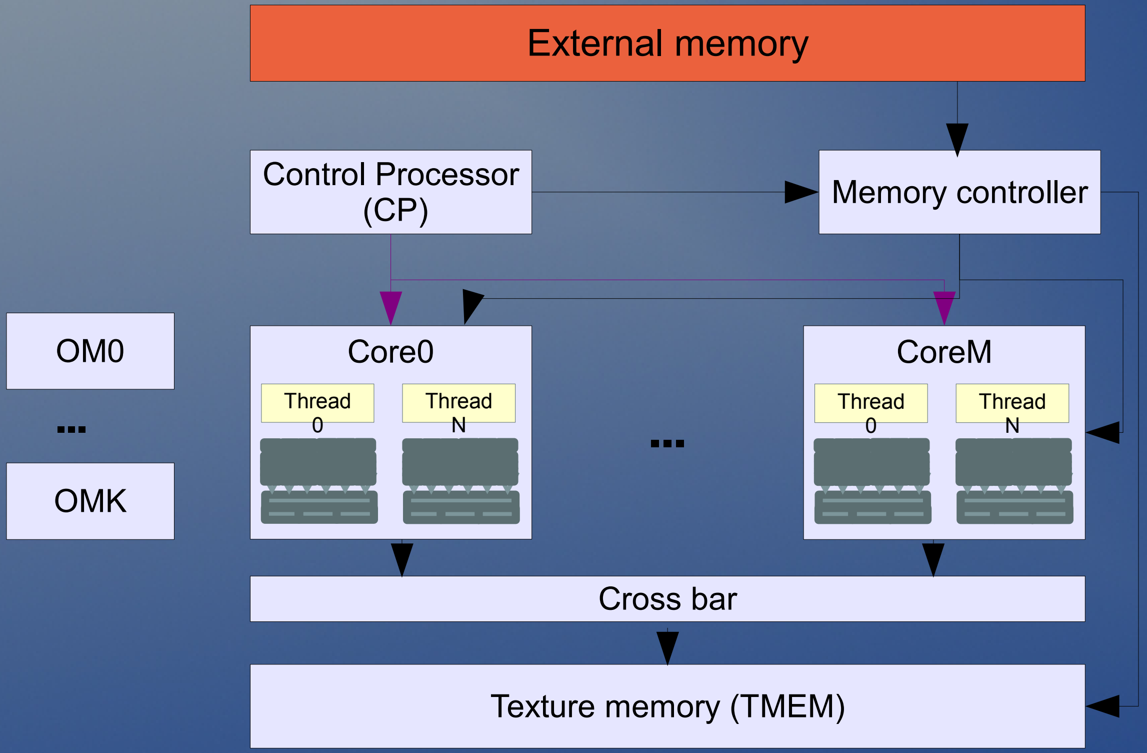
The memory controller



Takes care of transferring data from the "external memory" to the Texture memory or the vector processors.

The CP controls the memory controller, issuing asynchronous block transfer commands

The external memory



Used by the CPU in order to read or read data for the GPU to process.

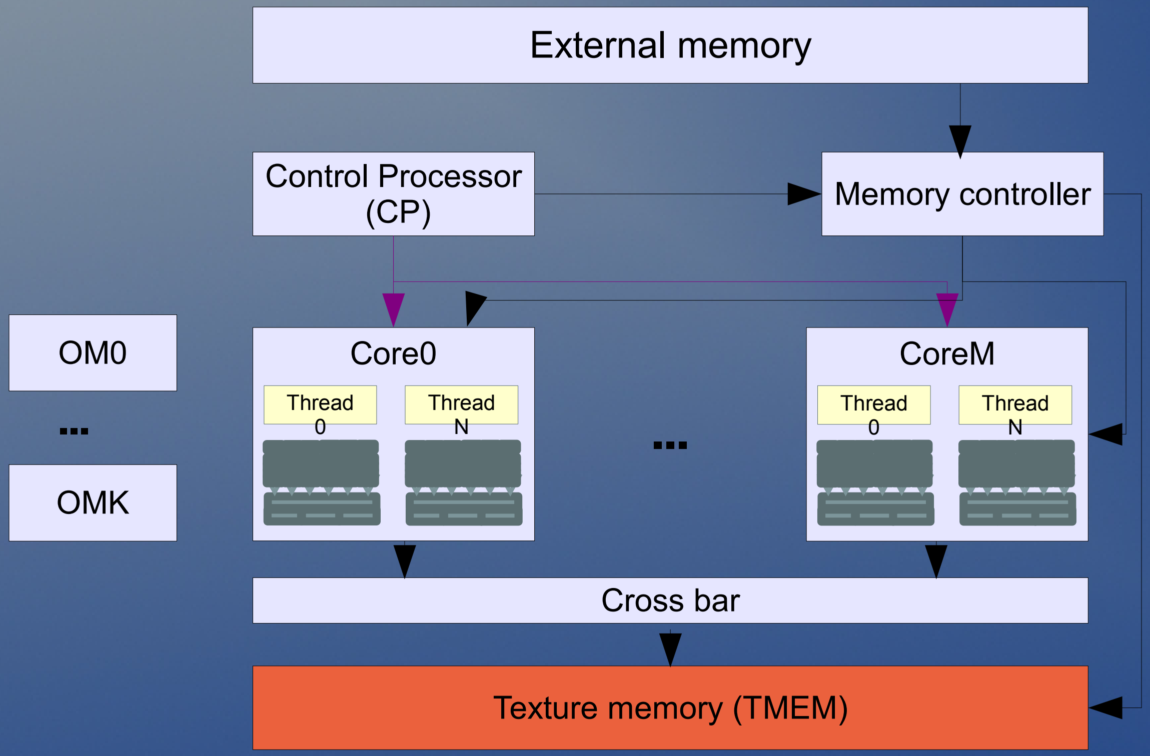
Can store GPU code or data

Is not part of the GPU, per-se.

Conceptually a large RAM.

GPU can only access this memory via the Memory controller.

The texture memory



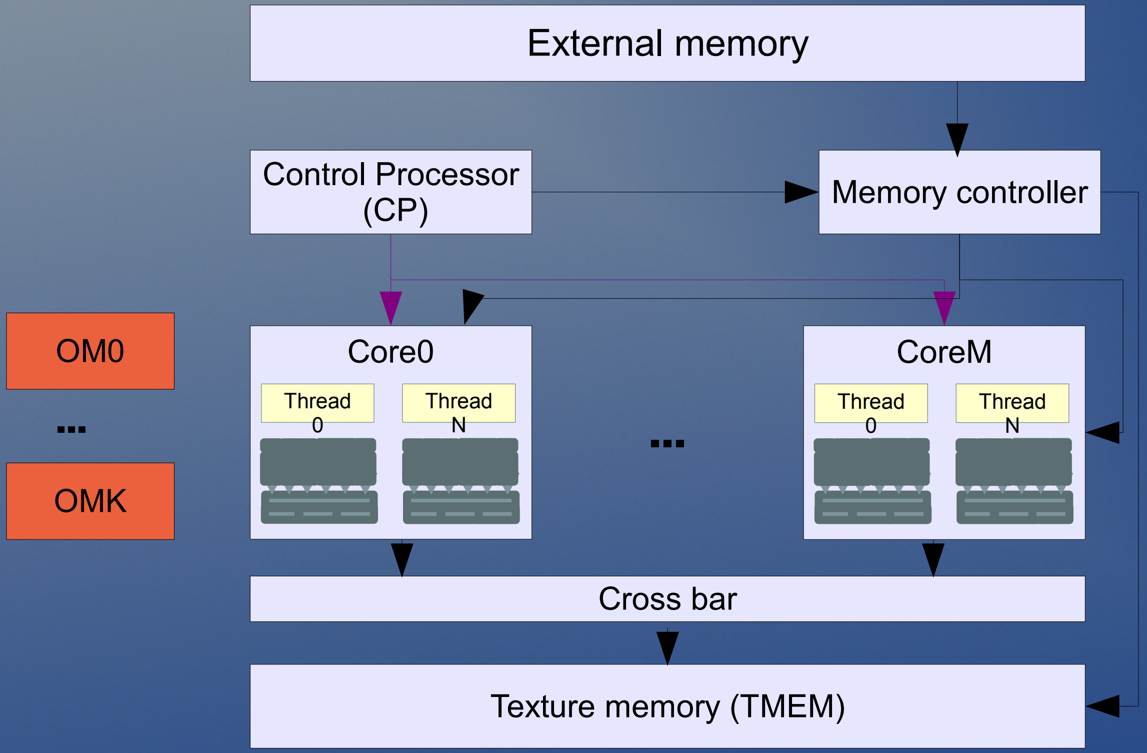
Read-Only from the vector processor perspective.

Multiple VPs can simultaneously read using a full mesh cross bar.

Only Memory controller can write into TMEM.

Default store location for texture data (although the CP code can decide to store anything in there)

The output memories



Write-Only from the vector processor perspective.

Each VP can only write into a single and unique OM.

Each OM is “owned” a VP to do write operations, the OM cannot be shared.

Default store location for program result data. The CP can request the OM data to be transfer back into the external memory, or into the graphics frame buffer

Programming the GPU

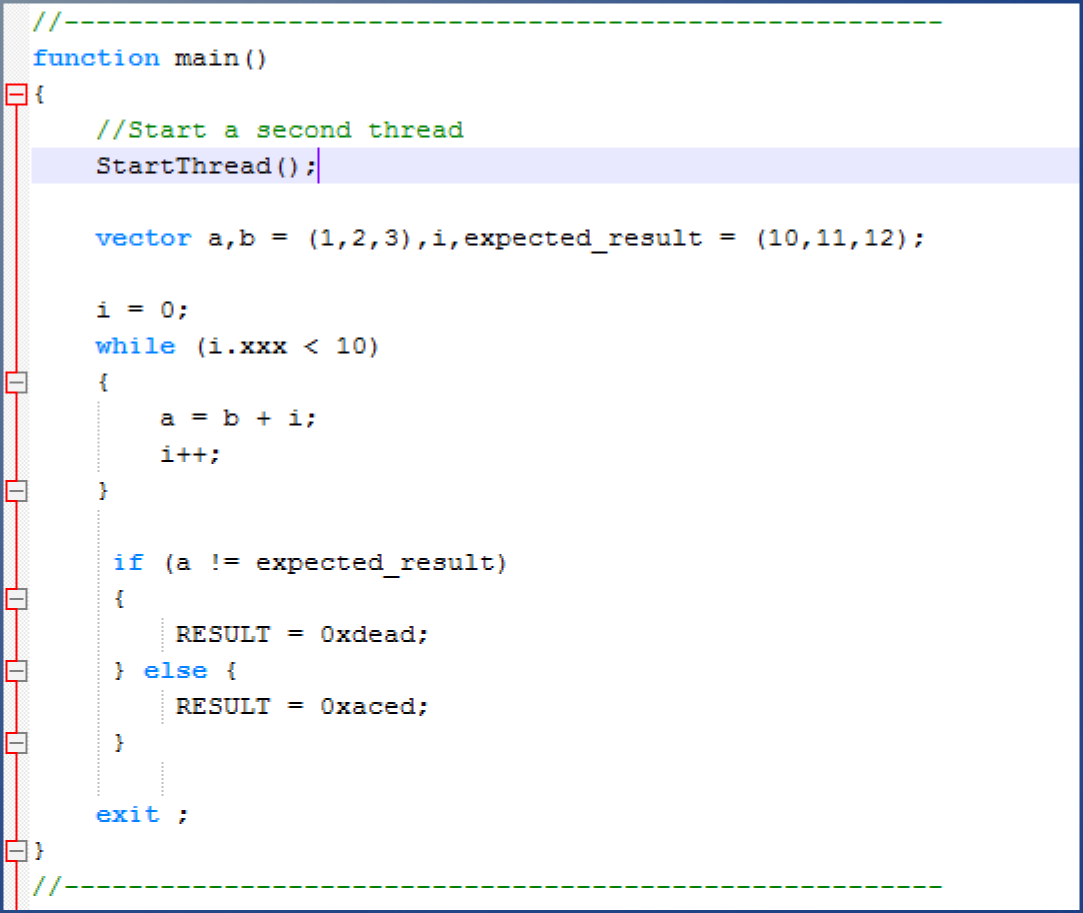
* Has a high level programming language called “T-Language”.

* Reminds of C but designed for 3D operations.

* Clean exposes the features of the hardware with no need for the user to know about low-level details.

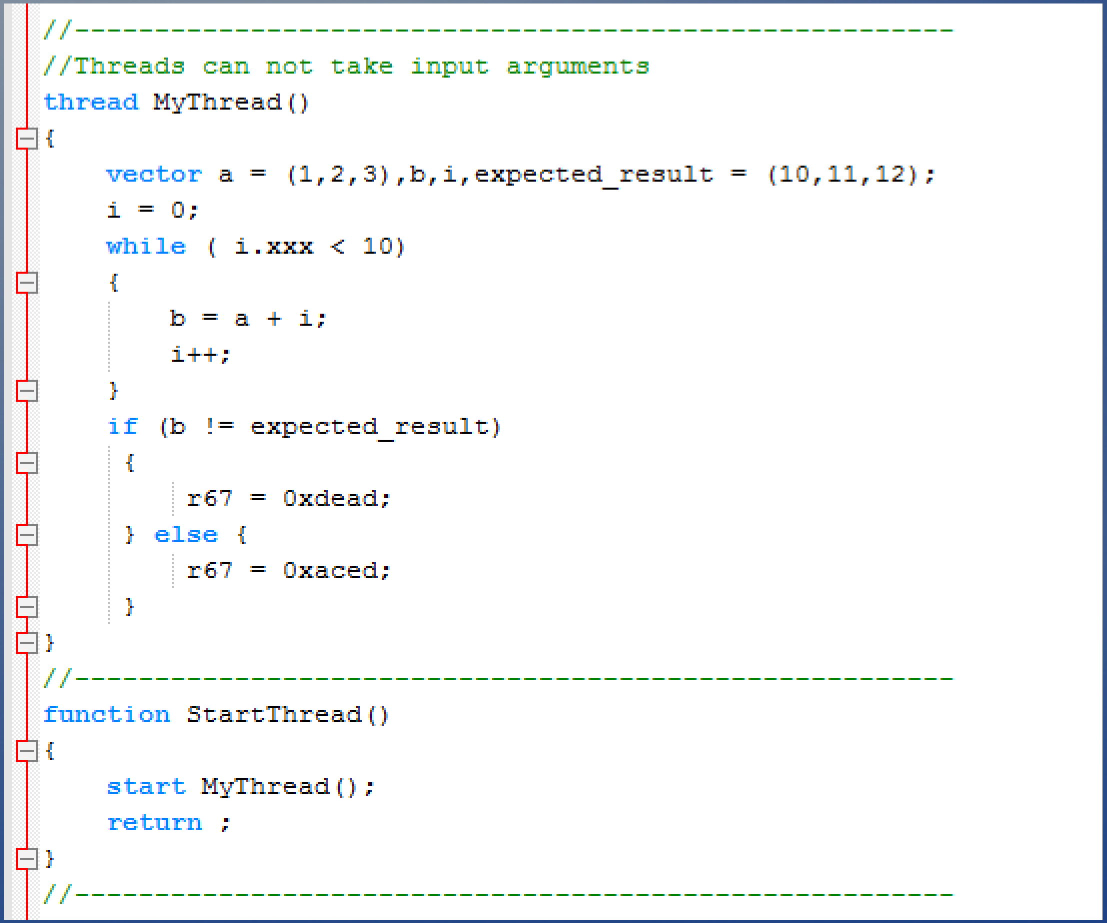
* User writes separate code for the CP and the VP (grammar is similar, but features change)

How does the VP code looks like?



```
//-----------------------------------------------------
function main()
{
    //Start a second thread
    StartThread();

    vector a,b = (1,2,3),i,expected_result = (10,11,12);

    i = 0;
    while (i.xxx < 10)
    {
        a = b + i;
        i++;
    }

     if (a != expected_result)
     {
         RESULT = 0xdead;
     } else {
         RESULT = 0xaced;
     }

    exit ;
}
//-----------------------------------------------------
```

How does the VP code looks like?



```
//--------------------------------------------------------
//Threads can not take input arguments
thread MyThread()
{
    vector a = (1,2,3),b,i,expected_result = (10,11,12);
    i = 0;
    while ( i.xxx < 10)
    {
        b = a + i;
        i++;
    }
    if (b != expected_result)
     {
         r67 = 0xdead;
     } else {
         r67 = 0xaced;
     }
}
//--------------------------------------------------------
function StartThread()
{
    start MyThread();
    return ;
}
//--------------------------------------------------------
```

T-Language allows thread declaration as part of the grammar.

Variables are declared as “Vector” data types, 3D vectors divided into x, y and z.

Allows subroutines, variable stacks, arrays and many other things