If more than 1 memory loads or 1 store accesses are present in a instruction and these accesses are:
- Coalescent accesses
- Strided access with stride <= cache_line_size

The memory accesses are fused into one big request.

In order to be made once, make the minimum amount of modifications, and do not disturb the MOB management made on the **decode** stage, this code is implemented in the **fetch** stage.

This implementation do not cover the situation in which many elements are overlapping, and one of the middle ones is huge, ending after the (strides + last element size) calculation.

Location: Fetch

```
// The new code was inserted after the branch control and before the fetch buffer insertion

if (operation.opcode_operation == INSTRUCTION_OPERATION_BRANCH)
{ … }

<HERE>

//============================
//Insert into fetch buffer
//============================

if (POSITION_FAIL == this->fetchBuffer.push_back(operation))
{
	break;
}
```

Code

```
//============================
// Memory accesses fusing
//============================

if (operation.num_reads > 1) {
  // Check for stride
  bool is_strided;
  uint64_t stride = get_instruction_loads_stride(&operation, &is_strided);

  if (is_strided && stride <= LINE_SIZE) { // Smaller than the cache line size
    // ***************************************
    // Fuse loads
    // ***************************************
    uint64_t initial_address = operation.reads_addr[0];
    uint64_t accesses_size = (stride * (operation.num_reads - 1)) + operation.reads_size[operation.num_reads - 1];

    // Adjust requests
    operation.reads_addr[0] = initial_address;
    operation.reads_size[0] = accesses_size;
    operation.num_reads = 1;
  }
}

if (operation.num_writes > 1) {
  // Check for stride
  bool is_strided;
  uint64_t stride = get_instruction_stores_stride(&operation, &is_strided);

  if (is_strided && stride <= LINE_SIZE) { // Smaller than the cache line size
    // ***************************************
    // Fuse stores
    // ***************************************
    uint64_t initial_address = operation.writes_addr[0];
    uint64_t accesses_size = (stride * (operation.num_writes - 1)) + operation.writes_size[operation.num_writes - 1];

    // Adjust requests
    operation.writes_addr[0] = initial_address;
    operation.writes_size[0] = accesses_size;
    operation.num_writes = 1;

  }
}

//============================
```

get_instruction_loads_stride (operation, bool *is_strided)

```
uint64_t stride = operation->reads_addr[1] - operation->reads_addr[0];

// For each load, check the stride
for (uint32_t i=2; i < operation->num_reads; ++i) {
  if (stride != (operation->reads_addr[i] - operation->reads_addr[i-1])) {
    *is_strided = false;
    return 0;
  }
}
*is_strided = true;
return stride;
```

get_instruction_stores_stride (operation, bool *is_strided)

```
uint64_t stride = operation->writes_addr[1] - operation->writes_addr[0];

// For each load, check the stride
for (uint32_t i=2; i < operation->num_writes; ++i) {
  if (stride != (operation->writes_addr[i] - operation->writes_addr[i-1])) {
    *is_strided = false;
    return 0;
  }
}
*is_strided = true;
return stride;
```