NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

HETEROGENEOUS PERSONAL COMPUTING: A CASE STUDY IN MATERIALS SCIENCE

This work was supported by FCT - Fundação para a Ciência e Tecnologia, under project UID/CEC/04516/2013 (NOVA LINCS research center)

Nuno Oliveira

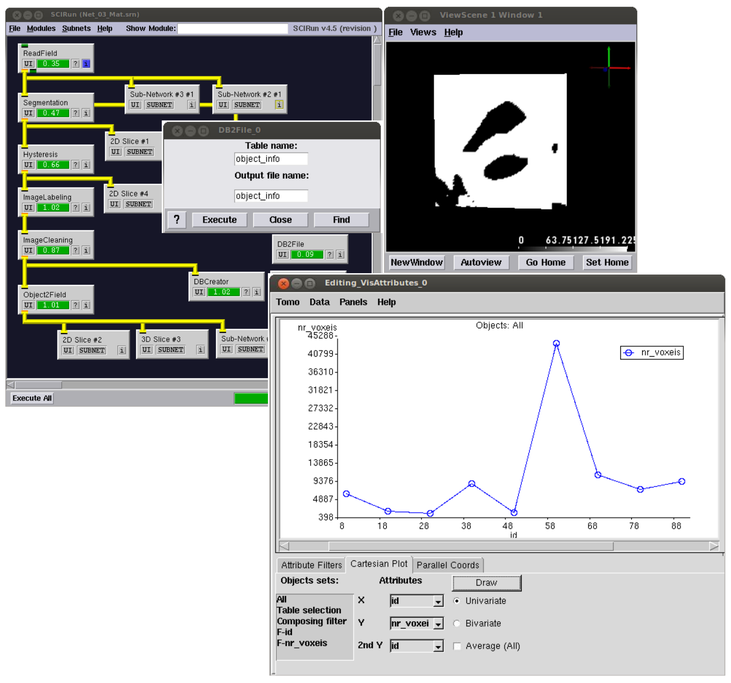
PhD Student

Supervisor: Pedro Medeiros, UNL

My research focuses on heterogeneous multi-core platforms and parallel programming.

Motivation for HRTE

- Scientific computing has been evolving towards the use of heterogeneous architecture encompassing classical CPUs, GPUs, etc.
- Visual programming environments (**PSE - Problem Solving Environment)** have been successfully used by scientists allowing the easy exploitation of the emerging parallel architectures.
- Most of these environments are based on the workflow paradigm: a program is a set of processing modules organized as a pipeline. Modules are interconnected by logical channels transferring huge data sets.



- The main objective of this work is to build a **Heterogeneous Runtime Environment (HRTE)** that efficiently supports the execution of programs defined by a visual program environment, to be executed in a desktop PC with one or more accelerators.

HRTE Main Characteristics

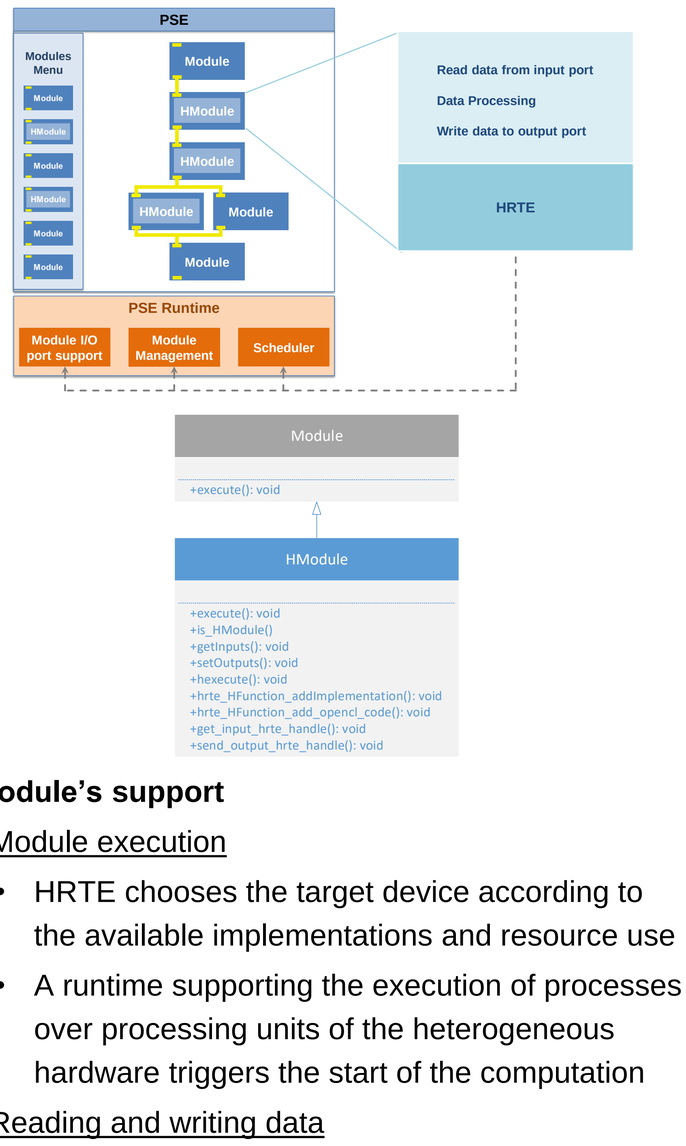
Supports the combination of existing modules with **Heterogeneous Modules (HModules)**

- **HModules** can be executed in different platforms chosen at runtime
- Transparent management of data copy between main memory and accelerator's memory, including semi automatic data partition.

HModules and HRTE

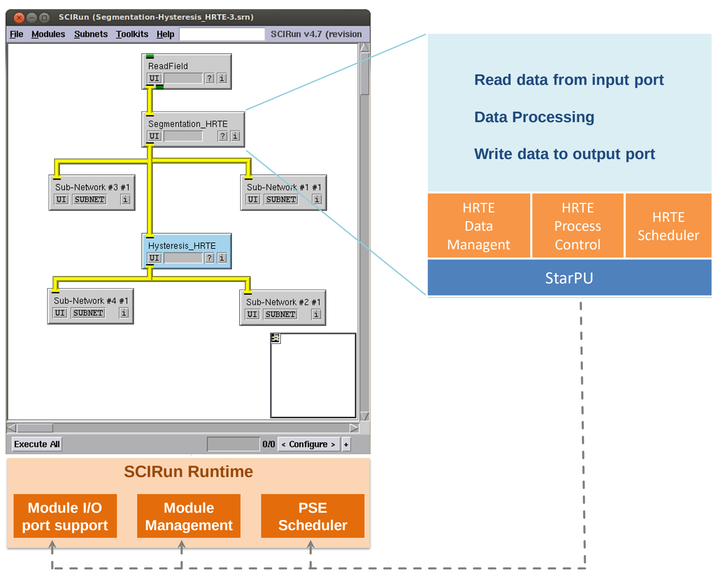
HModules

- Can have several implementations over different hardware and parallel runtime environments
- Can be interconnected with other **HModules** or classical Modules
- Minimizing intrusion in PSE code
- **PSE** scheduler sees **HModules** as normal Modules



HModule's support

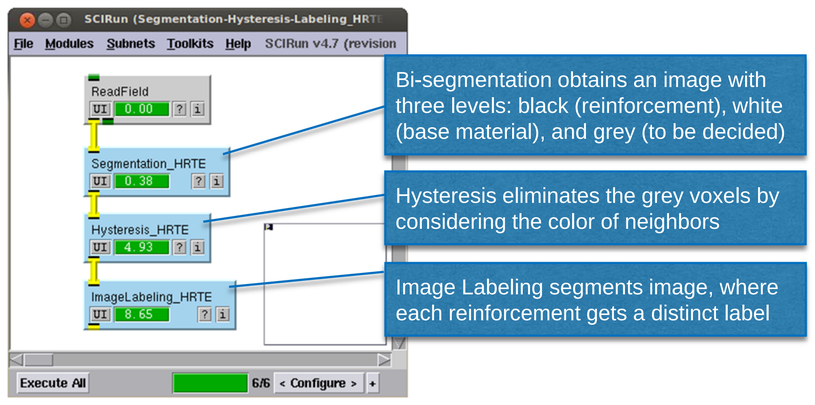
- Module execution
  - HRTE chooses the target device according to the available implementations and resource use
  - A runtime supporting the execution of processes over processing units of the heterogeneous hardware triggers the start of the computation
- Reading and writing data
  - *getInputs()* and *setOutputs()* operations uses a virtual shared address space accessed through an *handle* that abstracts a vector or matrix
  - The runtime transparently manages the movement of data along the hierarchy of memory of the heterogeneous hardware



A prototype of HRTE exists using SCIRun PSE toolkit and StarPU runtime environment.

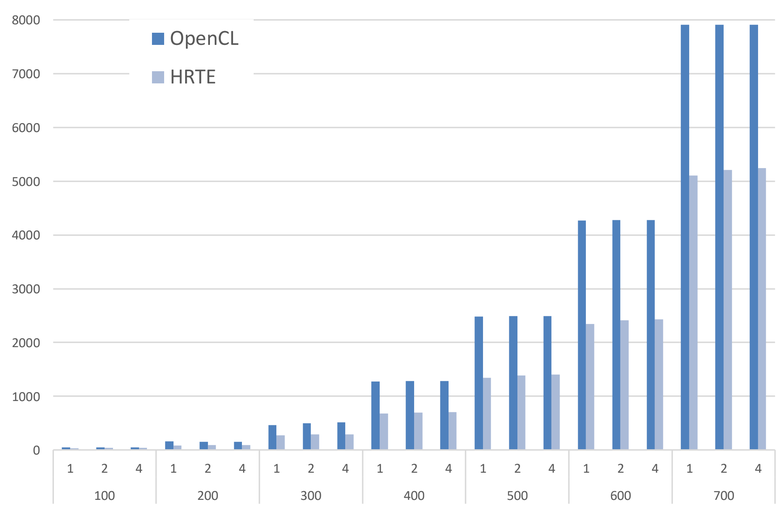
Case Study

Processing of tomographic images of composite materials: one that constitutes the base matrix and another that acts as reinforcements



Code fragment corresponds to the hysteresis module:

```
class Hysteresis_HRTE : public HModule {
public:
  void hexecute() {
    hrte_data_handle matImage, matImageaux;
    ...
    // read module input image and set partitions' number to be used
    get_input_hrte_handle("Input", matImage, ...);
    hrte_matrix3d_set_partitions(matImage, nPartitions);
    // create an auxiliar image with the same partitions of read image
    hrte_matrix3d_create(&matImageaux, nx, ny, nz, ...);
    hrte_matrix3d_set_partitions(matImageaux, nPartitions);
    // the 1st kernel will be apply to image until there is no change
    for (bool thereAreChanges = true, int iter = 0; thereAreChanges; ++iter) {
      // 1st kernel: use stencil pattern to get the new image - each voxel is
      // equals to the majority of neighbors
      hrte_task_stencil(hf_hysteresisFirstPhase, (iter % 2 == 0) ?
                  matImage:matImageaux, (iter % 2 == 0) ? matImageaux:matImage);
      thereAreChanges = !hrte_task_isEquals(matImage,matImageaux);
    }
    // 2nd kernel: use stencil to eliminate grey voxels adopting the black or
    // white in function of the neighborhood
    hrte_task_stencil(hf_hysteresisSecondPhase, matImageaux, matImage);
    send_output_hrte_handle("Output", matImage);
  }
}; // end of class Hysteresis_HRTE
__kernel void hysteresis_firstStep ( ... ) {
  ...
  // set voxel with the value of the majority of neighbors
  countNeighborhood(blockin,..., x, y, z, nx, ny, nz, &blacks, &whites, &greys);
  ...
  blockout[INDEX(x, y, z, nx, ny, nz)] = newValue;
}
__kernel void hysteresis_secondStep( ... ) {
  ...
  // set voxel to WHITE or BLACK using majority of neighbors
  countNeighborhood(blockin,..., x, y, z, nx, ny, nz, &blacks, &whites, &greys);
  ...
  blockout[INDEX(x, y, z, nx, ny, nz)] = newValue;
}
```



Execution times obtained using two different processing networks: one with modules implemented directly in OpenCL and another with modules implemented using HRTE

- OpenCL kernels are the same in both cases
- HRTE allows an average a speedup of 1.67