A Heterogeneous Runtime Environment for Scientific Desktop Computing

Nuno Oliveira[1, 2], Pedro D. Medeiros[2]

[1]Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal
no@deetc.isel.ipl.pt
[2]NOVA LINCS/Dept. Informática, Universidade Nova de Lisboa, Portugal
pdm@fct.unl.pt

Abstract. Heterogeneous architectures encompassing traditional CPUs with two or more cores, GPUs and other accelerators like the Intel Xeon Phi, are available off the shelf at an affordable cost in a desktop computer. This paper describes work towards the definition, implementation and assessment of an environment that will empower scientists and engineers to develop and run their demanding applications in such personal computers.

We describe HRTE (Heterogeneous Runtime Environment) that allows the construction of dedicated problem solving environments (PSE) taking advantage of those powerful and local processing elements, thus avoiding the use of remote machines through resource managers that introduce large latencies. HRTE is tailored to the communication and execution patterns of a PSE, efficiently mapping them to the heterogeneous architecture described. We also developed an API that eases the development of modules (*HModules*) that support multiple parallel implementations and are easily integrated in a traditional PSE.

HRTE functionality and performance and the API used to build *HModules* are assessed in the construction of a PSE in the area of Materials Science.

Keywords: Heterogeneous architecture, GPU, PSE (Problem Solving Environment), Runtime environment, accelerator, OpenCL

1 Introduction

Scientists have been conducted their research using increasing computational power to run their simulation models, to analyze large experimental data, and to compare observed and predicted data. The exploitation of the parallel hardware that supports the required levels of performance is too complex to one that is not a computer science expert. This complexity of hardware, middleware, software versions and standards must be hidden from the user. The objective is that an expert in a specific science could define his model or simulation without worrying about the runtime environment.

Problem solving environments (PSE) are integrated environments for solving a target class of problems in an application domain. Typically, they encapsulate the state of the art algorithms and problem solving strategies through an easy interface in a

adfa, p. 1, 2011.

way that an expert in the application domain could run his model without specialized knowledge of the underlying computer hardware or software. Several open source frameworks for building PSEs exist, namely OpenDX, Voreen and SCIRun [1].

PSE environment offers the possibility of using building blocks from a library and interconnecting them in a network of modules that supports a dataflow model. The runtime of the PSE toolkit supports the dataflow between modules, the visualization of the intermediate and final results, as well as the modification of some parameters during the execution (steering computation). The network of modules can incorporate domain specific libraries such as numeric computation and visualization.

A PSE provides a diverse set of modules with specific functions and the interface allows the user to build easily a network of modules. The execution of this network by the PSE runtime performs the processing steps needed to achieve the goal of the user. In each moment, the PSE scheduler determines the subset of modules that need to be executed according to data stream dependencies.

The runtime environments of PSEs need to support high requirements of computational power. This computational power is typically supported by cluster machines or even through the grid infrastructure. However, the use of remote parallel processing platforms implies the submission of requests through batch schedulers that introduce intolerable latencies for interactive use. A change in the technologies used for executing PSE modules is necessary in order to achieve a significant reduction of the processing times combined with small latencies that allows an interactive use by users. One promising way to achieve the above stated goal is through the exploitation of the heterogeneous multi-core architectures present in current desktop computers.

Thereby it is possible to develop new modules that take advantage of multiple CPU using frameworks as PThreads or OpenMP. For the same reason the operation of other processing units (PU), such as GPUs, can be carried out by the individual modules using frameworks like NVIDIA CUDA, OpenCL, etc. Therefore there is no obstacle in develop a module to take advantage of this type of hardware. These PUs are also known as accelerators and can share the main memory of the main processor (CPU) or having a private addressing space. In this work we used GPUs with its own separate address space. These types of PUs causes the module to explicitly copy the data into the memory of the PU, submit the code (kernel) to be executed, and finally copy of the data back to the main system memory.

The authors of [2] claim that in many cases the coordinated use of all the PUs of a heterogeneous architecture allows performance gains when a comparison with a homogeneous solution is performed. The programmer could implement modules targeting the most suitable hardware in mind, using a specific programming model and/or specific programming libraries.

To deal with the diversity of modules used for a given goal, we propose *HRTE* (*Heterogeneous Runtime Environment*) to support the execution of PSE tasks over the heterogeneous hardware available on a single desktop computer. HRTE has two main parts that correspond to the two main contributes of this work:

- *Simplifying the development of new modules.* HRTE offers the notion of Heterogeneous Module (*HModule*) supporting several implementations for each type of PU allowing it to run on multiple hardware architectures. Support of transparent man-

agement of data copy between main memory and memory of the PUs is also included. The development of *HModules* is simplified through the availability of methods that implements *map* and *stencil* parallel control patterns [3] over HRTE.

- *Optimizing the execution of the of module network:* HRTE supports efficient access to large volumes of data flowing between modules in a complex memory hierarchy (including multi-core CPUs, GPUs and other kinds of PUs). This data flow optimization between *HModules* is achieved through the minimization of the number of data transfers between CPU and PUs memories, taking advantage of the current location of the data.

Related work. Several research efforts that allow the exploitation of heterogeneous architectures for building efficient applications have been successful: OpenCL [4], HSA [5], StarPU [2], Harmony [6], and PTask [7]. Regarding the convergence of such efforts and PSE toolkits most of the projects have targeted clusters and grids [8,9]. Several references exist regarding the use of GPU-enabled modules in PSEs [10]. Parallel structured programming projects like FastFlow [11] address both heterogeneous architectures and support of dataflow between components (pipeline pattern).

Paper organization. This paper is organized as follows. We begin by describing the characteristics and organization of HRTE in Section 2, followed by the presentation of some relevant aspects of the current implementation of HRTE using SCIRun and StarPU in Section 3. In section 4 we present a case study, namely the application of HRTE in the implementation of a PSE in the area of Materials Science. Finally we present the conclusions and current work in Section 5.

2 HRTE Organization

A PSE toolkit provides modules that can be interconnected with other modules in a dataflow approach. Each module reads its data from inputs, executes an algorithm and generates its outputs to be sent to other modules. HRTE introduces a new type of module (*HModule)*. These modules allow the execution of an algorithm in several platforms (hardware and software). These extensions should maintain compatibility with original features of PSE. Therefore all existing modules can still be used and can be interconnected with the new *HModules* (see Fig. 1).

In most PSEs large volumes of data are transferred between modules. The efficient support of these huge data transfers and the optimization of its sharing between modules must be tailored to an environment where a hierarchy of levels of memory exists; if we consider that some of the modules will be offloaded to an accelerator this implies that the data must also be transferred to and from the accelerator's memory. The transfer costs must be considered by the runtime environment, otherwise the gains of using the accelerator can be hidden by the overheads intrinsic to data transfer between separate components of the memory hierarchy. Another issue is related with the limited memory in some accelerators, imposing that the accelerator's memory may not accommodate all the data thus implying its partition. Therefore, HRTE must also

extend the PSE's dataflow between *HModules* in order to send additional information about the locality and partition mode of the data transferred.

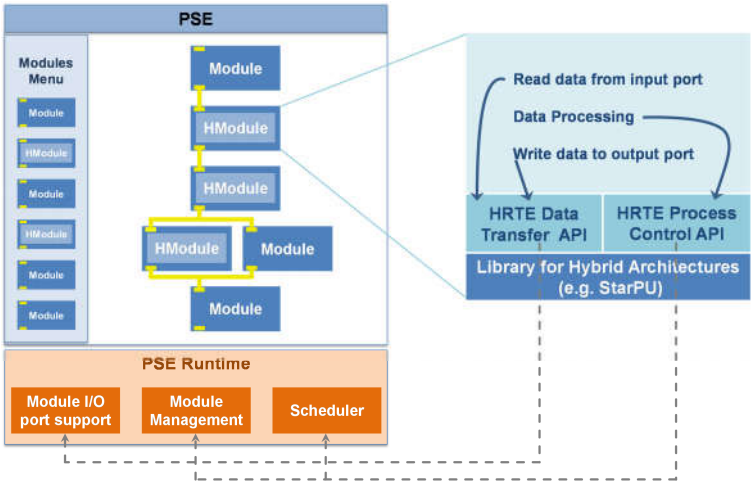

Fig. 1. A PSE environment with standard modules and the new *HModule* in same application

To be able to incorporate *HModules* in an existing PSE framework one must modify the PSE code to handle the execution of new *HModules* and the dataflow between both types of modules. The HRTE organization allows the minimization of changes of PSE code thus easing the integration of HRTE in different PSE toolkit. These modifications allow the definition of a new *HModule* by defining the following methods:

`void getInputs()`	Extract data from input ports and register it.
`void setOutputs()`	Generate the data to the output ports of the module.
`void hexecute()`	Definition of actions performed by the *HModule*.

A dynamic library supports all the functionalities of the runtime and is used by the *HModule* code. It supports the concept of a heterogeneous function allowing, in the same module, the availability of different implementations. Next, we give an example of adding an OpenCL implementation to a *HModule*:

`void hrte_HFunction_add_opencl_code( hrte_HFunction *hf,char *kernelName,char *clFile);`	Register an OpenCL kernel implementation indicating the filename containing the OpenCL code.

The library also supports data management allowing data registration, data partition (with and without ghost zones). The registration of the data in HRTE is done using the functions below:

`void hrte_matrix3d_register (hrte_data_handle *handle,void *ptr,uint32_t nx,uint32_t ny,uint32_t nz,size_t elemsize);`	Register a 3D matrix.
`void hrte_matrix3d_set_partitions (hrte_data_handle handle,int n);`	Set number of partitions on data.

To simplify the definition of a *HModule* **map** and the **stencil** parallel control patterns [3] are available as presented here:

`void hrte_task_map(hrte_HFunction *hf, hrte_data_handle in,hrte_data_handle out,hrte_HFunctionArgs *ha);`	Map pattern will apply the heterogeneous function to every element of the input data.
`void hrte_task_stencil(hrte_HFunction *hf, hrte_data_handle in,hrte_data_handle out);`	Stencil pattern will apply the heterogeneous function to every element and its neighbors.

3 Current HRTE Prototype

At present our prototype has been developed using SCIRun [1] as the PSE framework. As described in previous section we need to extend the SCIRun Module and the dataflow between modules to integrated HRTE and augmented SCIRun to support *HModules*. The definition of a new module in SCIRun implies the definition of a new C++ class extending from the Module class and the definition of the virtual method execute that is called when the module is executed. The optional graphical user interface associated with the module is defined in TCL script language and finally the specification of the input and output ports are made in a XML file. The dataflow that interconnect modules was extended to include additional information when we have *HModules* interconnected.

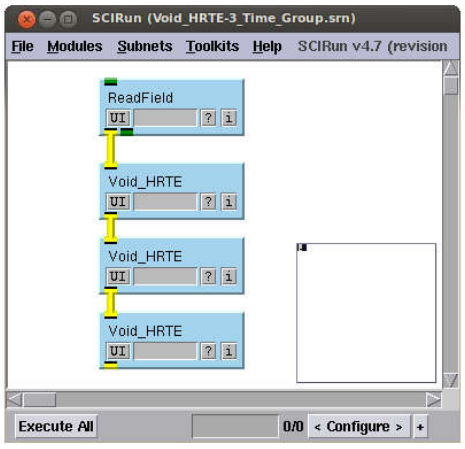

Fig. 2. Network used to evaluate the performance of dataflow between modules

image size	Partitions	OpenCL	HRTE
100	1	29.3	23.6
	2	31.0	25.0
	4	27.5	26.1
200	1	55.2	31.6
	2	56.2	34.5
	4	67.7	31.4
300	1	200.9	53.3
	2	176.8	60.2
	4	205.6	62.4
400	1	545.3	188.3
	2	554.0	184.6
	4	556.3	188.5
500	1	1009.9	339.1
	2	1018.8	340.6
	4	1023.0	342.5
600	1	1772.7	575.6
	2	1768.8	574.1
	4	1785.8	578.1
700	1	2714.6	891.6
	2	2636.8	894.5
	4	2734.4	894.6

Fig. 3. Comparison of total execution times with and without HRTE support

All the HRTE runtime described in section 2 is supported by StarPU [2]. *HModule*s are mapped to tasks and codelets; module input and output uses StarPU block management interface. The modifications made to the PSE dataflow part and the use of StarPU allows significant performance improvements when executing a sequence of *HModules*. This claim had been validated through the use of a network of *HModules*

that runs an OpenCL kernel (Fig. 2) that only outputs the data received without any processing. The evaluation used a machine with an Intel Xeon CPU E5506 at 2.13GHz, 12 GB of RAM and two NVIDIA Tesla C1060. The operating system is Ubuntu 12.04 x86_64. The GPUs driver is the NVIDIA 340.29. The GPU SDK is CUDA 6.5.14 (OpenCL 1.1). PSE is SCIRun 4.7 and StarPU is 1.2.0rc2.

In Fig. 3 we compare the execution times of this network with a similar one without HRTE (same OpenCL kernel). Optimization of dataflow between HModules allows a reduction in execution time up to 33% over the version that does not use HRTE.

4 A Case Study in Materials Science

In the field of Materials Science, research on composite materials (comprising two distinct materials, where one constitutes a base matrix and the other acts as reinforcement) has a growing relevance in transportation and energy areas [12].

To forecast the characteristics of a new material, it is vital to characterize the reinforcement's population regarding aspects, such as position, size and orientation of the particles. X-ray computed tomography (X-ray CT) images are used for the characterization activities. The task of processing and analysing such data is a complex one: not only there is a huge volume of data to be processed but also there are noise and artefacts that must be removed; low contrast between the matrix and the reinforcement particles, due to small density difference makes this processing computing intensive.

Support of this processing and its easy handling by a non IT specialist requires an environment that allows the definition of a sequence of computational processing steps as well as its parameterization values in an interactive and real time way. In this setting, the construction of a PSE to the characterization of reinforcement population in 3D tomographic data is an opportunity for assessing the functionality and performance of HRTE. The images obtained by CT need processing to eliminate noise and allow the detection of boundaries between the base material and the reinforcement particles.

We developed three *HModules* to process the tomographic 3D image. The modules perform in sequence bi-segmentation, hysteresis and ImageLabeling operations. Bi-segmentation transforms the CT 3D original greyscale image to an image with only three colors: black, grey and white. The base material is represented as white, the reinforcements objects as black and the grey color represents voxels that due to the low contrast of the image aren´t yet classified as belonging to the base material or to the reinforcements. The main goal of hysteresis is to eliminate the grey voxels. The hysteresis is implemented following the majority color of the neighbor's voxels. The ImageLabeling segments the image labeling each particle with a unique identifier. This Labeling allows the characterization of each reinforcement object.

In **Fig. 4** we present a simplified declaration of the Segmentation *HModule* including the virtual method *hexecute*. The method begins by reading the tomographic image from the input port of the module. After reading, the map parallel pattern is used to apply the OpenCL kernel to all the voxels of the 3D image. After, the result image is sent to the output port of the module.

```
class Segmentation:public HModule{
void hexecute() {
   hrte_data_handle img;
   get_input_hrte_handle(…,img,…);
...
   hrte_task_map(…,img,img,…);
   send_output_hrte_handle(…,img);
 }
```

```
__kernel void Segmentation (…){
  BYTE value;
  value=bk[INDEX(x,y,z,nx,ny)];
  if(value<=min) value=BLACK;
  else if(value>max) value= WHITE;
  else value=GREY;
  bkO[INDEX(x,y,z,nx,ny)]=value;
}
```

Fig. 4. Definition of SCIRun Segmentation *HModule* and the OpenCL kernel used

Fig. 5 presents the PSE network used in the experiment; Fig. 6 contains a table comparing the execution times of two networks built using the same approach described in section 3: the 3rd column corresponds to a network where modules don't use HRTE support while the 4th column shows the execution time for the same kernels wrapped as *HModules*.

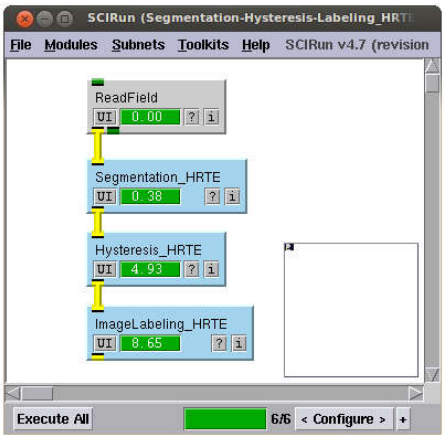

Fig. 5. Network to process CT raw images allowing the adequate objects identification

image size	Partitions	OpenCL	HRTE
100	1	59.6	57.2
	2	64.8	69.4
	4	66.3	70.1
200	1	218.6	175.7
	2	226.0	172.0
	4	228.1	180.9
300	1	709.9	512.9
	2	701.0	527.1
	4	725.1	532.1
400	1	1717.2	1193.4
	2	1724.4	1226.1
	4	1732.6	1240.8
500	1	3266.7	2323.6
	2	3274.5	2383.9
	4	3261.2	2403.9
600	1	5986.5	4297.8
	2	5984.0	4381.0
	4	6018.9	4424.7
700	1	10195.3	8255.1
	2	10209.3	8384.7
	4	10182.3	8466.8

Fig. 6. Evaluations times of the network executed with and without HRTE support

As presented, the version of the visual program that uses HRTE reduces the execution time up to 70% to that one which uses directly OpenCL.

5 Conclusions and future work

In this paper we presented HRTE which aims to ease the development of applications that use parallelism to tackle computational problems characterized by big needs in computational power and processing of big volumes of data. Our target is to help scientists and engineers that are not parallel processing specialists to develop modules for Problem Solving Environments toolkits that make an efficient exploitation of desktop computers equipped with accelerators.

As far as we know, this effort to create a framework allowing the integration in Problem Solving Environments toolkits of modules that can have different implementations and communicate efficiently by optimizing the data transfers is original. The results obtained in our prototype using SCIRun and StarPU, assessed through a realistic 3D image processing are promising in terms of performance and also regarding the ease of development of new modules. The experiments described gave us valuable insights to further developments of our research efforts.

Acknowledgements. FCT MCTES and NOVA LINCS UID/CEC/04516/2013. The Polytechnic Institute of Lisbon (IPL) supports the 1st author as a doctoral student.

References

1. Parker, S.G., Johnson, C.R.: SCIRun: A Scientific Programming Environment for Computational Steering. In: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing. Supercomputing '95, New York, NY, USA, ACM (1995)
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. Concurr. Comput. : Pract. Exper. **23**(2) (February 2011) 187–198
3. McCool, M., Reinders, J., Robison, A.: Structured Parallel Programming: Patterns for Efficient Computation. 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2012)
4. Khronos: OpenCL. https://www.khronos.org/opencl/ (2016)
5. Hwu, W.M.W., ed.: Heterogeneous System Architecture: A New Compute Platform Infrastructure. Morgan Kaufmann Publishers Inc. (2016)
6. Diamos, G., Yalamanchili, S.: Harmony: An Execution Model and Runtime for Heterogeneous Many Core Systems. In: HPDC'08, Boston, Massachusetts, USA, ACM (June 2008)
7. Rossbach, C.J., Currey, J., Silberstein, M., Ray, B., Witchel, E.: PTask: Operating System Abstractions to Manage GPUs as Compute Devices. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11, New York, NY, USA, ACM (2011) 233–248
8. Peterson, J., Hallock, M., Cole, J., Luthey-Schulten, Z.: A problem solving environment for stochastic biological simulations. In: Proceedings of the 3rd Workshop on Python for High-Performance and Scientific Computing. (2013)
9. Miller, M., M.C.D.J..J.C.: Grid-Enabling Problem Solving Environments: a Case Study of SCIRun and NetSolve. In: Proceedings of HPC 2001. (April 22-26 2001) 98–103
10. Leeser, M., Yablonski, D., Brooks, D., King, L.S.: The Challenges of Writing Portable, Correct and High Performance Libraries for GPUs. SIGARCH Comput. Archit. News **39**(4) (December 2011) 2–7
11. Aldinucci, M., Danelutto, M., Kilpatrick, P., Meneghin, M., Torquati, M.: Accelerating Code on Multi-Cores with FastFlow. In: Proceedings of the 17th international conference on Parallel processing - Volume Part II. Euro-Par'11, Berlin, Heidelberg, Springer-Verlag (2011) 170–181
12. Cadavez, T., Ferreira, S.C., Medeiros, P., Quaresma, P.J., Rocha, L.A., Velhinho, A., Vignoles, G.: A Graphical Tool for the Tomographic Characterization of Microstructural Features on Metal Matrix Composites. International Journal of Tomography & Statistics **14**(S10) (2010) 3–15