

# A Unified Programming Model for Heterogeneous Computing with CPU and Accelerator Technologies\*

Yuqing Xiong

(Computer Science Department, Shanghai Institute of Technology, Shanghai, China)  
yqxiong@sit.edu.cn

## Abstract

This paper consists of three parts. The first part provides a unified programming model for heterogeneous computing with CPU and accelerator (like GPU, FPGA, Google TPU, Atos QPU, and more) technologies. To some extent, this new programming model makes programming across CPUs and accelerators turn into usual programming tasks with common programming languages, relieves complexity of programming across CPUs and accelerators, and causes programs modularity higher. It can be achieved by extending file managements in common programming languages, such as C/C++, Fortran, Python, MPI, etc., to cover accelerators as I/O devices. In the second part, we show that all types of computer systems can be reduced to the simplest type of computer system, a single-core CPU computer system with I/O devices, by the unified programming model. Thereby, the unified programming model can truly build the programming of various computer systems on one API (i.e. file managements of common programming languages), and can make programming for various computer systems easier. In the third part, we present a new approach to coupled applications computing (like multidisciplinary simulations) by the unified programming model. The unified programming model makes coupled applications computing more natural and simpler since it only relies on its own power to couple multiple applications through MPI.

**Keywords:** unified programming model, CPU, accelerator, general printer, one API, coupled applications computing

## 1 Introduction

Heterogeneous computing with CPU and accelerator technologies is widely concerned. However, there are some challenges in the heterogeneous computing. To remove the difficulties, for example, an architecture for unified deep learning with CPU, GPU, and FPGA technologies is presented[1]. The examples of underlying hardware approaches in the architecture are shown in Figure 1 and Figure 2 (extracted from [1]). This is a typical heterogeneous computing architecture with CPU and accelerator technologies.

A key problem for the heterogeneous computing is that a full and seamless programming environment that works across CPUs and accelerators is necessary so that the architecture can work well [1]. However, it seems to difficult to design and build the full and seamless programming environment

---

\*The first draft of the first part in this paper was written in June 2018. To the best of the author's knowledge, the author is the first to propose the idea of the unified programming model for Heterogeneous Computing with CPU and Accelerator Technologies.

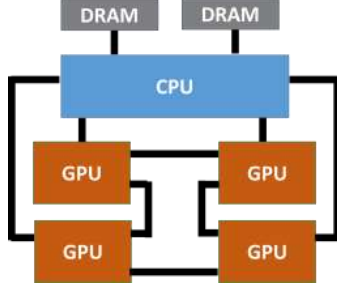


Figure 1: GPU Scale out using Vega20 and Epyc  
(extracted from [1])

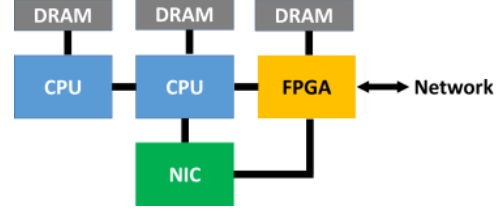


Figure 2: FPGA Exploitation  
(extracted from [1])

since it is not easy that communication between application programs of common programming languages for CPUs and programs of programming languages for accelerators are carried out. For example, data movement between MPI for distributed memory parallel programming and CUDA is difficult, it makes programming with MPI+CUDA complicated [2].

To overcome the challenge of communication between CPUs and accelerators in the common programming language world, this paper tries to provide a unified programming model that can work across CPUs, GPUs, FPGAs, and other accelerators (such as Google TPU, Atos QPU, etc.) by extending file managements in common programming languages, such as C/C++, Fortran, Python, MPI, etc., to cover GPUs, FPGAs, and other accelerators (such as Google TPU, Atos QPU, etc.) as I/O devices. To some extent it makes programming across CPUs and accelerators turn into usual programming tasks, and relieve complexity of programming across CPUs, GPUs, FPGAs, and other accelerators since it makes heterogeneous systems turn into homogeneous systems from a certain perspective (accelerators as one kind of I/O devices). Thus the programming model can contribute to improve software productivity for computing across CPUs and accelerators.

To further explain the unified programming model to simplify programming complexity, in this paper, we will show that all types of computer systems can be reduced to the simplest type of computer system, a single-core CPU computer system with I/O devices, by the unified programming model. Thereby, the unified programming model can truly build the programming of various computer systems on one API (i.e. file managements of common programming languages), and can make programming for various computer systems easier.

Coupled applications computing (like multidisciplinary simulations) is very important in many fields. It usually needs additional software to support [3]. In this paper, we will present a new approach to coupled applications computing, which is based on the unified programming model. The unified programming model makes coupled applications more natural and easier since it only relies on its own power to couple multiple applications through MPI.

## 2 A Unified Programming Model for Heterogeneous Computing with CPU and Accelerator Technologies

### 2.1 Accelerators as One Kind of I/O Devices in Common Programming Languages

File managements in common programming languages are a kind of mechanisms which make application programs access I/O devices easier. An application's access to I/O devices generally involves many tasks carried out by systems, that is, the tasks are accomplished by systems (not by applications) in the name of application access to I/O devices through file managements. This name is the key to making it easy for applications (running on CPUs) to access I/O devices.

Therefore, if we can also regard accelerators as one kind of I/O devices (or one kind of general printers exactly) in common programming languages (although accelerators, such as GPUs, aren't any kind of I/O devices in the usual sense) and extend file managements in common programming languages to cover accelerators as one kind of I/O devices, the data movement between CPUs and accelerators can also be carried out by systems in the name of application access to I/O devices through the file managements. Thus we can explicitly avoid data movement between CPUs and accelerators in application programs of the common programming languages and make programming across CPUs and accelerators easier. GPUs are taken as an example to illustrate it here.

Let's compare GPUs with HP printers. There are many models (such as LaserJet P1008) for the HP printers, there is a driver for each model, and outputs of printers are to papers. Similarly, "GPU+CUDA code running on the GPU" can be regarded as an I/O device (a general printer), its model is the CUDA code, its driver is also the CUDA code, and its output is to CPUs (application processes more exactly).

Let's take an example to illustrate it further, see Figure 3. Assume that P is a program written in a common programming language and runs on a CPU, Q is a program written in CUDA and runs on a GPU. "The GPU+Q" is regarded as a general printer, the code of Q is its model, and Q is its deriver. Now P invokes Q, so P sends data to Q (i.e. P sends input to the general printer to print), and Q receives the data, processes the data, and then sends the computational result to P (i.e. the general printer prints the results to P). Thus communication between P and Q is carried out, and P completes the invoking to Q. Figure 3 shows the communication between P and Q.

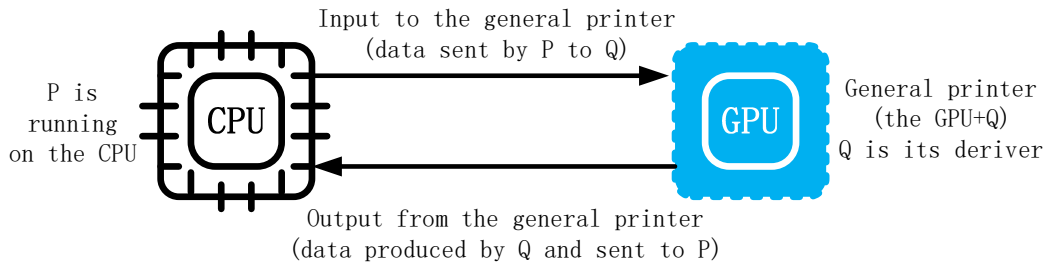


Figure 3: Communication between P running on a CPU and Q running on a GPU.

Assume that `print()` is a function in the extended file management of the common programming language P is written in,  $D_{input}$  is the data sent by P to Q, and the computational result sent to

P by Q is “printed” on  $D_{output}$ . The above description of P invoking Q can be expressed by the function as follows:

$$D_{output} = \text{print}(\text{the GPU} + Q, D_{input})$$

This way, P invoking Q through `print()` triggers the system to carry out the communication between P in the CPU and Q in GPU implicitly, that is, we realize data movement between application programs of common programming languages on CPUs and programs of CUDA on GPUs in the name of printing the data by the general printer.

In operating systems and common programming languages, such as C/C++, Fortran, Python, etc., all I/O devices (including printers) are regarded as files, so it should be natural that “GPU+CUDA code running on the GPU” can also be regarded as files in common programming languages.

Naturally, we should extend file managements in common programming languages, such as C/C++, Fortran, Python, etc., to include accelerators as one kind of I/O devices. When the common programming language is MPI, MPI-I/O should be extended to cover accelerators as one kind of I/O devices, and correspondingly the file management in C should be extended to include accelerators as one kind of I/O devices since MPI is implemented in C.

Thus, we simplify programming across CPUs, GPUs, FPGAs, and other accelerators into usual programming tasks with common programming languages to some extent, and make programming in heterogeneous systems with accelerators easier since it makes heterogeneous systems turn into homogeneous systems from a certain perspective (accelerators as one kind of I/O devices). So this can contribute to improve software productivity for computing across CPUs and accelerators.

By this point, we have established a unified programming model for heterogeneous computing with CPU and accelerator technologies. For convenience, we use UPM as an abbreviation for “the Unified Programming Model for heterogeneous computing with CPU and accelerator technologies”.

## 2.2 For Multi-core CPUs for OpenMP or Pthread in UPM

In UPM, for multi-core CPUs where programs based on OpenMP or Pthread run, we can imagine that every multi-core CPU consists of one “single-core CPU” for programs of common programming languages and one “multi-core accelerator” for programs based on OpenMP or Pthread\*\*. In the same way, we can regard the “multi-core accelerators + program codes based on OpenMP/Pthread running on the multi-core accelerators” as I/O devices, and extend file managements in common programming languages to include them as one kind of I/O devices (i.e. files). Thus to some extent we also simplify programming model “common programming languages + OpenMP/Pthread” into usual programming with common programming languages.

Thus, UPM allows OpenMP/Pthread-based code to be separated from the entire program, resulting in higher modularity.

---

\*\* After the first part of this paper appeared (i.e., after UPM was established), Mr. James Reinders at Intel thought that a multi-core CPU could be regarded as an accelerator. Inspired by it, the author thought a multi-core CPU should be imagined as a single-core CPU + a multi-core “accelerator” in UPM. The author would like to thank Mr. Reinders although he didn’t mention UPM. The author think that it should make little sense without UPM that a multi-core CPU can be regarded as an accelerator.

### 2.3 For Clusters of CPUs for MPI in UPM

In UPM, for clusters of CPUs where programs based on MPI run, we imagine that every cluster of CPUs is composed of one CPU for programs of common programming languages and one “cluster-of-CPU accelerator” for programs based on MPI. We use similar ideas as in section 2.2 and regard the “cluster-of-CPU accelerator + program codes based on MPI running on the cluster-of-CPU accelerator” as an I/O device, and extend file managements in common programming languages to include it as one kind of I/O devices (i.e. files).

According to this, UPM may make coupled applications computing (like multidisciplinary simulations) more natural and easier, see section 4 below.

## 3 Computer Systems under the View of UPM

So far, there are many kinds of computer systems in the world. We roughly divide these computer systems into several types according to whether CPUs are single-core and whether there is a cluster of CPUs and whether there are accelerators. Although some computer systems may not belong to any type, this does not affect the results of our discussion in this paper. Assume that all the computer systems are attached with I/O devices. We give a description of every type of computer system as follows.

Type	Description
I	a single-core CPU
I <sup>+</sup>	a single-core CPU with accelerators
II	a multi-core CPU
II <sup>+</sup>	a multi-core CPU with accelerators
III	a cluster of single-core CPUs
III <sup>+</sup>	a cluster of single-core CPUs with accelerators
IV	a cluster of mutil-core CPUs
IV <sup>+</sup>	a cluster of multi-core CPUs with accelerators

For convenience, we use  $\implies$  as “can be reduced to”.  $A \xRightarrow{UPM} B$  represents that A can be reduced to B by UPM.

### 3.1 All Types of Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

In this section, we show that every type of computer system can be reduced to a type-I computer system by UPM, i.e., any type of computer system  $\xRightarrow{UPM}$  a type-I computer system.

#### 3.1.1 Type-I Computer Systems

A type-I computer system contains only a single-core CPU and is the simplest computer system. Its programming model is common programming languages, such as C/C++, Fortran, Python, Java and so on, and is the simplest programming method.

#### 3.1.2 Type-I<sup>+</sup> Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

A type-I<sup>+</sup> computer system is composed of type-I computer system and accelerators. According to section 2.1, the accelerators in the type-I<sup>+</sup> computer system can be regarded as I/O devices (general

printers exactly), and then as files by UPM. Thus, the accelerators in the type-I<sup>+</sup> computer system “disappear”, and the type-I<sup>+</sup> computer system turns into a type-I computer system, i.e., the type-I<sup>+</sup> computer system  $\xRightarrow{UPM}$  a type-I computer system.

### 3.1.3 Type-II Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

In a type-II computer system, the CPU is a multi-core CPU. According to section 2.2, the type-II computer system is composed of a type-I computer system and a “multi-core accelerator”. The “multi-core accelerator” can be regarded as an I/O device (a general printer exactly), and then a file by UPM. Thus, the “multi-core accelerator” also “disappears”, and the type-II computer system turns into a type-I computer system, i.e., the type-II computer system  $\xRightarrow{UPM}$  a type-I computer system.

### 3.1.4 Type-II<sup>+</sup> Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

A type-II<sup>+</sup> computer system is composed of a type-II computer system and accelerators. According to the same reason as in section 3.1.2, the type-II<sup>+</sup> computer system  $\xRightarrow{UPM}$  a type-II computer system. By section 3.1.3, the type-II computer system  $\xRightarrow{UPM}$  a type-I computer system. Therefore, the type-II<sup>+</sup> computer system  $\xRightarrow{UPM}$  the type-I computer system.

### 3.1.5 Type-III Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

There is a cluster of single-core CPUs in a type-III computer system, and programs based on MPI are for it. According to section 2.3, the type-III computer system is composed of a single-core CPU and a “cluster-of-single-core-CPU accelerator”. By UPM, the “cluster-of-single-core-CPU accelerator” can be regarded as an I/O device (a general printer exactly), and then a file. Thus, the “cluster-of-single-core-CPU accelerator” also “disappears”, and the type-III computer system turns into a type-I computer system, i.e., the type-III computer system  $\xRightarrow{UPM}$  a type-I computer system.

### 3.1.6 Type-III<sup>+</sup> Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

A type-III<sup>+</sup> computer system is composed of a type-III computer system and accelerators. According to the same reason as in section 3.1.2, the type-III<sup>+</sup> computer system  $\xRightarrow{UPM}$  a type-III computer system. By section 3.1.5, the type-III computer system  $\xRightarrow{UPM}$  a type-I computer system. Therefore, the type-III<sup>+</sup> computer system  $\xRightarrow{UPM}$  the type-I computer system.

### 3.1.7 Type-IV Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

In a type-IV computer system, the CPUs are multi-core CPUs. According to the same reason as in section 3.1.3, the type-IV computer system  $\xRightarrow{UPM}$  type-III computer system. By section 3.1.5, the type-III computer system  $\xRightarrow{UPM}$  a type-I computer system. Therefore, the type-IV computer system  $\xRightarrow{UPM}$  the type-I computer system.

### 3.1.8 Type-IV<sup>+</sup> Computer Systems $\xRightarrow{UPM}$ Type-I Computer Systems

A type-IV<sup>+</sup> computer system is composed of a type-IV computer system and accelerators. According to the same reason as in section 3.1.2, The type-IV<sup>+</sup> computer system  $\xRightarrow{UPM}$  a type-IV computer

system. By section 3.1.7, the type-IV computer system  $\xRightarrow{UPM}$  a type-I computer system. Therefore, the type-IV<sup>+</sup> computer system  $\xRightarrow{UPM}$  the type-I computer system.

### 3.2 Programming of Various Computer Systems Is Built on One API by UPM

The above analysis shows that every type of computer system, from type-I<sup>+</sup> computer systems to type-IV<sup>+</sup> computer systems, can be reduced to a type-I computer system, a simplest computer system containing only a single-core CPU, by UPM, and so all types of computer systems are unified into one by UPM in this sense, and so UPM can truly build the programming of various computer systems on one API (i.e. extended file managements of common programming languages for type-I computer systems). The programming of the type-I computer system is the simplest, therefore, UPM can make the programming for various computer systems easier.

## 4 Coupled Applications Computing by UPM

Today, coupled applications computing (like multidisciplinary simulations) plays a key role in many fields. However, It usually requires additional software to provide support to couple multiple applications [3]. This makes coupled applications computing complicated. Here, we present a more natural and simpler approach to coupled applications computing, which is based on UPM.

Assume that a coupled applications computing consists of  $n$  applications  $A_1, A_2, \dots, A_n$ , which are based on MPI, and runs on  $n$  clusters cluster<sub>1</sub>, cluster<sub>2</sub>, ..., cluster <sub>$n$</sub> , respectively. See Figure 4.

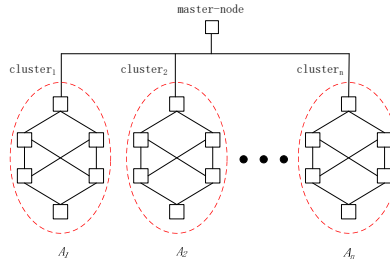


Figure 4: A coupled applications computing.

According to section 2.3, in UPM, cluster <sub>$i$</sub>  (where  $1 \leq i \leq n$ ) can be imagined to be composed of a node (one CPU) for programs of common programming languages and one “cluster-of-CPU accelerator” for programs based on MPI. We can regard the “cluster-of-CPU accelerator (cluster <sub>$i$</sub>  +  $A_i$ )” as a general printer (general-printer <sub>$i$</sub> ), and extend file managements in common programming languages and MPI-I/O to include it as one kind of I/O devices (i.e. files), see Figure 5.

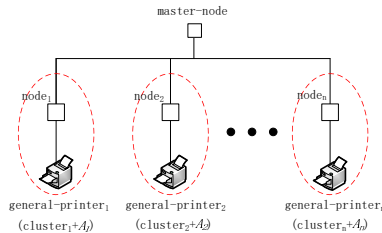


Figure 5: The clusters as general printers in the coupled applications computing by UPM.

Now we form a new cluster consisting of master-node, node<sub>1</sub>, node<sub>2</sub>, ..., node<sub>n</sub>, as shown in the blue circle in Figure 6. We call the cluster cluster<sub>super</sub>. Assume that  $A_{super}$ , a program based on MPI, runs on the cluster<sub>super</sub>. The  $A_{super}$  is responsible for coordinating the  $n$  applications  $A_1, A_2, \dots, A_n$ .

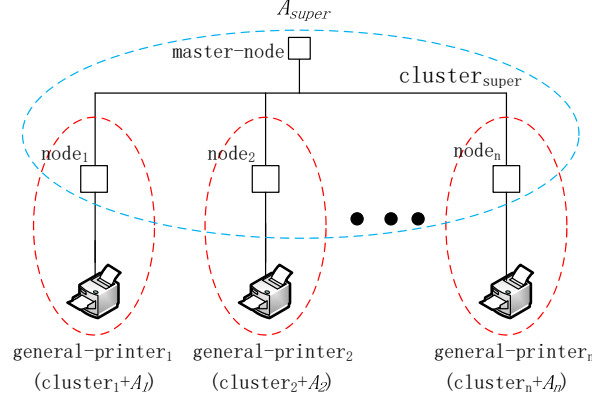


Figure 6: cluster<sub>super</sub> consisting of master-node, node<sub>1</sub>, node<sub>2</sub>, ..., node<sub>n</sub>.

When  $A_i$  sends data to  $A_j$  (where  $1 \leq i, j \leq n$ ), the communication is carried out as follows:

**Step 1**  $A_{super}$  gets the data in node <sub>$i$</sub>  through MPI-I/O from general-printer <sub>$i$</sub>  ( $A_i$  exactly);

**Step 2**  $A_{super}$  sends the data from node <sub>$i$</sub>  to node <sub>$j$</sub>  through MPI over the cluster<sub>super</sub>;

**Step 3**  $A_{super}$  sends the data in node <sub>$j$</sub>  to general-printer <sub>$j$</sub>  ( $A_j$  exactly) through MPI-I/O.

Usually  $A_{super}$  can not communicate with  $A_i$  (where  $1 \leq i \leq n$ ) since they are in different parallel MPI worlds. But in UPM,  $A_i$  is a driver of a I/O device (a general printer), and its MPI world is subordinate to (through MPI-I/O), rather than parallel to, the MPI world of  $A_{super}$ .

The UPM makes coupled applications computing more natural and easier since it only relies on its own power to couple multiple applications through MPI.

## 5 Scope of Application of UPM

Architectures composed of CPUs and accelerators are increasingly popular in computing systems. Today many computing systems, ranging from embedded systems, to mobile computing systems, to HPC systems, to cloud computing systems, and to quantum-classical hybrid computing systems are equipped with accelerators. UPM can be applied to all these systems since CPU and accelerator technologies are used in the computing systems.

## 6 Conclusions and Future Work

The key idea of UPM is that an accelerator and a program executing on it can be regarded as a general printer. We can get the following three results from this:

- This makes any computer system a single-core CPU computer system, which makes programming of any computer systems simpler.



- This allows OpenMP/Pthread-based code to be separated from the entire program code, resulting in higher modularity.
- This can provide another new approach to coupled applications computing.

The above means that UPM is similar to an axiom system, so it is possible that UPM may provide more new solutions to other problems.

## Acknowledgment

The author would like to thank Dr. Pavan Balaji at Facebook AI. The author was invited by him to visit Mathematics and Computer Science Division at Argonne National Laboratory in 2013, and the idea that GPUs are regarded as one kind of I/O devices was formed preliminarily at that time.

## References

- [1] Allen Rush, Ashish Sirasao, and Mike Ignatowski. Unified Deep Learning with CPU, GPU, and FPGA Technologies. White paper, AMD and Xilinx, 2017
- [2] William Gropp and Marc Snir. Programming for exascale computers. *Computing in Science and Engineering*.15, 6(2013), 27-35. 2013
- [3] MpCCI, <https://www.mpcci.de/>