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

Yuqing Xiong

April 10, 2022

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

YuqingXiong
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, 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, and relieves complexity of programming across CPUs and accelerators. 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 third part, we present a new approach to coupled applications (like multidisciplinary simulations) computing by the unified programming model. The unified programming model makes coupled applications computing more natural and easier since it only relies on its own power to couple multiple applications through MPI.**

***Index Terms*—unified programming model, CPU, accelerator, general printer, one API, coupled applications**

I. 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 Fig. 1 and Fig. 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 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, in this paper, we will try to provide a unified programming model that can work across CPUs, GPUs, FPGAs, and other accelerators (such as Google TPU, 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, etc.) as I/O devices. To some extent it makes programming across CPUs and accelerators turn into usual programming tasks, and relieves 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 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

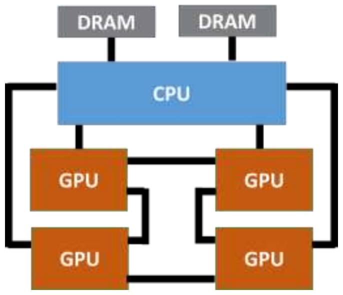

Fig. 1. GPU Scale out using Vega20 and Epyc (extracted from [1]).

The first draft of the first part in this paper was written in June 2018.

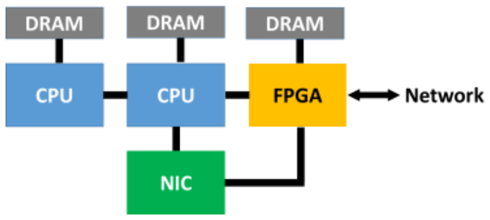

Fig. 2. FPGA Exploitation (extracted from [1]).

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 (like multidisciplinary simulations) computing is very important in many fields. It usually needs addtional software to support. In this paper, we 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 since it only relies on its own power to couple multiple applications through MPI.

II. A Unified Programming Model for Heterogeneous Computing with CPU and Accelerator Technologies

A. 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) 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 exactly).

Let's take an example to illustrate it further, see Fig. 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. 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. Fig. 3 shows the communication between P and Q.

This way, 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 through 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" (a general printer) 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. For MPI, MPI-I/O should be extended to cover accelerators as one kind of I/O devices.

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.

B. For Multi-core CPUs for OpenMP or Pthread

For multi-core CPUs where programs of 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 of OpenMP or Pthread. In the same way, we can regard the "multi-core accelerators + OpenMP/Pthread codes 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

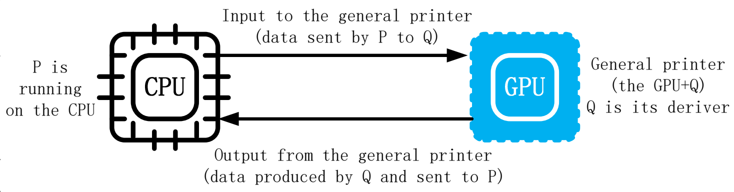
Input to the general printer
(data sent by P to Q)
P is
running
on the CPU
CPU
GPU
General printer
(the GPU+Q)
Q is its deriver
Output from the general printer
(data produced by Q and sent to P)

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

to some extent we also simplify programming model "common programming languages + OpenMP/Pthread" into usual programming with common programming languages.

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

C. For Clusters of CPUs for MPI

For clusters of CPUs where programs of 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 the above and regard the "cluster-of-CPU accelerator + codes of program 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). Thus, to some extent we can simplify programming model "common programming languages + MPI" into usual programming with common programming languages.

III. Computer Systems under the View of the Unified Programming Model

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^+	a single-core CPU with accelerators
II	a multi-core CPU
II^+	a multi-core CPU with accelerators
III	a cluster of single-core CPUs
III^+	a cluster of single-core CPUs with accelerators
IV	a cluster of mutil-core CPUs
IV^+	a cluster of multi-core CPUs with accelerators

For convenience, we use UPM as an abbreviation for "the Unified Programming Model for heterogeneous computing with CPU and accelerator technologies", and $\Longrightarrow$ as "can be reduced to". A $\overset{UPM}{\Longrightarrow}$ B represents that A can be reduced to B by UPM.

A. All Types of Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems

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

1) I-Type Computer Systems: A I-type 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.

2) I^+-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: A I^+-type computer system is composed of I-type computer system and accelerators. According to section II, the accelerators in the I^+-type computer system can be regarded as I/O devices (general printers exactly), and then as files by UMP. Thus, the accelerators in the I^+-type computer system "disappear", and the I^+-type computer system turns into a I-type computer system, i.e., the I^+-type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system.

3) II-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: In a II-type computer system, the CPU is a multi-core CPU. According to section II, the II-type computer system is composed of a I-type 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 II-type computer system turns into an I-type computer system, i.e., the II-Type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system.

4) II^+-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: A II^+-type computer system is composed of a II-Type computer system and accelerators. According the same reason as in section II , the II^+-type computer system $\overset{UPM}{\Longrightarrow}$ a II-type computer system. By *3)* in this section, the II-Type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system. Therefore, the II^+-type computer system $\overset{UPM}{\Longrightarrow}$ the I-type computer system.

5) III-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: There is a cluster of single-core CPUs in a III-type computer system, and programs based on MPI are for it. We imagine that the III-type computer system is composed of a single-core CPU and the cluster of single-core CPUs. We use similar ideas as in section II and regard the cluster of single-core CPUs as 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. The deriver of the I/O device is an application written in MPI and running on the "cluster-of-single-CPU accelerator".

Thus, the "cluster-of-single-core-CPU accelerator" is regarded as an I/O device, and then a file. The "cluster-of-single-core-CPU accelerator" also "disappears", and the III-type computer system turns into a I-type computer system, i.e., the III-type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system

6) III^+-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: A III^+-type computer system is composed of a III-type computer system and accelerators. According to the same reason as in section II, the III^+-type computer system $\overset{UPM}{\Longrightarrow}$ a III-type computer system. By *5)* in this section, the III-type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system.

Therefore, the III$^+$-type computer system $\overset{UPM}{\Longrightarrow}$ the I-type computer system.

7) IV-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: In a IV-type computer system, the CPUs are multi-core CPUs. According to the same reason as in section II, the IV-type computer system $\overset{UPM}{\Longrightarrow}$ III-type computer system. By *5)* in this section, the III-type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system. Therefore, the IV-type computer system $\overset{UPM}{\Longrightarrow}$ the I-type computer system.

8) IV$^+$-Type Computer Systems $\overset{UPM}{\Longrightarrow}$ I-Type Computer Systems: A IV$^+$-type computer system is composed of a IV-type computer system and accelerators. According to the same reason as in section II, The IV$^+$-type computer system $\overset{UPM}{\Longrightarrow}$ a IV-type computer system. By *7)* in this section, the IV-type computer system $\overset{UPM}{\Longrightarrow}$ a I-type computer system. Therefore, the IV$^+$-type computer system $\overset{UPM}{\Longrightarrow}$ the I-type computer system.

B. Programming of Various Computer Systems Is Built on One API by UPM

The above analysis shows that every type of computer system, from I$^+$-type computer systems to IV$^+$-type computer systems, can be reduced to a I-type computer system, a simplest computer system containing only a single-core CPU, by UMP, 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. file managements of common programming languages for I-type computer systems). The programming of the I-type computer system is the simplest, therefore, UPM can make the programming for various computer systems easier.

IV. 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 communication support to couple multiple applications [3]. This makes coupled appications computing complicated. Here, we present a more natual approach to coupled applications computing, which is based on UPM.

Assume that a coupled applications computing consists of n applications A_1, A_2, ..., A_n, which are based on MPI, and runs over n clusters cluster$_1$, cluster$_2$, ..., cluster$_n$, respectively. See Fig. 4.

According to section II, in UPM, cluster$_i$ (where $i = 1, 2, ..., n$) 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$_i$) + A_i” as a general printer (general-printer$_i$), 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 Fig. 5.

Now we form a new cluster consisting of master-node, node$_1$, node$_2$, ..., node$_n$, as shown in the blue circle in Fig. 6. We call the cluster cluster$_{super}$. Assume that A_{super}, a program based on MPI, runs over the cluster$_{super}$. The A_{super} is responsible for coordinating the n applications A_1, A_2, ..., A_n.

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$_i$ through MPI-I/O from general-printer$_i$ (A_i exactly);

step-2 A_{super} sends the data from node$_i$ to node$_j$ through MPI over the cluster$_{super}$;

step-3 A_{super} sends the data in node$_j$ to general-printer$_j$ (A_j exactly) through MPI-I/O.

Usually A_{super} can not communicate with A_i (where $i = 1, 2, ..., n$) since they are in diffferent parallel MPI worlds. But in UPM, A_i (where $i = 1, 2, ..., n$) 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 more natural since it only relies on its own power to couple multiple applications through MPI.

V. 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, and to cloud 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.

VI. Conclusions and Future Work

In UPM, to carry out data movement between CPUs and accelerators in the common programming language worlds, accelerators can be regarded as one kind of I/O devices. Thus UPM makes programming across CPUs and accelerators turn into usual programming tasks to some extent.

UPM should also work for multicore for OpenMP or Pthread, and to some extent can simplify programming model

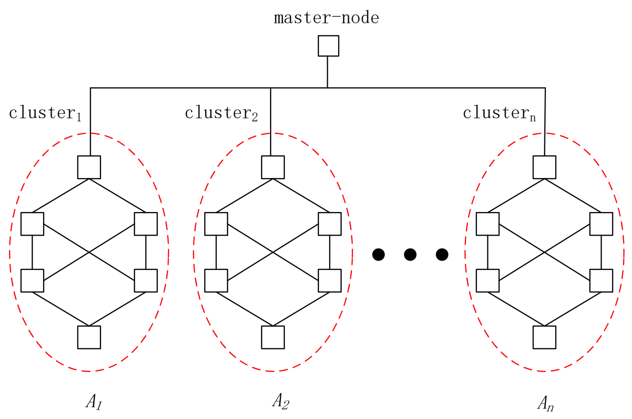

Fig. 4. A coupled applications computing.

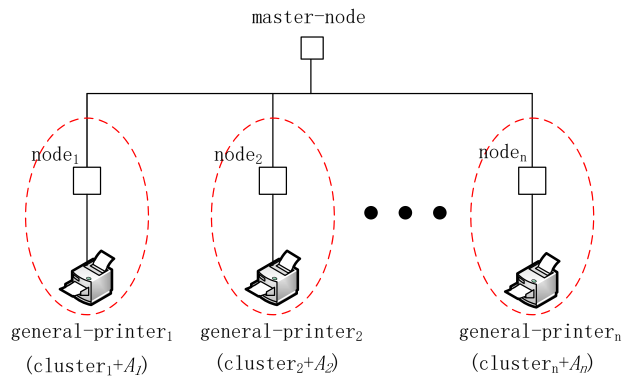

Fig. 5. The clusters as general printers in the coupled applications by UPM.

"common programming languages + OpenMP/Pthread" . This allows OpenMP/Pthread-based code to be separated from the entire program, resulting in higher modularity.

UMP is also applicable to clusters for MPI, and to some extent can simplify programming model "common programming languages + MPI" into usual programming with common programming languages. This makes coupled applications more natural since UPM only relies on its own power to couple multiple applications through MPI.

All types of computer systems can be unified into one (i.e., I-type computer systems) by UPM, and so UPM can truly build the programming of various computer systems on one API (i.e., file managements of common programming languages for I-type computer systems). The programming of the I-type computer system is the simplest, therefore, UPM can make the programming for various computer systems easier.

UPM can be applied to a number of different computing systems equipped with accelerators, ranging from embedded systems, to mobile computing systems, to HPC systems, and to cloud computing systems since CPU and accelerator technologies are used in the computing systems.

UPM can accommodate various kinds of accelerators from different companies. So it is important that users should be provided with an installation interface so that they can connect their accelerators to heterogeneous platforms conveniently by themselves in the sense that the UPM can work. The users can enter the parameters about the accelerators, the programming languages for the accelerators, and their compilers, etc. through this interface into the heterogeneous platforms so that the UPM can work using the parameters. Therefore, its design and implementation will be necessary for UPM technology to be applied widely.

UPM does not restrict which accelerator to assign code to. So there must be an optimal assignment for accelerators using same languages which will improve the code running efficiency. It will be meaningful how to find the optimal assignment.

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] A. Rush, A. Sirasao, and M. Ignatowski, "Unified Deep Learning with CPU, GPU, and FPGA Technologies," White paper, AMD and Xilinx, 2017.
[2] W. Gropp and M. Snir, "Programming for exascale computers," Computing in Science and Engineering. vol. 6, pp. 27-35. 2013.
[3] MpCCI, https://www.mpcci.de/

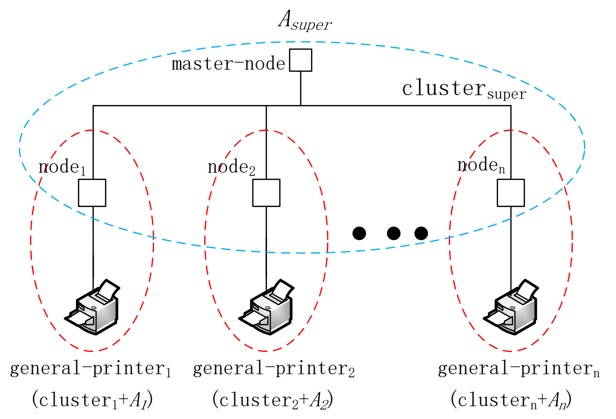

Fig. 6. cluster$_{super}$ consisting of master-node, node$_1$, node$_2$, ..., node$_n$.