Automated Deployment of Radio Astronomy Pipeline on CPU-GPU Processing Systems: DiFX as a Case Study

Ewen Michel, Ophélie Renaud, Karol Desnos, Adam T. Deller, Chris Phillips, Jean-François Nezan

Automated Deployment of Radio Astronomy Pipeline on CPU-GPU Processing Systems: DiFX as a Case Study *

Ewen Michel,[1] Ophélie Renaud,[1] Karol Desnos,[1] Adam Deller,[2]
Chris Phillips,[3] and Jean-François Nezan[1]
[1]*Univ Rennes, INSA Rennes, CNRS, IETR - UMR 6164, Rennes, France*

[2]*CAS, Swinburne University of Technology, Hawthorn, Australia*

[3]*CSIRO, Space and Astronomy, PO Box 76, Epping, NSW 1710, Australia*

Abstract. The next generation of radio telescopes, such as the Square Kilometre Array (SKA), will require peta-FLOPS processing power to handle the massive amount of data acquired. A new generation of computing pipelines are required to address the SKA challenges leading to the integration of the pipelines on a dedicated heterogeneous High-Performance Computing (HPC) system. The tight real-time and energy constraints are driving the community to study the use of hardware accelerators like GPUs in the computing system. Allocating resources, such as processor times, memory, or communication bandwidth, to support complex algorithms in such systems is known as an NP-complete problem. Existing tools such as Dask and Data Activated 流 Graph Engine (DALiuGE) rely on dataflow Model of Computation (MoC) and have proven to be an efficient solution to specify parallel algorithms and automate their deployment. These models are efficient programming paradigms for expressing the parallelism of an application. However, state-of-the-art dataflow resource allocation only targets CPUs and usually relies on complex graph transformations resulting in a time-consuming process. This paper introduces an automated dataflow resource allocation method and code generation for heterogeneous CPU-GPU systems. Our method efficiently and quickly manages pre-scheduling graph complexity, and optimizes the dataflow model to the target architecture. Experimental results show that the proposed method improves resource allocation and speeds up the process by a factor of 13 compared to the best existing method on a basic architecture. Moreover, the execution times of the obtained implementations are comparable to those of manual implementations.

1. Introduction

Integrating CPU-GPU architectures enhances image processing by leveraging the strengths of both components. CPUs provide flexibility and sequential processing, while GPUs excel in massive parallel computing. Tools such as CUDA for NVIDIA GPUs and OpenCL for multiple platforms have facilitated the adoption of accelerators for years. Despite the advantages of CUDA, its effective deployment still requires developer ex-

*This work was supported by DARK-ERA (ANR-20-CE46-0001-01) and has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 873120.

pertise. The dataflow programming paradigm, represented by Dataflow Process Network (DPN), offers a structured approach to manage parallelism and hardware deployment. DPN represent computations as nodes and communication as directed arcs. Methods like Algorithm-Architecture Adequation (AAA) model algorithms and architectures as graphs, automating the Design Space Exploration (DSE) process Grandpierre et al. (1999). While these methods facilitate hardware-software matching, the allocation of resources remains complex. Existing solutions, like dataflow actor clustering Pino et al. (1995), primarily target CPU architectures, and automating the entire process for heterogeneous systems is still a challenge. The proposed method provides automatic resource allocation and code generation for dataflow applications, employs a single application description for any architecture, and optimizes resource allocation and code for CPU-GPU systems.

2. Dataflow resource allocation for CPU-GPU systems

The standard dataflow resource allocation process involves a dataflow graph transformation, called Single rate Directed Acyclic Graphs (SrDAG) transformation, to reveal parallelism and communication, followed by mapping, scheduling, and code translation. The proposed method is based on a clustering method, called Scaling up of Clusters of Actors on Processing Element (SCAPE) Renaud et al. (2024), and happens upstream of the standard resource allocation process. One particularity is that the proposed method takes as input Model of Architecture (MoA) a GPU-oriented System-Level Architecture Model (GSLA) graph Payvar et al. (2021) allowing internal parallelism description of Processing Element (PE), which is suitable for describing GPUs. The main steps of the method are the following:

1. **Extraction**:
   GPU-friendly pattern identification: This involves identifying data parallel pattern matching of dataflow actors, one is called Unique Repetition Count (URC) pattern which consists of at least two sequential actors sharing the same Repetition Vector (RV) coefficient. The other data-parallel pattern is called Single Repetition Vector (SRV) which consists of one isolate actor with a RV coefficient greater than 1.
   Subgraph generation: The identified actors are then isolated into a subgraph *sub* so that all the data parallelism is contained in the subgraph on which the rates are adjusted to match the parallelism of the architecture.

2. **Scheduling**: For any chosen target the chosen scheduling strategy for the cluster is the Pairwise Grouping of Adjacent Nodes for Acyclic graph (APGAN) method Bhattacharyya et al. (1997).

3. **Timing**: This estimates the execution time of the subgraph, the cluster of actors, running on a GPU. Considering the memory transfer between GPUs during the execution of a cluster, its execution time $Tim_{\mathrm{GPU}}(sub)$ is expressed as follow:

$$\begin{aligned} Tim_{\mathrm{GPU}}(sub) &= \sum_{a \in A_{sub}} \left\lceil \frac{Tim_{\mathrm{GPU}}(a) \times \mathbf{q}(a)}{\mathbf{q}(sub)} \right\rceil \\ &+ \sum_{\tau \in F'_{sub}} \alpha_{n_{\mathrm{GPU}}} \cdot \mathrm{size}(\tau) + \beta_{n_{\mathrm{GPU}}} \end{aligned} \tag{1}$$

4. **Mapping**: This estimates the parallelism gain and the transfer loss based on architecture properties. If GPU offloading is estimated beneficial, the process

proceeds with the following steps; otherwise the original SCAPE generates a cluster of actors mapped on CPUs.

$$\text{map} = \begin{cases} \text{CPU}, & \text{if } Tim_{\text{CPU}}(sub) > Tim_{\text{GPU}}(sub) \\ & + \sum_{\tau \in F_{sub}} \alpha_{n_{\text{CPU}}} \cdot \text{size}(\tau) + \beta_{n_{\text{CPU}}} \\ \text{GPU}, & \text{otherwise} \end{cases} \tag{2}$$

5. **Translation**: The subgraph is translated into a CUDA file. First, internal buffers are allocated on the GPU using *cudaMalloc*. The data are then transferred from the CPU to the GPU memory with *cudaMemcpy* to initiate the offload process. Next, actors are successively called according to the schedule. Each actor corresponds to a CUDA kernel launched with «<...»> parameterized with the size of a thread block and the size of the grid based on the data size. The kernels are synchronized with *cudaDeviceSynchronize* for coherence. Finally, processed data are transferred back to CPU memory via *cudaMemcpy*. The simplified and optimized graph is then sent to the standard resource allocation process.

3. DiFX correlator dataflow model

The data processing of radio interferometers such as the SKA can be broadly separated into two main components: the CSP pipeline transforms the ensemble of electric field representations obtained at each antenna into a *visibility* dataset containing coherence measurements between pairs of antennas, while the subsequent SDP pipeline calibrates the visibility dataset and recovers a sky brightness image from it Thompson et al. (1991). An example of a general-purpose Central Signal Processor (CSP) pipeline is the Distributed FX (DiFX) software correlator Deller et al. (2007), and as such this is suitable tested for considering dataflow resource allocation. The correlator is composed of six main steps that transform sky signals into visibility: The *Data Acquisition* involves extracting data from an input file that contains a sampled, quantized representation of the voltage, which corresponds to the measured electric field from the antennas. The *Floating Point Conversion* converts the input integers into a complex consisting of the sample value and the delay representing the phase shift between two telescopes. The *Fringe Rotation* applies the differential Doppler shift correction to compensate for the rotation of the earth that affects the signal. The *Fast Fourier Transform (FFT)* converts the input signal from the time domain to its individual frequency components. This step facilitates subsequent signal compensation in a frequency-dependent manner and helps prevent bandwidth smearing artefacts during image reconstruction. The *Cross-correlation (X)* consists of, for every telescope pair, the Fourier transform output frequency data are multiplied to form correlation products. The *Accumulation* consists of connecting each sample to the previous one, creating the final visibility products.

4. Experiments

The proposed method is applied on the DiFX correlator dataflow model available in Github [*]. To conduct these experiments, the proposed method was implemented within

[*] `https://github.com/preesm/preesm-apps`

the Parallel and Real-time Embedded Executives Scheduling Method (PREESM) Pelcat et al. (2014) rapid prototyping framework, which is part of open-source projects. The experiments were conducted using two computing platforms: a desktop computer equipped with an 8-core Intel i5-1335U processor, 16 GB of RAM, and a single Nvidia GeForce RTX 2050 GPU with 4 GB of VRAM. Figure 1 presents the average reduction factor between the method and different resource allocation strategies deploying several configuration complexities of the DiFX dataflow model. The elements compared are the number of actors on the resulting SrDAG and the resource allocation time. The result shows a significant reduction by clustering, reflected in the acceleration of the resource allocation process. Figure 2 compares the execution time between manual and optimized dataflow implementation of the DiFX correlator deployed on CPU and GPU architecture. The figure shows that optimized dataflow implementations compete with manual ones.

	SrDAG	Allocation
STD(1c) / M(1c,1g)	93.67	58088
STD(8c) / M(1c,1g)	93.67	230509
SCAPE(1c) / M(1c,1g)	2.55	13.83
SCAPE(8c) / M(1c,1g)	5.64	38.00

Figure 1. Average reduction factor: the existing methods over the proposed method (M), 1c: 1 core, 1g: 1 GPU

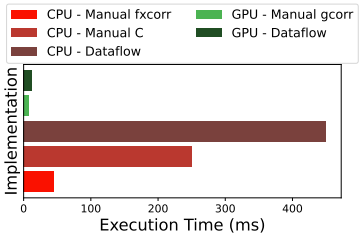

Figure 2. Execution time by DiFX implementation

5. Conclusion

This paper introduces a new automated method for deploying dataflow applications on heterogeneous CPU-GPU systems. The method streamlines the dataflow graph by clustering actors reproducing particular patterns and optimizes the mapping across CPU and GPU. Experimental results demonstrate that the proposed method enhances resource allocation, and accelerates the process by a factor of 13 compared to the best existing method studied on the simplest architecture. Additionally, the produced implementation achieves execution time comparable to manual implementations. Potential directions for future work include improving memory management and data transfer between GPU and CPU.

References

Bhattacharyya, S., et al. 1997, Design Automation for Embedded Systems, 2
Deller, A. T., et al. 2007, Publications of the Astronomical Society of the Pacific, 119, 318
Grandpierre, T., et al. 1999, in 7th International Workshop on Hardware/Software Codesign, 74
Payvar, S., et al. 2021, Design Automation for Embedded Systems, 25, 43
Pelcat, M., et al. 2014, in 6th european Embedded Design in Education and Research Conference, 36
Pino, J., et al. 1995 (University of California at Berkeley)
Renaud, O., et al. 2024, in Journal of Systems Architecture (Elsevier), vol. 154, 103217
Thompson, A., et al. 1991, Interferometry and Synthesis in Radio Astronomy, vol. -1