RRTO: A High Performance Transparent Offloading System for Model Inference on Robotic IoT

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

***Abstract*—The abstract goes here.**

I. Introduction

This demo file is intended to serve as a "starter file" for the Robotics: Science and Systems conference papers produced under LaTeX using IEEEtran.cls version 1.7a and later.

II. Background

A. Transparent offload on Robots

When the robot performs GPU calculation locally, the application specifies the calculation to be performed on the GPU by defining the CUDA kernel function. The specific call process is shown in the figure of 1 :

A The robot application completes the whole calculation process of applying each service by calling different operators in turn.

B Each operator goes through the Unified Operator API to find the local CUDA runtime library based on the data type of the application running device.

C The local CUDA runtime API is loaded by default.

D The robot local CUDA library starts the corresponding CUDA core function on the robot GPU and returns the calculation result to the upper-layer application.

NVIDIA GPU computing power unloading tool This project rewrites a new dynamic link library, in which the function of the same name is defined, and the LD_PRELOAD environment variable is used to make the custom dynamic link library load preferentially, and the dynamic linker will parse the original library function into the library function written by itself, so as to achieve the intercept of the library function. After that, through the management of GPU memory and the modification of CUDA kernel function startup, the call information such as the computation-related data and the specific parameters of the related kernel function is sent to the remote server, so as to realize the transparent unloading of GPU computing power. Step C in the above process is mainly modified. As with the local calculation using the robot GPU, after going through steps A and B, that is, after finding the corresponding CUDA runtime API for each operator in the robot application, the NVIDIA GPU power offloading tool is loaded preferentially. This NVIDIA GPU Power offloading tool performs the following steps:

(1) By modifying the dynamic connection library, each operator through the CUDA runtime API preferentially invokes the computing power with the same name as the CUDA runtime API to unload the client, thus identifying and intercepting all CUDA core function calls.

(2) The computing power unloading client transfers the CUDA runtime API and required parameters to the GPU server over the network.

(3) The server starts the corresponding CUDA core function on the cloud GPU and completes the corresponding self-calculation.

(4) The power offload server passes the calculation result back to the power offload client and returns the calculation result to the upper layer application.

III. Overview

The core problem facing this NVIDIA GPU power offloading system is how to solve the high overhead caused by wireless transmission in transparent offloading in real robot environment. As the figure below shows, in a real robot environment, a single application needs to call the CUDA kernel thousands to tens of thousands of times per minute. For example, one gesture recognition with KAPAO requires about 20 CUDA kernel calls, while smooth gesture recognition calculations require gesture recognition at a frequency of no less than 10hz, which means that KAPAO needs to call the CUDA kernel at least 12,000 times per minute. If you use the simple transparent offloading method, that is, one RPC transfer per CUDA kernel call. In the case of network fluctuations, according to the WiFi6 standard, a round trip delay (RTT) takes 2ms to calculate, and only the wireless network transmission process (that is, the RTT process) consumes 24s in 12,000 RPC transfers. Therefore, the use of simple transparent uninstall methods can introduce serious network transmission overhead, which in turn affects the uninstall computing experience. For example, for vacuum cleaners, high latency can lead to sluggish path planning, making the robot slower to complete cleaning tasks or even run into obstacles.

Therefore, this NVIDIA GPU power offloading system proposes an automatic recording and replay mechanism to reduce the network transmission overhead during the transparent offloading process of GPU power. For the same household robot application, the order of CUDA cores invoked to process different data inputs is usually fixed. For example, multi-person pose recognition requires different images to be input into the same neural network model for recognition. The automatic recording replay mechanism automatically records the CUDA kernel call sequence of the model used by the home robot application and automatically repeats the use process of the model on the GPU service area. This mechanism does not need to wait for the robot to repeat the specific CUDA kernel call during RPC transmission, which greatly reduces

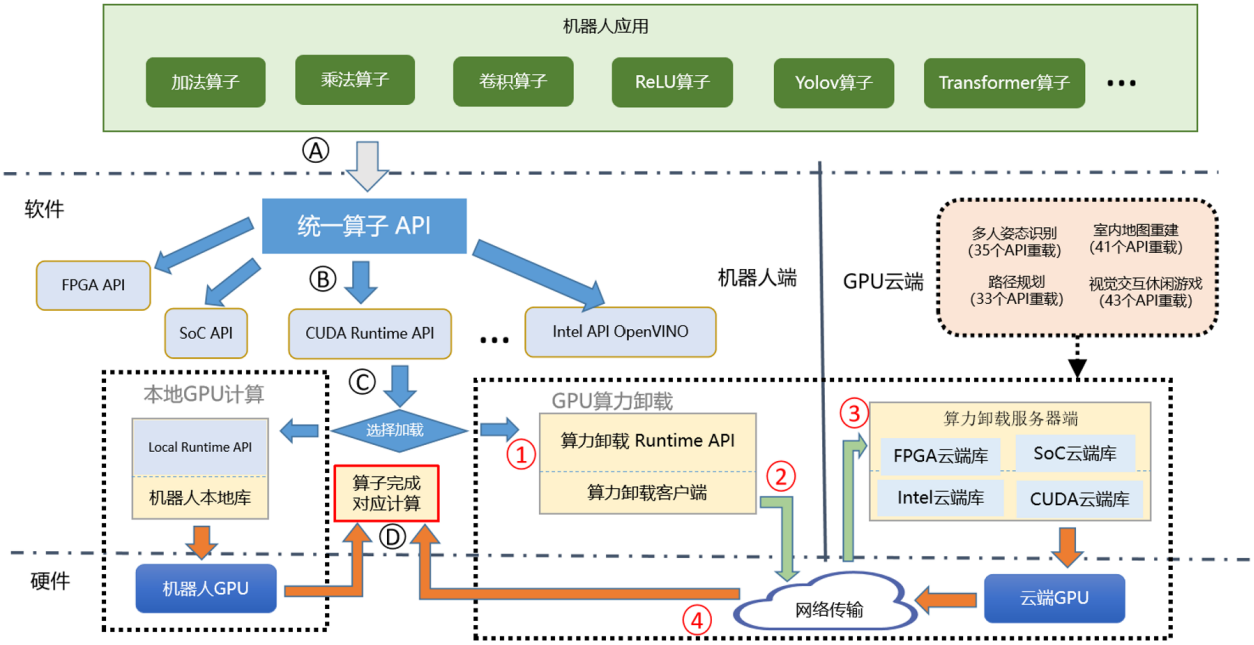

Fig. 1. Transparent Offloading System for Model Inference on Robotic IoT

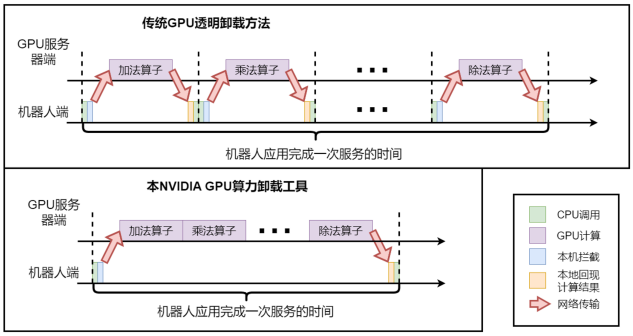

Fig. 2. Record/Replay Mechanism

the number of repeated RPC transfers and thus reduces the network transmission overhead.

IV. Design

This demo file is intended to serve as a "starter file" for the Robotics: Science and Systems conference papers produced under LaTeX using IEEEtran.cls version 1.7a and later.

V. Implementation

This demo file is intended to serve as a "starter file" for the Robotics: Science and Systems conference papers produced under LaTeX using IEEEtran.cls version 1.7a and later.

VI. Evaluation

This demo file is intended to serve as a "starter file" for the Robotics: Science and Systems conference papers produced under LaTeX using IEEEtran.cls version 1.7a and later.

VII. Conclusion

The conclusion goes here.

Acknowledgments

References