RICH: Heterogeneous Computing for Real-Time Intelligent Control

Jintao Chen , Yuankai Xu , Yinchen Ni , An Zou , and Yehan Ma

Shanghai Jiao Tong University
{jintaochen, xuyuankai_095, niyinchen, an.zou, yehanma}@sjtu.edu.cn

***Abstract*—Over the past years, intelligent control tasks, such as deep neural networks (DNNs), have demonstrated significant potential in control systems. However, deploying intelligent control policies on heterogeneous computing platforms presents open challenges. These challenges extend beyond the apparent conflict between intensive computation and timing constraints and further encompass the interactions between task executions and complicated control performance. To address these challenges, this paper introduces RICH, a general and end-to-end approach to facilitate intelligent control tasks on heterogeneous computing architectures. RICH incorporates both *offline Control-Oriented Computation and Resource Mapping (CCRM)* and *runtime Most Remaining Accelerator Segment Number First Scheduling (MRAF)*. Given the control tasks, the CCRM starts with balancing the computation workloads and processor resources with the goal of optimizing overall control performance. Subsequently, the MRAF employs segment-level real-time scheduling to ensure the timely execution of tasks. Extensive experiments on the robotic arms applications (by hardware-in-the-loop simulator) demonstrate that the RICH can work as a general and end-to-end approach. These experiments reveal significant improvements in control performance, with enhancements of 50.7% observed for intelligent control applications deployed on heterogeneous computing platforms.**

***Index Terms*—heterogeneous processor, real-time scheduling, intelligent control, cyber-physical system**

I. Introduction

Modern control systems, such as robotics, drones, and autonomous cars, are embracing intelligent control policies. For example, deep neural networks (DNNs) have achieved tremendous success in feedback control in recent years since they combine the strengths of machine learning (ML) for scalability with analytical traceability [1], [2]. Heterogeneous computing platforms, featuring diverse computing resources, such as CPUs, GPUs, and FPGAs, have become necessary for executing and accelerating ML-based algorithms [3], [4]. The incorporation of accelerators, such as GPUs and FPGAs, with classic CPUs enables the acceleration of intelligent control policies, which is particularly vital in scenarios with stringent timing constraints. With the control systems increasingly incorporating computationally intensive tasks, there is a growing need for a general and end-to-end approach to facilitate these tasks on the complicated heterogeneous architecture.

As shown in Fig. 1, the physical parts of the architecture comprise multiple physical plants (i.e., robots, tanks), and a large number of sensing (e.g., barometers, thermometers) and actuation terminals (e.g., motors) while the cyber parts, which comprises communication and computation, play roles in generating control commands by transmitting sensing/actuation data and executing control tasks. Control tasks are usually executed on the embedded device with heterogeneous computation resources located near physical plants with reliable and wired connections to sensors and actuators. However, as modern control systems continue to integrate data-intensive intelligent control tasks and an increasing number of control loops, significant challenges emerge.

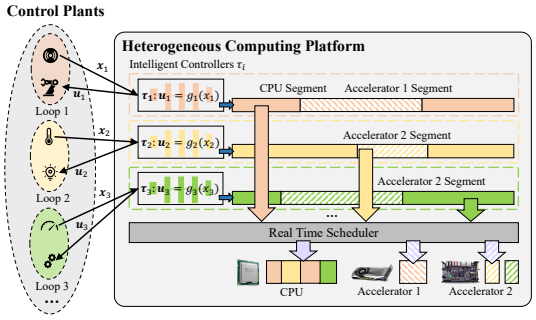


Fig. 1: Intelligent control systems over heterogeneous computing platforms.

Mainstream computational tasks are generally assessed by quality of service (QoS), while intelligent control tasks emphasize timing and control performance. The execution method for intelligent control tasks plays a critical role in achieving these goals. Unlike traditional homogeneous processors, the interleaved execution pattern across heterogeneous processors intensifies task dependencies and competition [5], complicating real-time deadline adherence and resource utilization optimization. Furthermore, the disparity between computation latency and control performance further exacerbates these challenges.

To address these challenges and enable efficient execution of intelligent control tasks within heterogeneous architectures, this paper presents RICH (an acronym for Heterogeneous Computing for Real-Time Intelligent Control Systems). The contributions of this work are summarized as three-fold.

- By modeling the interaction of control performance and heterogeneous computational resources, we introduce a computation and resource mapping strategy for intelligent control tasks to optimize performance.
- We present a runtime segment-level scheduling mechanism for real-time intelligent control tasks to ensure the runtime timing and control performances. Corresponding response time analysis is performed.
- We develop a hardware-in-the-loop simulator that inte-

Yehan Ma is the corresponding author. This work is supported in part by the NSF of China under Grants 62473254 and 62202287.

grates Simulink Desktop Real-Time (SLDRT) for simulating physical plants and real heterogeneous computing platforms for the execution of intelligent control tasks.

Extensive experiments with real CPU-GPU platforms are conducted to demonstrate that RICH can improve control and timing performances.

II. Related Work

Intelligence technologies, such as DNNs, have led to revolutionary changes in the control of robotics [6]–[8], multi-agent systems [9]–[11], and industrial automation [12]–[14] in recent years. Chen et al. [1], [15] use neural networks (NNs) to approximate the optimal control laws. Klaučo et al. [2] propose to apply ML to accelerate primal active set methods used to solve quadratic programming (QP)-based model predictive control (MPC) problems. However, all the above works assume that the computation platform of intelligent control policies is a black box and ignores impacts and the potential benefits from the heterogeneous computation.

In recent years, it has been proved that different task execution patterns can lead to various performances on control systems. For example, Dai et al. [16] adapt the period of control tasks at runtime based on historical measurements, considering the task set with one control task and multiple non-control tasks. Wu et al. [17] propose composite resource scheduling in real-time networked control systems with a strict execution order of sensing, computing, and actuating segments, which guarantees the resource of the utilization of computation and network less than 100%. However, the above real-time control designs focus on tasks running on single processors or multiple homogeneous processors or wireless network scheduling, which cannot support computationally expensive intelligent control, especially on the more complicated heterogeneous computing architectures.

Targeting the sensing or regular computation tasks on the heterogeneous architecture, Xiang et al. [18] propose DART, which allocates heterogeneous resources to DNNs on the layer level with response time analysis and optimizes worst-case execution time. Kang et al. [19] introduce quantization and runtime layer migration to further reduce inference time. Ling et al. [20] adopt operator fusion to optimize the execution time of DNNs. Deadline miss rate of real-time tasks is reduced through block-level optimization for parallel DNNs. While it has been demonstrated that optimizing task execution and employing scheduling approaches can enhance control performance, previous studies have yet to present a general and end-to-end approach for facilitating control tasks on popular heterogeneous architectures. Therefore, we introduce RICH, aiming to address this gap from offline resource mapping to runtime scheduling and finally effectively deploy real-time intelligent control tasks on heterogeneous architectures.

III. System Modeling

In this section, we present the modeling of control systems and the control tasks on heterogeneous computing platforms, based on which we propose the RICH framework.

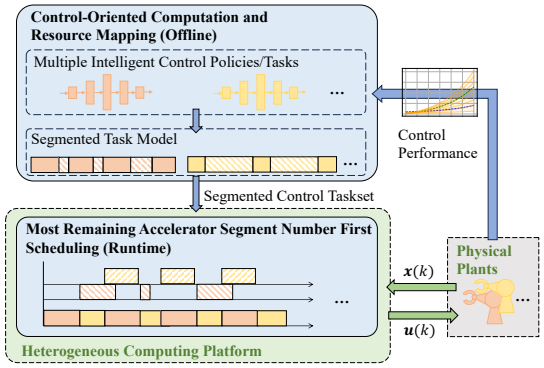


Fig. 2: RICH framework.

A. Control loops

The general model of control loop i, i.e., a control unit in the control system, is presented as follows,

$$\boldsymbol{x}_i(k+1) = f_i(\boldsymbol{x}_i(k), \boldsymbol{u}_i(k)), \boldsymbol{u}_i(k) = g_i(\boldsymbol{x}_i(k)), \quad (1)$$

where $\boldsymbol{x}_i(k) \in \mathbb{R}^n$ is the physical state vector, $\boldsymbol{u}_i(k) \in \mathbb{R}^m$ is the control command vector, $f_i(\cdot)$ is the dynamic and kinematic model of the physical plant, and $g_i(\cdot)$ is the control policy, $i \in \{1, 2, ..., N\}$ is the control loop (control task) index, k is the time index of control periods T_i, N is the number of control loops. The heterogeneous computing platform is connected to multiple physical plants via wired networks. The physical states $\boldsymbol{x}_i(k)$ of the plants are collected by sensors and sent to the heterogeneous computing platform periodically, which executes control policies $\boldsymbol{u}_i(k) = g_i(\boldsymbol{x}_i(k))$ in each period T_i and generates actuation commands $\boldsymbol{u}_i(k)$ for actuators to actuate the physical plants.

B. Intelligent Control Tasks

The control policy for each loop is regarded as the periodic control task, τ_i, $i \in \{1, 2, 3, ..., N\}$, with a period of T_i. The deadlines D_i of control tasks are *equal* to their periods, i.e., $D_i = T_i$. Control tasks τ_i share the heterogeneous cores, i.e., CPU, GPUs, etc. We target the most fundamental case, which has one CPU core and multiple accelerator cores, such as GPU SMs or FPGA IP cores [18]. Any heterogeneous system can be regarded as a combination of this fundamental case. The CPU cores apply real-time scheduling by the operating system and support the preemption, but each accelerator core is generally not preemptive [20]–[22].

One of the classic task models on heterogeneous architecture is the segmented model [23]–[25], which can be expressed as a 3-tuple,

$$\tau_i = \left((C_i^1, A_i^1, C_i^2, ..., A_i^{M_i-1}, C_i^{M_i}), D_i, T_i\right). \quad (2)$$

In this model, a control task τ_i contains M_i CPU segments (CSs) and $M_i - 1$ accelerator segments (ASs). C_i^m and A_i^m denote the worst-case execution time (WCET) of CS and AS.

C. RICH Framework

To effectively facilitate the tasks on the heterogeneous computing architectures, we propose RICH, as shown in Fig. 2, which incorporates offline control-oriented computation and resource mapping (CCRM) and runtime most remaining accelerator segment number first (MRAF) scheduling. In this

paper, we take the NN as an example of the intelligent control task. Each layer of NN is treated as the basic unit to map with heterogeneous resources and schedule with other tasks. In the stage of CCRM, we model the intelligent control task, i.e., NN, with a segmented model and establish the mapping between the segmented model and heterogeneous computing resources targeting overall control performance optimization. CCRM serves as the foundation of the MRAF. In the stage of MRAF, we present a runtime segment-level scheduling mechanism for real-time intelligent control, which assigns available heterogeneous computation resources to certain segments at runtime to achieve both timing and control performance.

IV. Control-Oriented Computation and Resource Mapping

We present the CCRM, as shown in Fig. 3, which maps the computation in intelligent control tasks to different types of processor cores in order to optimize the control performance.

A. Layer-level Resource Type Selection

Usually, there is a complicated interaction between control performance $\mathcal{J}_i$ and the end-to-end latency Δt_i. The deviation with respect to latency can be approximated as [26],

$$\mathcal{J}_i = q_i e^{h_i \Delta t_i} + v_i, \tag{3}$$

where q_i, h_i, and v_i are the approximation coefficients, which can be derived by data fitting.

An NN control task is specified as

$$\tau_i = \big((l_{i,1}, l_{i,2}, ..., l_{i,L_i}), T_i, D_i\big), \tag{4}$$

where $l_{i,j}$ indicates the resource type for layer j, i.e. $l_{i,j} = 1$ for CPU core and $l_{i,j} = 0$ for the accelerator cores, $j \in \{1, 2, ..., L_i\}$, and L_i is total layer number of τ_i. In order to utilize the limited heterogeneous computation resources to optimize the overall control performance of the multi-loop system, we establish an optimization problem for layer-level CCRM as follows.

$$\underset{l_{i,j}, i\in\{1,...,N\}, j\in\{1,...,L_i\}}{\text{minimize}} \quad \sum_{i=1}^{N} w_i \mathcal{J}_i \tag{5a}$$

$$\text{subject to} \quad \delta_i \leqslant T_i \tag{5b}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{L_i} \frac{\omega^c}{T_i} l_{i,j} \pi^c_{i,j} \leqslant \gamma \tag{5c}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{L_i} \frac{\omega^a}{T_i} (1 - l_{i,j}) \pi^a_{i,j} \leqslant \eta \tag{5d}$$

$$l_{i,j} \in \{0, 1\} \tag{5e}$$

where (5a) is the optimization objective, which is the overall control performance of N control tasks, w_i indicates the specified importance of control loop i. For simplicity, we assign δ_i to Δt_i in (3). (5b) is the execution time constraint. In CCRM, we only consider the execution time δ_i of individual control tasks. The affects of other control tasks on response time will be discussed in Sec. V.

$$\delta_i = \sum_{j=1}^{L_i} \big(l_{i,j} \omega^c \pi^c_{i,j} + (1 - l_{i,j}) \omega^a \pi^a_{i,j}\big) \tag{6}$$

As in Eq. (6), δ_i is composed of CPU time, which is the first term on the right, and accelerator time, which is the second, where $\pi^c_{i,j}$ and $\pi^a_{i,j}$ are the WCET of layer j on CPU and accelerator, respectively. $\pi^c_{i,j}$ and $\pi^a_{i,j}$ can be profiled experimentally offline in advance. Eqs. (5c) and (5d) indicate the constraints of CPU and accelerator resources, respectively, where ω^c, ω^a are safe margins since the profiled execution time may not cover WCET. ω^c and ω^a are adjusted based on the fluctuation of the profiled execution time or the stringency of timing performance. And γ and η are restrictions of the resource utilization ratio, which are adjusted based on the workloads of other applications. With only intelligent control tasks, the default value of γ, η are the core counts of corresponding resources. Prob. (5) is a binary nonlinear programming (BNLP) problem, and we solve it using MATLAB OPTI Toolbox. The overhead of CCRM is tested in Sec. VI-D.

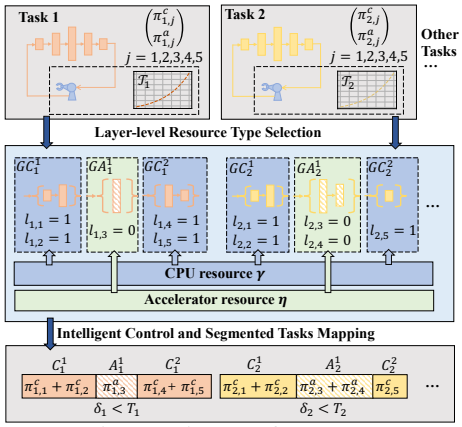


Fig. 3: Diagram for CCRM.

B. Control and Segmented Tasks Mapping

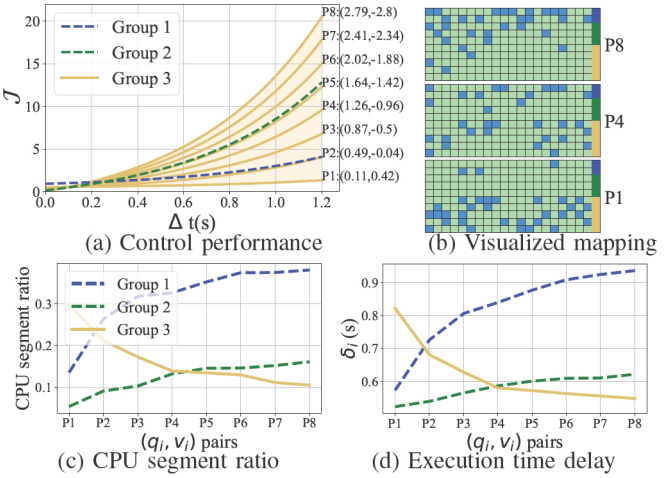
Given the resource type $l_{i,j}$ derived by Prob. (5), we map the intelligent control task model of Eq. (4) to the segmented model of Eq. (2). GC^m_i denotes the m-th consecutive layer group assigned to CPU core, i.e., $l_{i,j} = 1, \forall j \in GC^m_i$. GA^m_i denotes the m-th consecutive layer group assigned to the accelerator cores, i.e., $l_{i,j} = 0, \forall j \in GA^m_i$. Thus, we derive the m-th segment on the CPU and the accelerator core, which are CS C^m_i and AS A^m_i in Eq. (4), respectively.

$$C^m_i = \sum_{j \in GC^m_i} \pi^c_{i,j}, \quad A^m_i = \sum_{j \in GA^m_i} \pi^a_{i,j} \tag{7}$$

In this way, we map the computation of each network layer to CSs and ASs in the segmented model.

C. Intermediate Results

We develop a numerical example to illustrate the effectiveness of CCRM. There are 10 control loops/tasks, and Groups 1, 2, and 3 contain 2, 3, and 5 tasks, respectively. We fix the $\mathcal{J}_i$ parameters (h_i, q_i, v_i) in Eq. (3) of Groups 1 and 2 to $(1.77, 0.43, 0.49)$ and $(1.77, 1.72, -1.56)$, respectively, and fix h_i of Group 3 to 1.77 while changing (q_i, v_i) gradually from $P1$ to $P8$, as shown in Fig. 4a. Each NN control task has 20 layers with $\pi^c_{i,j}$ and $\pi^a_{i,j}$ randomly selected in the ranges of $[1, 150]$ and $[1, 50]$ ms for 100 times, respectively, with each (q_i, v_i) pair. Fig. 4b shows the visualized mapping results $l_{i,j}$, where the i-th row and j-th column of the grid mean layer j of the control task i and blue/green means $l_{i,j}$ is assigned to CPU/accelerators. As Group 3 becomes



(a) Control performance (b) Visualized mapping
(c) CPU segment ratio (d) Execution time delay

Fig. 4: Preliminary results of CCRM.

more sensitive to Δt, more layers in Group 3 are assigned to accelerators (more green grids) to speed up task execution. Figs. 4c and 4d show the average CPU segment ratio $\sum_{i\in Group}\sum_{j=1}^{L_i} l_{i,j}/\sum_{i\in Group} L_i$ and execution time δ_i with the changes of (q_i, v_i) of Group 3. At the same time, the CPU segment ratio decreases in Group 3 and increases in Groups 1 and 2 due to the limitation of accelerator resources. The same happens to the resultant execution time. Group 1 has the highest increase rates of CPU segment ratio and execution time since the control performance is not sensitive to Δt as indicated in Fig. 4a.

V. Most Remaining Accelerator Segment Number First Scheduling

A. Scheduling Mechanism

Since, compared with CS, AS is difficult to be preempted for higher-priority tasks, prioritizing AS may effectively improve the timing performance of control tasks. Different from commonly used rate monotonic (RM) [27] and earliest deadline first (EDF)-like scheduling [24], which lean more on task rates/deadlines or CSs, we explore a novel segment-level dynamic-priority scheduling mechanism, called MRAF, which assigns priorities based on the number of remaining accelerator segments (RSs), i.e., more RSs lead to higher priorities. The RS number of τ_i is denoted as r_i. We schedule the tasks in a non-increasing order of r_i, and lower index one is executed first if two tasks have the same r_i, where tasks are indexed in a non-decreasing order of period T_i. If an AS of a task is finished, the priority order is recalculated. For the last CS $C_i^{M_i}$, CSs from other tasks could only preempt current segment once, implemented by recording preempted segments.

Runtime Complexity. For a taskset $\boldsymbol{\tau}$ with N tasks, every task contains at most M ASs. And we consider the situation that jobs of a task must be executed in turn, and a job cannot be executed until its preceding one is finished [24], indicating there are at most N jobs competing the computation resources at the same time. In MRAF, once a AS of a job is finished, the priority of the job is recalculated. For simplicity, the priority can be assigned the remaining segment number. * The complexity of one calculation is just $O(1)$. During the execution progress of a job, the update of priorities occurs M times and takes up to $O(M)$ running time. Adding up all N tasks, the total runtime complexity $O(MN)$.

B. Worst-case Response Time Analysis

Although MRAF indicates its superior timing performance in Sec. V-C, the worst-case response time (WCRT) analysis is challenging since MRAF is a segment-level scheduling algorithm. In this work, we provide a conservative upper bound of WCRT. The WCRT of task τ_i is defined as R_i, which is the maximum time interval between release time and finishing time. Similarly, the response time of m-th CS is defined as R_i^m. Since a CS could be preempted by higher-priority CSs, while an AS could be blocked but cannot be preempted by other ASs, the WCRT of τ_i is calculated by summing up the WCRT of CSs, and the block time and execution time of ASs.

Theorem V.1. *Consider a taskset $\{\tau_1, \tau_2, ..., \tau_N\}$ under MRAF scheduing, the WCRT of task $\tau_i, i \in \{1, 2, ..., N\}$ is*

$$R_i = \sum_{m=1}^{M_i} R_i^m + \sum_{m=1}^{M_i-1} (A_i^m + B_i^m) \tag{8}$$

where the WCRT for the m-th CS is

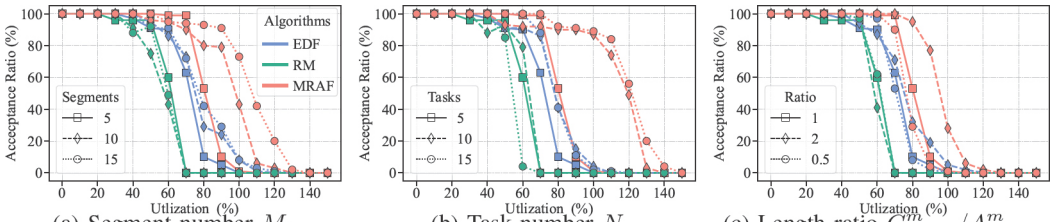
$$R_i^m = \sum_{q=1,q\neq i}^{N} \sum_{k=1}^{\Lambda_{q,i,m}-1} C_q^k + \sum_{\tau_q\in hp(\tau_i)} C_q^{\Lambda_{q,i,m}} + C_i^m \tag{9}$$

with $\Lambda_{i,q,m} = M_q - M_i + m$, $hp(\tau_i)$ is the set of tasks with higher priority than τ_i and B_i^m is the worst-case block time that other ASs block the m-th AS of τ_i, calculated by:

$$B_i^m = \max_{q\neq i, 1\leqslant k\leqslant M_q-1} A_q^k \tag{10}$$

Proof. For R_i^m of m-th CS of τ_i, τ_i^m, the remaining ASs number is $(M_i - 1) - (m - 1) = M_i - m$. All other jobs of task $\tau_q, q \neq i$, preempt CS τ_i^m if they have more remaining AS number than $M_i - m$. And they will no longer preempt it once their remaining ASs are less than the $M_i - m$ (condition 1), or equal to $M_i - m$ with $\tau_q \in lp(\tau_i)$(set of tasks with lower priority than τ_i) (condition 2). As τ_q has $M_q - 1$ ASs in total, the largest preemption CS count of τ_q on τ_i^m is $(M_q-1)-(M_i-m) = M_q-M_i+m-1 \triangleq \Lambda_{i,q,m}-1$. Adding up preemption of all other jobs, the worst-case preemption time caused by condition 1 is $\sum_{q=1,q\neq i}^{N}\sum_{k=1}^{\Lambda_{q,i,m}-1} C_q^k$. For condition 2, adding up the execution time of $\Lambda_{i,q,m}$-th CSs with $\tau_q \in hp(\tau_i)$, the worst-case preemption time is $\sum_{q=1}^{i-1} C_q^{\Lambda_{q,i,m}}$. Adding up conditions 1, 2 and the execution time of τ_i^m, R_i^m is derived. For blocking time B_i^m, since we use the accelerator as an entity in this work, the AS may be blocked by the ones of τ_q. The worst case occurs when the longest AS of other tasks occupy the accelerator at the same time. Therefore, B_i^m is derived by Eq. (10). □

*When the number of ASs or tasks exceeds 99, which is the real-time priority value range of Linux operating system, a heap could be maintained to sort the remaining segment numbers for all N tasks with single insert complexity of $O(\log N)$ and total runtime complexity of $O(MN\log N)$.



(a) Segment number M_i (b) Task number N (c) Length ratio $C^m_{i,max}/A^m_{i,max}$

Fig. 5: Schedulability of RICH with different configurations.

TABLE I: MRAF scheduling experiment setup

Parameters	Value (Default *)
Number of tasks N in taskset	5*,10,15
Task type	periodic
Number of CPU segments M_i in each task	5*,10,15
Taskset number	100
AS length A^m_i(ms)	[1 to 5]
Ratio of maximum segment length $C^m_{i,max}/A^m_{i,max}$	0.5,1*,2
Task period and deadline	T_i/D_i
Number of CPU and accelerator cores	1,2

C. Intermediate Results

We conduct schedulability test experiments on a real heterogeneous platform with Intel i5-12490F CPU @ 3.00GHz and NVIDIA RTX 3060 GPU @ 1.78GHz using synthetic workloads, i.e., matrix addition and multiplications. The implementation of MRAF is detailed in Sec. VI-A. The tasksets are generated following predefined rules as shown in Tab. I. For various utilization defined as $\sum_{i=1}^{N}\sum_{m=1}^{M_i} C^m_i/T_i$, we assess the acceptance ratio of the tasksets in which all the tasks can be completed before their respective deadlines. We examine the impact of three key factors: segment number M_i, task number N, and the ratio of maximum segment length $C^m_{i,max}/A^m_{i,max}$, i.e., 0.5, 1, and 2 representing the CPU segment length randomly distributed within the ranges of $[1, 2.5]$, $[1, 5]$, and $[1, 10]$, respectively, while keeping other parameters as default values (*). As shown in Fig. 5, with the increase of segment (a) and task (b) numbers, the acceptance ratios under all scheduling algorithms increase, and MRAF achieves the best performance, which demonstrates its scalability. As the length ratio $C^m_{i,max}/A^m_{i,max}$ (c) increases and thus the proportion of ASs decreases, the acceptance ratio grows because of less blocking between non-preemptive ASs.

VI. Evaluation

A. Hybrid Real-time Intelligent Control Simulator

To evaluate the computing on the control systems, we built a real-time *hardware-in-the-loop* simulator, Real-time Intelligent Control Simulator (RTICS), as shown in Fig. 6. Hybrid RTICS integrates 1) intelligent control tasks scheduled on *real* heterogeneous computing platforms; 2) MATLAB/Simulink Desktop Real-Time (SLDRT), which *simulates* physical plants in real-time; 3) *real* Ethernet networks connect the *simulated* physical plants and *real* computing platform†; 4) and the interfaces and management script are developed to manage all interfaces of RTICS contributing to its scalability.

†The Ethernet communication between SLDRT and heterogeneous computing platform takes less than 3 ms based on experimental measurements.

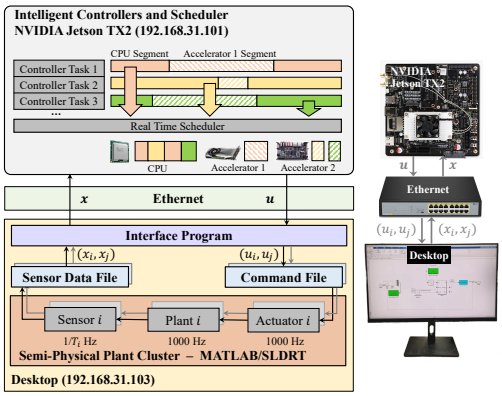


Fig. 6: Architecture of RTICS.

As shown in Fig. 6, sensors in SLDRT sample every T_i, and write measurements $\boldsymbol{x}_i(k)$ into Sensor Data Files, which is monitored by the interface program and forwarded to the heterogeneous computing platform via Ethernet through Socket. The τ_i is released on the heterogeneous computing platforms once $\boldsymbol{x}_i(k)$ is received. And the control tasks are scheduled according to different mapping and scheduling algorithms. When τ_i is completed, $\boldsymbol{u}_i(k)$ is sent to the interface program via Ethernet and written into Command Files, which is read by the actuator in SLDRT with system frequency. The actuator in SLDRT updates the actuation once it discovers an update of $\boldsymbol{u}_i(k)$. The event-trigger actuation enables actuators to respond to updated actuation commands immediately [28]–[30]. We incorporate multi-rate simulation in SLDRT by setting the frequencies of sensor sampling to $1/T_i$ Hz, and those of physical plants (system frequency) to 1000 Hz for simulating continuous nature of physical plants.

B. Experimental Setups

Control Performance Metric: We apply the most general control performance metric, linear quadratic regulator (LQR) defined in Eq. (11) [26], [31],

$$\mathcal{J}_i = \int_{t=0}^{\infty} (\boldsymbol{e}_i{}^T(t) Q \boldsymbol{e}_i(t) + \boldsymbol{u}_i^T(t) R \boldsymbol{u}_i(t)) dt \tag{11}$$

where $\boldsymbol{e}_i(t) = \boldsymbol{x}_i(t) - \boldsymbol{x}_i^{ref}(t)$ is state error, $\boldsymbol{x}_i^{ref}$ is tracking reference, Q and R are positive semi-definite matrixes.

Physical Plants: We model joint position control of robotic arms [32]:

$$\tilde{\theta}_i(k+1) = K_i^g g_i\big(\tilde{\theta}_i(k)\big). \tag{12}$$

where $\tilde{\theta}(k) = \theta(k) - \theta^{ref}(k)$, $\theta(k)$ is measured joint position, $\theta^{ref}(k)$ is position tracking reference, and intelligent controller $u_i(k) = g_i\big(\tilde{\theta}_i(k)\big)$, K_i^g is the mass constant of the link. We consider two types of plants with K_i^g of $1/100$

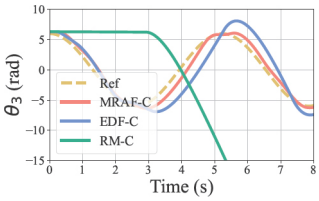


Fig. 7: Response curve sample of θ_3.

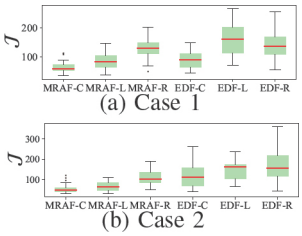


Fig. 8: The distributions of control performance $\mathcal{J}$.

(PLANT1) and $1/110$ (PLANT2), with control rates $1/T_i$ of 10 Hz and 20 Hz, respectively. With $Q = 10^{-2}$ and $R = 10^{-7}$, the control performance parameters h_i, q_i, v_i in Eq. (3) are $(53.89, 0.0079, 3.66)$ and $(65.86, 0.0009, 9.33)$. To evaluate RICH under different computation workloads, we setup two cases. Both cases control 6 physical plants. In case 1, loops 1-3 control PLANT1 and loops 4-6 control PLANT2. In case 2, loops 1-4 control PLANT1 and loops 5-6 control PLANT2.

System Settings: The 6 physical plants are simulated with SDLRT running on two desktop PCs with Intel i5-12490F CPU @ 3.00GHz to achieve high-granularity real-time simulation. We test RICH on NVIDIA TX2, and use one Cortex-A57 CPU and the 256-core NVIDIA Pascal GPU. We implement two feedforward NN with 10 full-connected hidden layers and ReLU activation function. The CSs are implemented in C, while ASs are deployed on GPU by CUDA kernels. CPU and GPU frequencies are fixed to 2 GHz and 1.3 GHz. For CCRM (Prob. (5)), γ and η are set to 1 since we use one of the CPU cores and utilize GPU as an entity. Since we observed WCET of GPU fluctuates more, ω^c and ω^a are set to 1 and 1.1. The weights w_i, $i \in \{1, 2, ..., N\}$, are assigned to 1.

Approaches: We compare RICH with other methods from the perspectives of both mapping methods and scheduling algorithms. For *mapping methods*, we compare CCRM to latency-only mapping [18], [20] and random mapping, denoted as '-C', '-L' and '-R'. Random mapping assigns layers to heterogeneous resources randomly, while latency-only mapping minimizes overall execution time by altering the objective in Prob. (5a) to $\sum_{i=1}^{N} \delta_i$. For *scheduling methods*, we compare MRAF with EDF and RM, resulting in 9 combinations, where RICH corresponds to MRAF-C. Each approach is tested in 30 rounds of experiments, with an 8 s interval per round.

C. Experimental Results

Fig. 7 shows the response curve of loop 3 under the combinations of CCRM and different scheduling approaches in Case 1 as illustrative examples. The response curve of MRAF-C tracks the reference the best, while RM-C cannot follow the reference, and the curve diverges since PLANT1 has low rates/priorities. Therefore, in the rest of the section, RM scheduling is not involved. The overall control cost $\mathcal{J}$ is shown in Fig. 8. RICH (MRAF-C) obtains the lowest $\mathcal{J}$ and the best overall control performance. Compared with the other mapping methods, CCRM achieves up to 50.71% improvement of control performance on average. And when mapping methods are fixed, MRAF always performs better

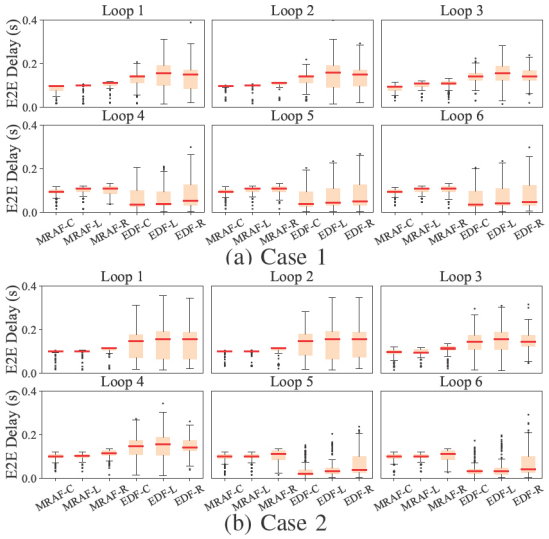


Fig. 9: The distributions of end-to-end delay.

than EDF, which decreases $\mathcal{J}$ by up to 55.07%. Looking into the end-to-end time delay Δt of each loop in Fig. 9, CCRM always achieves a lower latency than other mapping methods. Furthermore, it is observed that different scheduling approaches lead to different trade-offs among different plants. Since EDF prioritizes the execution of tasks with earlier deadlines, the trade-off is in favor of high-frequency tasks for PLANT2. Meanwhile, MRAF assigns priorities according to remaining AS numbers and leans towards equalizing all loops.

D. Offline and Runtime Overhead Analysis

We test the overheads of CCRM and MRAF. Solving Prob. (5) in CCRM offline takes less than 612.46 s, which is affordable because CCRM only needs to be executed once. And for MRAF, the overhead distribution for each job of Case 1 in Sec. VI-C is used as an illustrative example. The average and maximum overhead of loop 1-3 with 4 ASs are 42 us and 63 us, while 64 us and 85 us for loop 4-6 with 7 ASs. We can see that the overhead of MRAF increases (almost linearly) with the number of AS, which is consistent with the complexity analysis in Sec. V-A. The overhead is well acceptable compared to the end-to-end delay of data-intensive control tasks, which is around 50 - 350 ms.

VII. Conclusion

In this work, we proposed RICH, exploring control-performance-driven heterogeneous computing for intelligent control systems. As an end-to-end approach, the RICH incoperates both offline CCRM and runtime MRAF. The CCRM maps the layer-level computation in intelligent control tasks to a segmented model in order to optimize the control performance. The MRAF prioritizes tasks with more remaining accelerator segments, which mitigates waiting and blocking issues associated with nonpreemptive accelerators. Extensive experiments on the hardware-in-the-loop simulator RTICS demonstrate the benefits of RICH in perspectives of both control and timing performances, compared to state-of-the-art mapping and scheduling approaches.

REFERENCES

[1] Steven W Chen, Tianyu Wang, Nikolay Atanasov, Vijay Kumar, and Manfred Morari. Large scale model predictive control with neural networks and primal active sets. *Automatica*, 2022.

[2] Michal Kvasnica Klaučo, Martin Kalúz. Machine learning-based warm starting of active set methods in embedded model predictive control. *EAAI*, 2019.

[3] Juan Liu, Guoqing Xiao, Fan Wu, Xiangke Liao, and Kenli Li. Aapp: An accelerative and adaptive path planner for robots on gpu. *IEEE TC*, 2023.

[4] Qunsong Zeng, Yuqing Du, Kaibin Huang, and Kin K Leung. Energy-efficient resource management for federated edge learning with cpu-gpu heterogeneous computing. *IEEE Transactions on Wireless Communications*, 20(12):7947–7962, 2021.

[5] Ronald B Brightwell. Resource management challenges in the era of extreme heterogeneity. Technical report, Sandia National Lab., Albuquerque, NM, 2018.

[6] Yu-Shun Hsiao, Zishen Wan, Tianyu Jia, Radhika Ghosal, Abdulrahman Mahmoud, Arijit Raychowdhury, David Brooks, Gu-Yeon Wei, and Vijay Janapa Reddi. Mavfi: An end-to-end fault analysis framework with anomaly detection and recovery for micro aerial vehicles. In *DATE*. IEEE, 2023.

[7] Julian Nubert, Johannes Köhler, Vincent Berenz, Frank Allgöwer, and Sebastian Trimpe. Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5(2):3050–3057, 2020.

[8] Tim Salzmann, Elia Kaufmann, Jon Arrizabalaga, Marco Pavone, Davide Scaramuzza, and Markus Ryll. Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms. *IEEE Robotics and Automation Letters*, 8(4):2397–2404, 2023.

[9] Peng Shi and Bing Yan. A survey on intelligent control for multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[10] Qikun Shen, Peng Shi, Junwu Zhu, Shuoyu Wang, and Yan Shi. Neural networks-based distributed adaptive control of nonlinear multiagent systems. *IEEE Transactions on Neural Networks and Learning Systems*, 31(3):1010–1021, 2019.

[11] Hashim A Hashim, Sami El-Ferik, and Frank L Lewis. Neuro-adaptive cooperative tracking control with prescribed performance of unknown higher-order nonlinear multi-agent systems. *International Journal of Control*, 92(2):445–460, 2019.

[12] Baicun Wang, S Jack Hu, Lei Sun, and Theodor Freiheit. Intelligent welding system technologies: State-of-the-art review and perspectives. *JMS*, 2020.

[13] Kang-Di Lu, Guo-Qiang Zeng, Xizhao Luo, Jian Weng, Weiqi Luo, and Yongdong Wu. Evolutionary deep belief network for cyber-attack detection in industrial automation and control system. *IEEE Transactions on Industrial Informatics*, 17(11):7618–7627, 2021.

[14] Jorge Ribeiro, Rui Lima, Tiago Eckhardt, and Sara Paiva. Robotic process automation and artificial intelligence in industry 4.0–a literature review. *Procedia Computer Science*, 181:51–58, 2021.

[15] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D Lee, Vijay Kumar, George J Pappas, and Manfred Morari. Approximating explicit model predictive control using constrained neural networks. In *IEEE ACC*, 2018.

[16] Xiaotian Dai and Alan Burns. Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems. *JSA*, 2020.

[17] Peng Wu, Chenchen Fu, Tianyu Wang, Minming Li, Yingchao Zhao, Chun Jason Xue, and Song Han. Composite resource scheduling for networked control systems. In *IEEE RTSS*, 2021.

[18] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *IEEE RTSS*, 2019.

[19] Woosung Kang, Kilho Lee, Jinkyu Lee, Insik Shin, and Hoon Sung Chwa. Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks. In *RTSS*, 2021.

[20] Neiwen Ling, Xuan Huang, Zhihe Zhao, Nan Guan, Zhenyu Yan, and Guoliang Xing. Blastnet: Exploiting duo-blocks for cross-processor real-time dnn inference. In *20th SenSys*, 2022.

[21] Sujan Kumar Saha, Yecheng Xiang, and Hyoseung Kim. Stgm: Spatio-temporal gpu management for real-time tasks. In *IEEE RTCSA*, 2019.

[22] An Zou, Jing Li, Christopher D Gill, and Xuan Zhang. Rtgpu: Real-time gpu scheduling of hard deadline parallel tasks with fine-grain utilization. *IEEE TPDS*, 2023.

[23] Jian-Jia Chen and Cong Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *IEEE RTSS*, 2014.

[24] Mario Günzel, Georg von der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. Edf-like scheduling for self-suspending real-time tasks. In *IEEE RTSS*, 2022.

[25] Yuankai Xu, Tiancheng He, Ruiqi Sun, Yehan Ma, Yier Jin, and An Zou. Shape: Scheduling of fixed-priority tasks on heterogeneous architectures with multiple cpus and many pes. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[26] Byung Kook Kim. Task scheduling with feedback latency for real-time control systems. In *IEEE RTAS*. IEEE, 1998.

[27] Pontus Ekberg. Rate-monotonic schedulability of implicit-deadline tasks is np-hard beyond liu and layland's bound. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 308–318. IEEE, 2020.

[28] Wei Zhang, Michael S Branicky, and Stephen M Phillips. Stability of networked control systems. *IEEE Control Systems*, 21(1):84–99, 2001.

[29] Liwei An and Guang-Hong Yang. Data-based distributed sensor scheduling for multiple linear systems with H-∞ performance preservation. *IEEE Transactions on Automatic Control*, 67(12):6834–6841, 2021.

[30] Damoon Soudbakhsh, Anuradha Annaswamy, and Harald Voit. Adaptation in network control systems with hierarchical scheduling. *IET Control Theory & Applications*, 13(17):2775–2782, 2019.

[31] Danbing Seto, John P Lehoczky, Lui Sha, and Kang G Shin. On task schedulability in real-time control systems. In *17th IEEE RTSS*, 1996.

[32] Peter Corke. *Robotics, vision and control: fundamental algorithms In MATLAB® second, completely revised*, volume 118. Springer, 2017.