学校代号 10532　　学　　号 S161000900

分 类 号 TP301　　密　　级 普通

湖南大學
HUNAN UNIVERSITY

硕士学位论文

面向无人驾驶的计算结构及其若干算法研究

学位申请人姓名 邹文超

培 养 单 位 信息科学与工程学院

导师姓名及职称 李仁发 教授

学 科 专 业 计算机科学与技术

研 究 方 向 分布式系统

论文提交日 期 2019 年 3 月 5 日

学校代号：10532

学　　号：S161000900

密　　级：普通

湖南大学硕士学位论文

面向无人驾驶的计算结构及其若干算法研究

学位申请人姓名：　邹文超

导师姓名及职称：　李仁发 教授

培　养　单　位：　信息科学与工程学院

专　业　名　称：　计算机科学与技术

论 文 提 交日 期：　2019 年 3 月 5 日

论 文 答 辩日 期：　2019 年 3 月 15 日

答辩委员会主席：　彭蔓蔓教授

Research on Unmanned Vehicle Computing Structure and Related Algorithms

by

ZOU Wenchao

B.E. (Hunan Normal University) 2016

A thesis submitted in partial satisfaction of the

Requirements for the degree of

Master of engineering

in

Computer science and technology

in the

Graduate School

Of

Hunan University

Supervisor

Professor Li Renfa

March, 2019

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　日期：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密□。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　月　日

导师签名：　　日期：　　年　月　日

摘　　要

信息与通信技术领域的不断发展，推动了无人驾驶汽车领域的巨大的进步。无人驾驶汽车有效解决了用户对出行日益提高的安全、便捷、舒适和高效等个性化需求，有着巨大的商业前景和应用价值。无人驾驶技术相关的传感器数据处理，环境感知，深度学习，决策和控制，给无人驾驶汽车的计算能力带来了巨大的压力，因此它对硬件成本和能量的需求也进一步增加。无人车要想成为普通大众买得起的消费品，就必须要降低硬件成本。同时道路汽车功能安全标准 ISO26262 引出了汽车功能安全可靠性的问题，汽车功能应用应该满足相应的可靠性目标。随着绿色环保理念对汽车工业的影响，电动汽车已经成为目前未来汽车工业发展的方向，同时对电机的控制算法比喷油发动机相对简单，因此电动汽车也将是无人驾驶汽车的主要方向。为了增加电动汽车续航里程，在电动汽车中尽量减少电子系统消耗的能量尤为重要。

本文首先对无人驾驶汽车计算任务进行了分类总结，总结分析了无人驾驶汽车集移动端、边缘节点和云端一体的计算结构，介绍无人驾驶汽车领域的一些关键算法。分类讨论了目前无人驾驶汽车移动端计算平台，呈现出分布式异构多核的特点。

然后针对无人驾驶汽车遇到的性能与成本方面的挑战，本文提出了一个在无人车电子系统设计阶段，在满足无人车的安全需求前提下，即实时约束和可靠性约束，优化硬件成本和能耗的任务调度算法。通过快速傅里叶变换、高斯消元和真实汽车应用实验表明，本文提出的算法在满足实时和可靠性目标前提下，可以有效的降低硬件成本和能耗。

最后为了提高无人驾驶应用的可靠性，防止无人车中某个计算节点宕机或者随机硬件错误导致的功能失效，本文提出了一个容错机制下的可靠性目标保障调度算法。对现有的研究的算法进行了优化，不但可以满足更高的可靠性目标，而且使得应用的响应时间和能耗更小。并通过真实汽车应用和随机生成的应用实验证明了本文提出的算法真实有效。

关键词：无人驾驶；异构分布式系统；可靠性保障；硬件成本优化；能耗优化

Abstract

The continuous development of information and communication technology has promoted the great progress of unmanned cars. Unmanned cars have effectively solved users' personalized demands for increasingly improved travel safety, convenience, comfort and efficiency, and have great business prospects and application value. The sensor data processing, environmental perception, deep learning, decision-making and control related to unmanned technology have brought great pressure on the computing capacity of autonomous cars, so it further increases the demand of the hardware cost and energy. If unmanned cars are to become affordable consumer goods for the general public, it is necessary to reduce hardware costs. At the same time, road vehicle functional safety standard ISO26262 raises the problem of functional safety reliability. The application of automobile functions should meet the corresponding target reliability. With the influence of green environmental protection concept on the automobile industry, battery powered vehicles have become the direction of the development of the automobile industry in the future. At the same time, the control algorithm of the motor is relatively simpler than that of the fuel injection engine, so battery powered cars will also be the main direction of unmanned cars. In order to increase the range of electric vehicles, it is particularly important to minimize the energy consumed by electronic systems in battery powered vehicles.

Firstly, this paper classifies and summarizes the computing tasks of unmanned cars, expounds the computing structure of unmanned cars integrating mobile terminals, edge nodes and cloud, and introduces some key algorithms in the field of unmanned cars.This paper classifies and discusses the current mobile computing platform of unmanned cars, which presents the characteristics of distributed heterogeneous multi-core.

Then, aiming at the performance and cost challenges faced by unmanned cars, this paper proposes a task scheduling algorithm to optimize hardware cost and energy consumption under the premise of meeting the safety requirements of unmanned cars, namely real-time constraint and reliability constraint.

Finally, in order to improve the reliability of unmanned vehicle application and prevent the failure of a computing node or random hardware error in unmanned vehicle, an efficient target reliability guarantee scheduling algorithm based on fault-tolerant mechanism is proposed in this paper. The optimization of the existing algorithm can not only meet the higher reliability goal, but also reduce the application response time and energy consumption. The algorithm proposed in this paper is proved to be real and effective by real automobile application and random generation application experiment.

Key Words:Unmanned driving; Heterogeneous distributed system; Reliability guarantee; Hardware cost optimization; Energy consumption optimization

目　录

插图索引

附表索引

第1章 绪论

1.1 研究背景与意义

汽车已经有着 100 多年的发展历史了, 它的出现大大节约了人类的出行时间和出行成本，提高了人类的生产活动效率。但随着社会的不断发展和进步，汽车数量呈现爆发式的增长。这导致了交通拥堵、能源危机、环境污染、交通事故频发, 给人们的生活和城市建设带来了一定的阻碍。随着信息与通信技术领域的不断发展，无人驾驶汽车的出现将有可能解决这些问题。

无人驾驶汽车是一种能够自我感知周围环境，在没有人类驾驶操作下自动行驶的汽车[1]。它主要依靠人工智能、视觉计算、雷达、监控装置和定位系统协同合作，让汽车可以在没有任何人类主动干预情况下，自动安全地行驶。美国国家公路安全管理局(National Highway Traffic Safety Administration, NHTSA)和美国机动工程师协会(Society of Automotive Engineers, SAE)都发布了汽车自动化的分级标准[2]， SAE 给出的汽车自动化标准如下：

Level 0:无自动化。特点是传感探测和决策警报。车辆完全由人类负责驾驶，系统仅给出一些警报。比如并线辅助、车道偏离警告等。

Level 1:提供驾驶辅助。特点是单一功能实现自动化。比如自适应巡航系统、自动紧急制动、停车辅助等。

Level 2:部分自动化。特点是功能组合实现自动化，比如车道保持辅助系统。

Level 3:有条件的自动化。在特定条件下的部分任务实现自动化。比如拥挤辅助驾驶。

Level 4:高度自动化。特定条件下的全部任务实现自动化。比如停车场自动泊车。

Level 5:完全自动化。全部条件下全部任务实现自动化，完全由车辆自动驾驶。

NHTSA 与 SAE 在自动驾驶 0-3 级一致，都是有人驾驶，部分功能实现自动化。本文主要讨论具备高度自动化的无人驾驶汽车，对应于 Level 4 以上。

无人驾驶汽车领域相关的核心技术已经成为国内外的研究热点，无人驾驶汽车不但可带来巨大的经济效益，还具有以下优点：①增强行车安全，美国高速公路安全保险研究所的一项研究表明，在美国机动车事故中，有 94%的事故与人为失误有关，安装有自动安全装置的车辆能使高速公路事故死亡减少 31%[2]；②无人驾驶汽车可以解放人类的双手，人们将拥有更多自由时间，在出行时，人们可以选择休闲或工作；③缓解停车难问题，无人车可以自动完成停车任务，使

用较少预留空间；④减少空气污染，自动驾驶技术可提高能源利用效率，通过更顺畅的加速，减速，能比手动驾驶提高 4%-10%的利用效率。

近年来，随着人们对环境污染的认识不断加深以及能源危机的加剧，新能源汽车得到了大力发展，而其中的纯电动汽车成为新能源汽车中的主体。图 1.1 显示了我国近几年的新能源汽车的销售情况。可以看出纯电动汽车的销售量逐年增长。因此无人电动汽车也将有可能成为未来发展的主要趋势。另一个无人驾驶采用电动汽车的有利条件就是控制喷油发动机的算法比控制电机的算法复杂得多。

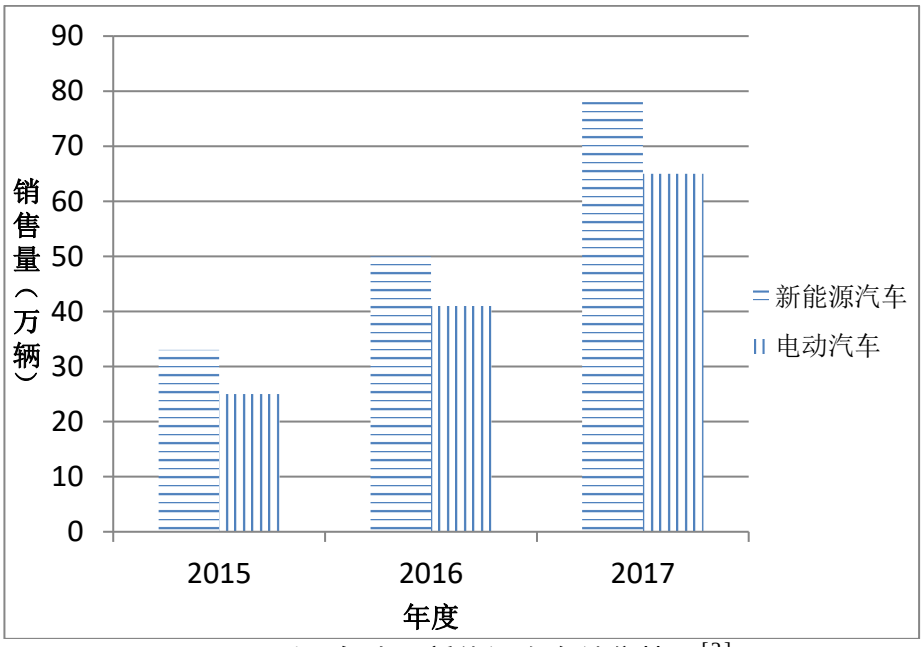

图 1.1 近几年我国新能源汽车销售情况[3]

无人驾驶车辆必须保证人的生命财产安全，因此无人驾驶系统必须满足以下三个要求：其一，系统必须确保捕捉到的传感器数据得到及时处理；其二，如果系统的某部分失效，则系统需要有足够的健壮性能从错误中恢复；其三，系统必须在设计的能耗和资源限定下有效完成所有计算操作。现有的一个领先的无人车驾驶产品的计算平台由两个计算盒组成[2],互为备份，每一个装备有 INTEL Xeon E5 处理器及 4 到 8 个 Nvidia Tesla K80 GPU。在峰值下运行，功耗高达 5000w,每一个计算盒高达 2 万至 3 万美元，这是普通消费者无法接受的。

现有的无人车设计方案存在可靠性、能耗及造价问题，使得无人车进入大众遥不可及。正基于此，如何设计一个无人驾驶系统，既满足应用实时性，高可靠性的计算需求，又要让硬件成本和能耗要尽可能小，是一个迫切需要解决的问题。因此本文针对无人驾驶汽车电子系统进行精确建模，抽象为异构分布式系统，在此基础上提出了一个在满足实时性和可靠性前提下，优化硬件成本和能耗的算法。另外针对无人驾驶汽车计算节点可能会出现宕机、随机硬件错误，通过对计算任务进行复制备份，是目前提高系统可靠性的主流方法。为了提高无人驾驶的可靠性，本文对现有研究的容错调度算法进行了总结和改进，提出了一个 PECORG 算法，改进后不但可以满足更高的可靠性目标，而且应用响应时间和能耗更低。

1.2 国内外研究现状

1.2.1 无人驾驶汽车

无人驾驶汽车目前已经可以上路自动驾驶了，如谷歌、特斯拉、百度等公司的无人驾驶汽车。如特斯拉在 2016 年 10 月，就发布了 Autopilot2.0 系统，实现了完全自动驾驶的商业化应用。百度在 2018 年发布了“阿波罗”无人驾驶汽车，获得北京市首批自动驾驶测试试验临时号牌[4]，以进行实际的无人驾驶道路测试。但无人驾驶汽车还面临着诸多问题。一是汽车的可靠性问题，包括硬件可靠性和软件可靠性。硬件可靠性问题是指计算节点宕机或者处理器的随机硬件错误等。软件可靠性问题是指软件设计时可能存在的错误。另外无人驾驶里采用了相当多的机器学习等技术来做物体识别、行为决策等任务，但由于深度学习模型是个黑盒模型，以及可能存在的训练不完全的情况，一旦汽车遇到陌生情形，可能会做出不正确的决策。二是目前无人驾驶汽车成本高昂，包括开发成本、传感器成本。另外无人车里应用的都是一些高性能的处理器，这些处理器也非常昂贵。三是无人车还存在信息安全的风险，无人车比有人驾驶车辆更容易受到黑客攻击。四是无人驾驶汽车面临着法律相关方面的挑战，比如一旦无人驾驶汽车发生事故，如何定责，法律上还存在空白。

无人驾驶的核心技术主要包括：算法创新、系统融合、新型传感技术、高精度地图、高性能处理平台、新一代车内网络、人工智能、视觉计算等。无人驾驶系统时多种技术的集成，在汽车移动端主要运行包括传感，感知和决策等实时应用。因此无人驾驶汽车系统计算任务多种多样，有适合采用 CPU 计算的，也有适合 GPU 计算的，目前无人驾驶计算平台主要采用异构计算。

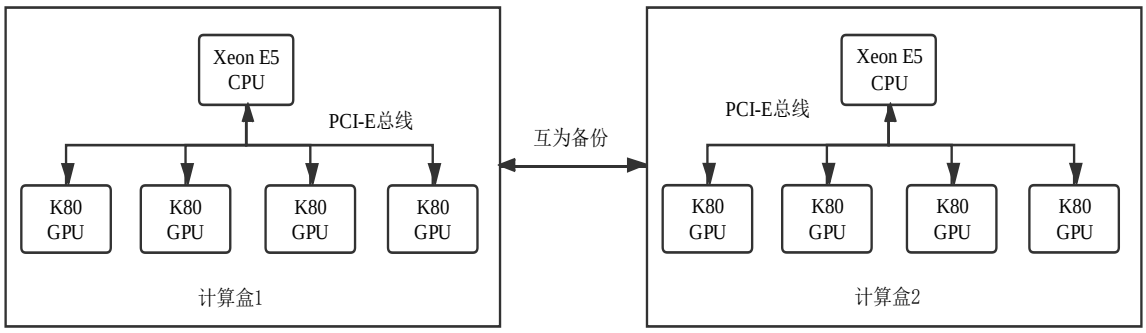

图 1.2 某无人驾驶公司的计算平台解决方案

一个行业领先的某无人驾驶公司提供了如下的计算解决方案[2]，如图 1.2 所示，计算平台由两个计算盒组成。每个计算盒配备一个 INTEL Xeon E5 处理器和 4 到 8 个 NVIDIA K80 GPU 加速器，它们彼此使用 PCI-E 总线连接。CPU 运算峰值速度 400 帧每秒，功率需求达到了 400W。每个 GPU 运算峰值速度可达

8TOP/S，功率需求是 300W。整个系统能够提供 64.5TOP/s 峰值运算能力，总功率可达 3000W。为了保证可靠性，两个计算盒执行完全相同的任务。一旦第一个计算盒失效，第二个计算盒将立即接管。如果两个计算盒都在峰值下运行，总功率高达 5000W，并将产生大量热量。其中每个计算盒高达 2 万至 3 万美元。

在汽车电子系统车身内分布着 100 多个异构处理器，通过 CAN、Flexray 等总线互联，因此我们可以将车内电子系统抽象为一个异构分布式系统。当一个任务在一个处理器上执行完成后，发送消息给所有后继任务，这些后继任务可能分布在不同的处理器上。以线控刹车为例，包括 1 个传感器，CAN 总线、多个处理器以及执行器。当驾驶人踩下刹车时，其中一个处理器通过传感器感知到相关数据，然后将信息通过 CAN、Flexray 总线传输给其他处理器，再由相关处理器控制执行器完成刹车过程。在这个过程中，刹车由多个子任务组成，每个子任务之间具有优先级约束关系，因此对于分布式功能应用，通常都将其抽象为有向无环图(Directed Acyclic Graph, DAG)模型[5]。

1.2.2 汽车功能安全和可靠性

随着人们对汽车性能与体验的追求，汽车上各类高端应用日益增多。与此同时，汽车电子系统发生系统性失效、随机硬件失效、计算节点宕机等危险事件发生的可能性也随之提升。因此汽车安全是汽车工业长久以来的不懈追求。无人驾驶汽车的安全相比普通汽车更加受到人们的重视。无人驾驶汽车上搭载的雷达、摄像头、红外等传感器感知与检测汽车周围环境，然后将数据传递到计算单元进行判断和处理，然后执行相应的操作。尽管人们不断努力在提升汽车安全层次，但仍不可避免汽车偶然性的失效发生。

由于无人驾驶汽车模块众多，功能复杂，处理器处于高性能状态运行会产生大量热量，导致系统部件温度升高，从而影响部件的可靠性，造成瞬态故障。2011 年，专门针对汽车电子系统的功能安全标准 ISO26262 正式发布，并成为国际标准，适用于从传统汽车到现在的新能源汽车[6-7]。

表 1.1 ISO26262 标准对严重性、暴露率、可控性的定义[5]

等级	严重性	等级	暴露率	等级	可控性
S0	无伤害	E0	不可思议概率	C0	普遍可控
S1	不威胁生命的伤害	E1	非常低概率	C1	简单可控
S2	威胁生命的伤害	E2	低概率	C2	通常可控
S3	致命伤害	E3	中概率	C3	难于控制
		E4	高概率		

ISO26262 标准包括功能安全管理，系统开发设计，硬件设计、软件设计等 10 个部分，提供了一个完整的汽车安全生命周期和风险等级的具体评估方法。

ISO26262 结合了严重性、暴露率以及可控性三大安全属性，如表 1.1 所示。

其中严重性表示发生安全事故的伤害严重程度，从低到高分为 S0、S1、S2、S3 四个安全等级；暴露率定义了不同故障发生的概率，分为 E0、E1、E2、E3、E4 四个安全等级；可控性是指驾驶员对故障的可控性，分为 C0、C1、C2、C3 四个等级。ISO26262 同时也指出在汽车设计阶段应该采取有效措施消除这些风险提高产品可靠性。暴露率与可靠性成负相关（即可靠性=1-暴露率）。

保障汽车功能的可靠性目标，优化汽车电子系统应用的执行时间、资源等，这一问题实质上属于对任务的调度问题[4]。然而实际情况下，执行时间，可靠性，消耗的资源这些因素相互影响，分布式汽车功能可靠性目标调度问题属于 NP 难问题。目前在保障可靠性目标的任务调度研究中，一般采用预分配技术，通过将整个应用的可靠性目标，转换为每个任务的可靠性目标，然后在调度每个任务时，选择可以满足任务可靠性目标的处理器执行。

无人驾驶系统是一个高可靠性系统，要保证高可靠性，一个简单的方法就是进行备份冗余，从而实现容错。2018 年 12 月份新发布的 ISO26262 标准相比第一个版本增加了容错的内容。如果无人车中的某个计算节点出现错误，而在另一个节点上运行着同样的计算任务，就可以通过备份任务来完成相应的功能，避免出现严重后果。出错的计算节点再通过重启等手段恢复运行。容错机制主要是对任务复制成多个版本，这种技术称为主/副版本技术（Primary Backup Copy）。

目前的国内外研究中，对汽车功能安全研究一般与能耗和硬件成本等进行综合研究。比如满足可靠性目标前提下优化响应时间[5]，或者能量约束下的可靠性优化[3]，但对硬件成本优化的研究相对较少。在满足实时和可靠性目标前提下优化硬件成本和能耗的研究更是没有。

1.3 主要贡献

本文主要工作和贡献总结如下：

1. 总结分析了无人驾驶汽车集移动端、边缘节点和云端一体的计算结构，介绍无人驾驶汽车领域的一些关键算法。并分类讨论了目前无人驾驶汽车移动端计算平台，呈现出异构多核的特点。
2. 本文提出了一个针对无人驾驶汽车场景下安全关键应用的硬件成本和能耗优化算法 HCECO(Hardware Cost and Energy Consumption Optimization Algorithm)，旨在早期设计阶段，在满足安全关键应用的实时约束和可靠性约束前提下，对硬件成本和能耗进行优化。
3. 为了提高无人驾驶汽车的计算可靠性，针对现有的容错可靠性目标保障调度算法的不足，本文提出了一个可靠性保障下的性能与能耗优化算法

PECORG(Performance and energy consumption optimization algorithm under reliability guarantee)。改进后相比现有的方法，不但可以满足更高的可靠性目标，而且应用响应时间和能耗更低。

1.4 本文组织结构

文章安排如下：

第 1 章绪论概述本文的研究背景，无人驾驶的研究现状，汽车功能安全。最后叙述了本文的主要研究工作以及本文组织结构。

第 2 章回顾无人驾驶汽车的计算结构、计算平台等相关研究，对汽车内电子系统进行建模，并总结了汽车电子系统内硬件成本优化，能耗优化，容错下的可靠性保障，基因算法和模拟退火算法等相关方面的研究。

第 3 章介绍了本文提出的集移动端，边缘节点和云端一体的无人驾驶计算结构，然后从传感、感知、决策、边缘节点、云服务这五个方面系统阐述。

第 4 章针对无人驾驶汽车遇到的硬件成本与能耗问题，本文提出了一个硬件成本和能耗优化算法，首先建立应用模型，可靠性模型，硬件成本和能耗模型，再介绍本文提出的 HCECO 算法详细过程，最后介绍实验部分。

第 5 章介绍本文提出的可靠性保障下的性能与能耗优化算法，首先建立容错机制下的应用模型，可靠性模型，能耗模型，然后分析已有的算法的不足，再介绍本文提出的改进算法，最后介绍实验部分。

最后是结论与展望，对本文的工作及创新点进行最后的总结，并对未来的研究工作进行展望。

第2章 相关研究

本章先介绍无人驾驶的计算结构及其计算平台相关研究，总结无人驾驶汽车计算任务种类多，计算平台呈现分布式异构多核的特点。然后在此基础上将汽车电子系统抽象建模成异构分布式嵌入式系统，再介绍异构分布式系统上关于任务调度、可靠性、能耗和硬件成本优化相关方面的研究以及基因算法和模拟退火算法。

2.1 无人驾驶计算结构及其计算平台

2.1.1 计算结构

文献[8]中最早提出了一个无人驾驶计算结构，它主要分为传感、基于外部数据的建模、感知、执行等部分。如图 2.1 所示。它将无人驾驶分为定位、外部数据、环境感知和自我感知、完成使命等几个主要部分。

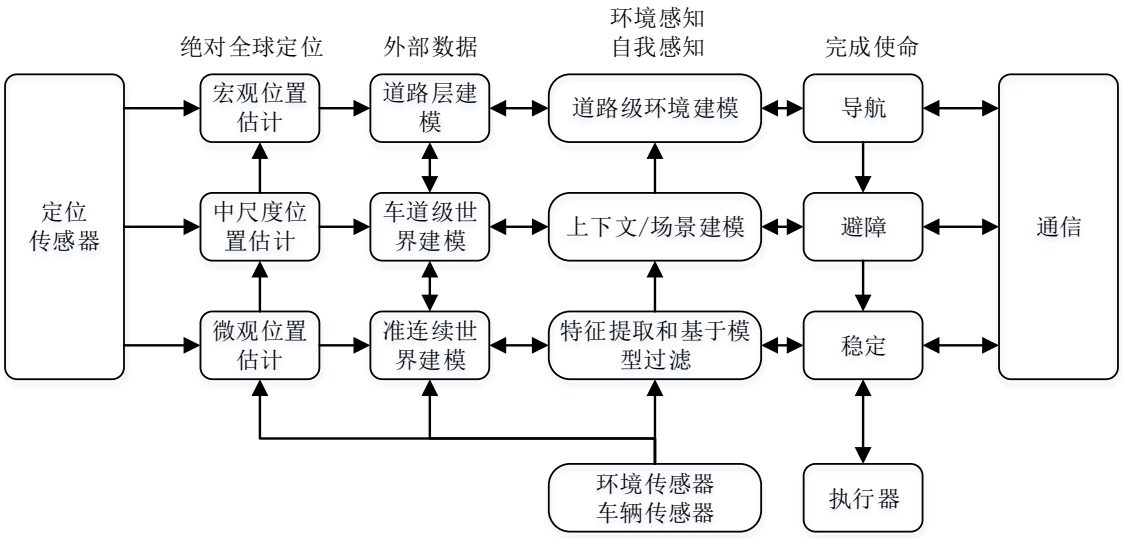

图 2.1 早期的无人驾驶系统结构

文献[9]中提出了一个无人驾驶计算结构，它主要分为有感知的智能驾驶和车辆平台这两个部分。如图 2.2 所示，但是该结构仅限于汽车内部系统结构，对汽车内部计算任务没有分层，并且缺乏对云平台的描述。

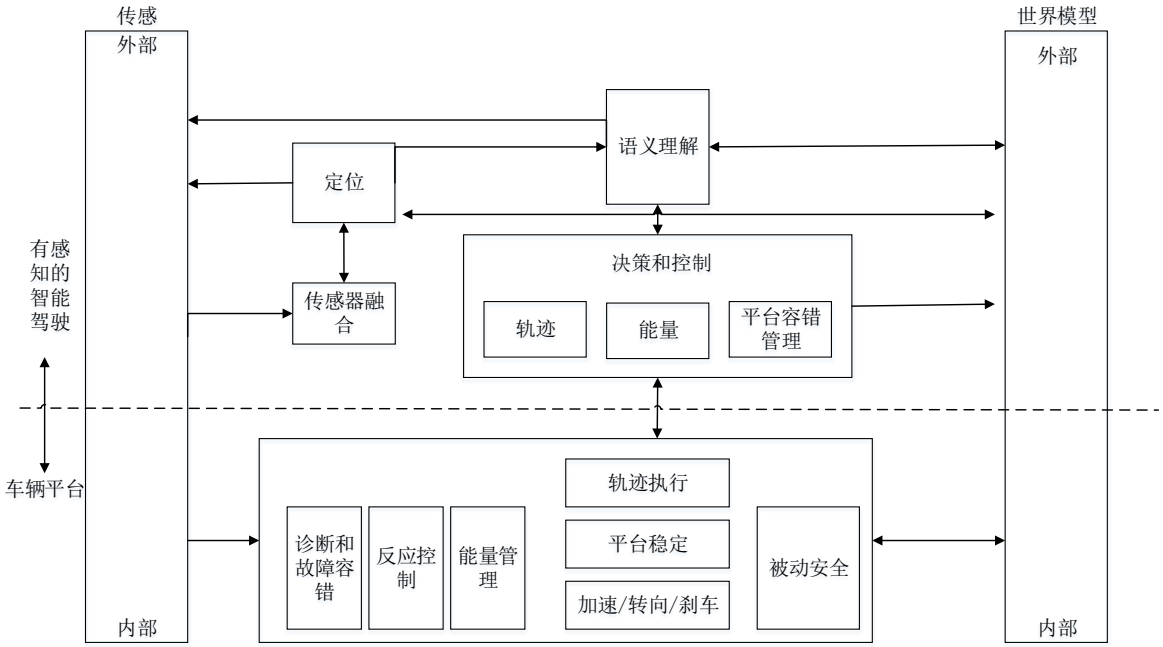

图 2.2 无人驾驶功能系统结构

在文献[2]中提出了一个比较全面的无人驾驶系统架构图，它将系统分为两个部分，客户端和云端。其中客户端包括机器人操作系统和硬件平台，运行包括面向传感、感知和决策等关键步骤算法。云端包括数据存储、模拟、高精度地图以及深度学习模型训练。

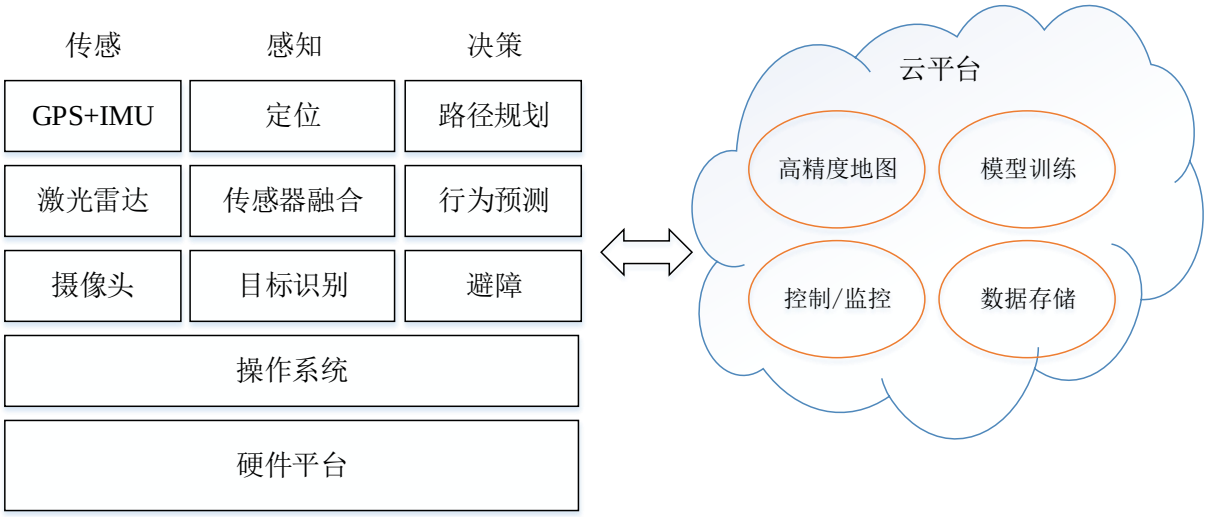

图 2.3 文献[2]中的无人驾驶计算结构

2.1.2 计算平台

本节我们将先讨论一个简化的无人驾驶汽车移动端计算平台的解决方案，然后分类总结目前无人驾驶计算的处理器平台，并对其中一个典型平台进行进一步分析。

S.Liu 和 J.Tang 等人[10]在 ARM SoC 上实现了一个简化的无人驾驶系统，计算结构如图 2.4 所示，这个 ARM SoC 拥有四个核，由 CPU,GPU,DSP,FPGA 组成。试验显示，在峰值情况下能耗仅 15w，在不损失任何位置信息的情况下达到每小时 8 公里的速度。ARM SoC 之所以能提供这样的性能，是因为充分利用了硬件系统的异构计算资源，为每一个不同无人驾驶子任务寻找最匹配的的计算单元，可以达到最优化的性能和能源效率。但是这样的设计思路仍然存在一个缺点：我们不可能为所有的子任务找到适配的计算单元。比如，目标跟踪、变更车道预测等计算较为密集的子任务。

图 2.4 基于 SoC 的简化无人驾驶系统

目前无人驾驶现有的底层计算平台解决方案，可分为如下几类：

基于 GPU 的解决方案：NVIDIA 的 DRIVE PX2 平台是一个典型的基于 GPU 的无人驾驶解决方案，如图 2.5 所示。每个 PX2 由两个 Tegra SoC 和两个 Pascal GPU 图形处理器组成，多个 Drive PX 2 平台的并行使用可以实现完全的自主驾驶。为了提高吞吐量，每个 Tegra SoC 使用 PCI-E 总线与 GPU 相连，其总带宽为 4GB/s,两个 GPU 通过千兆以太网相连，数据传输速度可达 70GB/s，每个 PX2 每秒可执行 24 万亿次深度学习计算。

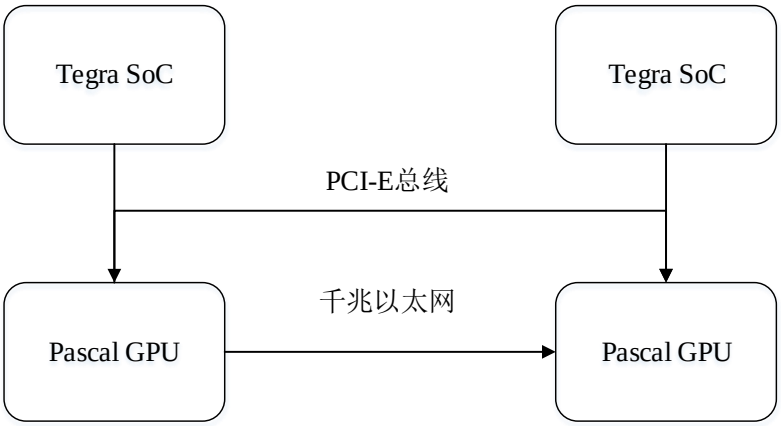

图 2.5 NVIDIA DRIVE PX2 解决方案

下面介绍 NVIDIA 官方给出的基于 PX2 平台的无人驾驶数据采集方案[11]，模型训练方案。数据采集示意图如图 2.6 所示：

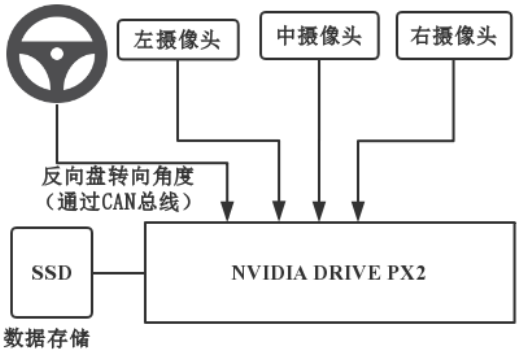

图 2.6 PX2 无人驾驶数据采集方案

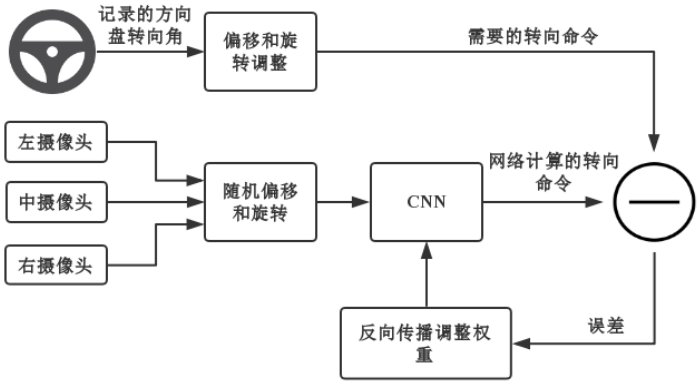

图 2.7 PX2 无人驾驶数据训练方案

在数据采集车挡风玻璃后安装了三个摄像头，用来实时采集车辆前方路况，摄像机拍摄的时间戳视频和驾驶员的方向盘的转向角被同时采集。数据采集完成后，通过如图 2.7 所示的方案对数据进行训练。只有来自人类驾驶员的数据训练是不够的，神经网络必须学会如何从错误中恢复过来，否则汽车会慢慢偏离道路。因此，在训练数据中增加了额外的图像，让汽车从不同方向偏离道路中心或按道路方向旋转。在训练过程中，摄像头采集到的图像被输入给卷积神经网络，然后由神经网络计算一个建议的转向命令。将该转向命令与实际采集到的命令进行比较，从而计算误差，根据误差，采用反向传播法调整卷积神经网络的权重，使卷积神经网络的输出更接近实际所需的输出。

基于 DSP 的解决方案:DSP 采用哈佛总线设计，数据总线和地址总线分开，取出指令和执行指令可以并行，这大大提高了微处理器速度。德州仪器提供了采用 DSP 的高级驾驶员辅助系统（Advanced Driver Assistance Systems, ADAS）解决方案，包括 TDA2x,TDA2Eco 和 TDA3x 等系列产品。其中 TDA2Hx 采用一个含有视觉加速器内核的 DSP 处理器（c66x 600MHz），同时搭载 ARM

Cortex-A15 MP Core™处理器，ARM Cortex-M4 处理器和一个 GPU 用来加速处理 3D 图像。

基于 FPGA 的解决方案：FPGA 硬件配置最灵活，具有低能耗，高性能及硬件可编程等特性。FPGA 相对于 CPU 和 GPU 有明显的的性能和能耗优势。FPGA 可定制传感器接口和知识产权(IP)内核，以支持任何汽车网络标准的连接。能够在硬件和软件中实现定制算法，进行视频和图像处理。Intel Altera 公司推出的 Cyclone V SoC 是一个基于 FPGA 的无人驾驶解决方案，现已应用在奥迪无人车产品中[2]。Cyclone V SoC 装备有 ARM Cortex™-A9 MPCore™处理器、Cyclone V FPGA 以及精度可调的 DSP 处理器。其 Cyclone V GT FPGA 可提供高达 6.144Gbps 的收发器功能,且采用了 28-nm 低功耗 (LP) 工艺技术进行开发，为需要 5G 收发器的应用提供了最低功耗解决方案。

基于 ASIC 的解决方案：ASIC（Application Specific Integrated Circuit）是一种为专门应用而设计的集成芯片。它是应特定用户要求和特定电子系统的需要而设计、制造的集成电路。ASIC 的特点是面向特定应用的需求，ASIC 在批量生产时与通用集成电路芯片相比具有体积更小、功耗更低、可靠性更高、性能更高、保密性更强、成本低等优点。Mobileye 是一家基于 ASIC 的无人驾驶解决方案提供商。其 EyeQ5 SoC 装备有多个异构的全编程加速器（Vector Microcode Processors、Multithreaded Processing Cluster、Programmable Macro Array），同时实现了两个 PCI-E 端口以支持多处理器间通信，具备高计算能力，低功耗等特点。EyeQ 目前已经被 27 个汽车厂商所采用，可实现 L5 级别的完全自动化驾驶汽车将在 2020 年上路。

表 2.1 计算平台总结

方案	典型平台	包含的处理器	特点
GPU	DRIVE PX2	Tegra SoC Pascal GPU	计算能力强大
DSP	TDA2x	ARM Cortex-A15 ARM Cortex-M4 DSP c66x	3D 图像 支持多达 10 个摄像头输入
FPGA	Cyclone V SoC	ARM Cortex-A9 Cyclone V FPGA DSP	低功耗 高速收发器
ASIC	EyeQ5 SoC	CPU 可编程加速器 视觉加速器	低功耗 硬件安全

上述四个方案的总结对比分析结果如表 2.1 所示，可以看出每个平台至少使用了两种类型处理器，因此无人驾驶系统主要采用异构多处理器作为硬件平台，将多个处理器通过高速网络相连，从而获得巨大的计算能力，因为异构多处理器可以充分利用硬件系统的异构计算资源。如表 2.2 所示，相同的任务，在不同的处理器上的计算时间和消耗的能量都具有显著性差异，不同任务在相同处理器上执行也具有显著性差异[10]。由于无人驾驶系统具有多种计算任务类型，如果所有的任务都在同种处理器上运行，很难兼顾所有的性能，因此在无人驾驶中应用异构处理器来作为计算平台是一个不错的选择。

表 2.2 异构处理器的性能表现

处理器		卷积	特征提取
CPU	计算时间（ms）	8	20
	能耗(MJ)	20	50
GPU	计算时间（ms）	2	10
	能耗(MJ)	4.5	22.5
DSP	计算时间（ms）	5	4
	能耗(MJ)	7.5	6

2.2 汽车电子系统资源优化

2.2.1 汽车电子系统模型抽象

在汽车电子系统内部分布着 100 多个处理器，这些处理器也被称为电子控制单元(Electronic Control Unit, ECU)，多个异构处理器通过 CAN、Flexray 等高速网络相连。CAN 是一种半双工、静态优先级和非抢占式的通信总线，它是基于事件触发的通信方式，非常适合于汽车内分布式系统。而 Flexray 总线传输速度更高，它具有两个通道来进行通信，保障通信的可靠性，它既支持基于时间触发的通信方式，也支持基于事件触发的通信方式[12]。汽车电子系统结构如图 2.8 所示，每个处理器可以连接着传感器或执行器，每个处理器包含 CPU、RAM、非易失性存储器以及一个网卡。目前针对汽车电子系统内资源优化，一般把它抽象为异构分布式系统，然后在此基础上进行研究。

汽车电子系统功能日益增多，功能日趋复杂，这些功能由很多可并行的小任务组成，因此我们把汽车上功能可以看成一个并行应用. 这些应用通常可拆分成具有优先约束的多个子任务，一个具有多个优先约束的子任务组成的并行应用通常用一个有向无环图（DAG）来表示。图 2.9 是一个简单的 DAG 应用示例图。图中的节点表示任务，边表示任务之间的通信代价，当一个任务在一个处理器上

执行完后，需要把数据传递到执行后继任务的处理器上，将数据传递到后继任务处理器上所需的时间即为通信代价。如果后继任务就是这个处理器，则这个通信代价为 0。

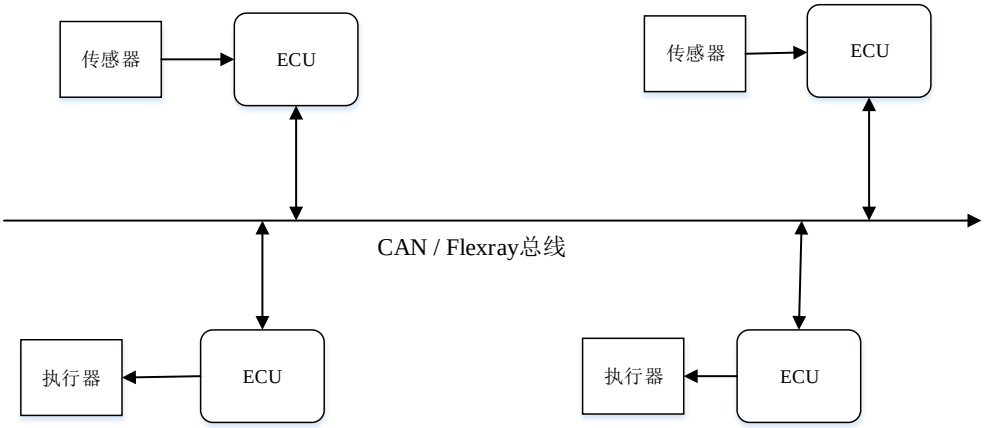

图 2.8 汽车电子系统内部结构

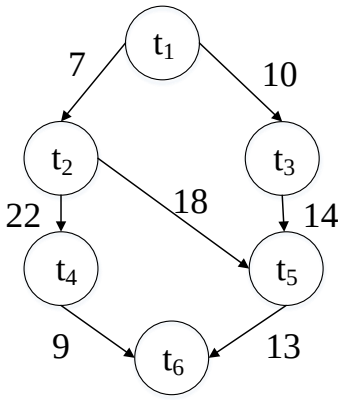

图 2.9 一个简单的 DAG 应用

2.2.2 响应时间和可靠性优化研究

在汽车异构分布式系统中，把这些 DAG 应用的子任务调度到具体处理器上执行，在满足任务之间优先约束条件下，使得资源消耗代价最小是一个经典的 NP 难问题。Hulak [13]等人针对异构多处理器静态调度提出了一个异构最早结束时间算法（HEFT）和 CPOP 算法。HEFT 算法流程如算法 2.1 所示。HEFT 算法使用升秩作为任务的优先级，如公式(2.1)所示，升秩定义为任务在所有处理器上的平均计算时间加上和前驱任务任务最大通信时间。通过对每个任务的升秩排序，每次在满足优先约束条件下，将升秩最大任务分配给结束时间最早的处理器上执行，基于插入的原则使得该应用的最早结束时间最小。

$$\mathrm{rank_u}(t_i)=\overline{w_i}+\max_{t_j\in succ(t_i)}(\overline{m_{i,j}}+\mathrm{rank_u}(t_j)) \qquad (2.1)$$

其中$\overline{w_i}$是任务t_i在处理器集合上的平均计算代价，$\overline{m_{i,j}}$是任务t_i到任务t_j的通信代价，$succ(t_i)$是任务t_i的后继任务集合。通过对每个任务的$\mathrm{rank_u}$进行降序排列，

每次选择$rank_u$最大的任务优先调度。注意只有当t_i的所有先继任务执行完成后，t_i才会准备执行。

算法 2.1: HEFT 算法

Input：应用 DAG,处理器集合 U
Output：调度结果
1:计算 DAG 中每个任务的向上排序值$rank_u$;
2:根据任务优先级$rank_u$，对任务排序后放到优先级队列 rank_task_list 中;
3:while 任务队列 rank_task_list 中有未调度任务 do
4: t_i= rank_task_list.out();
5: for 处理器集合中的每一个处理器u_k do
6: 计算任务t_i在处理器u_k上的最早结束时间;
7: end for
8: 将任务t_i分配到具有最小的最早结束时间的处理器上执行;
9:end while

CPOP 算法使用有向无环图（DAG）中的关键路径上的任务作为关键任务，关键路径长度定义为关键任务的计算时间加上关键任务之间的通信时间，在处理器选择阶段，通过将关键任务分配到使关键路径长度最短的一个处理器上，其他任务使用升秩加降秩作为任务的优先级，分配到最早结束的处理器上，从而减少应用执行时间。

S. Ding 和 J. Wu[14]等人提出了一个自适应参数的启发式基因算法来解决异构多处理器静态调度问题，将调度方案编码成染色体，使用 HEFT 算法得出一个初始解用来加快收敛，通过选择优良个体，模拟生物种群的交叉和变异，不断迭代从而逼近最优解，从而减少应用的响应时间。

可靠性目标在一些工业安全标准中广泛使用，比如汽车软件系统标准 ISO 26262，工业嵌入式安全标准 IEC61508[15]。如果一个应用的可靠性目标可以被满足，则根据这个安全标准，则这个应用就是可靠的。广泛接受的可靠性模型在论文[16]中提出，其中硬件的瞬态错误率用一个常量λ来表示，DAG 应用的单位时间内的可靠性服从泊松分布，整个应用的可靠性等于所有任务的可靠性的乘积。G. Xie 和 Y. Chen 在文献[17]中提出了一个可靠性目标预分配方法，使用公式(2.2)将整个应用的可靠性目标转化为每个子任务要满足的可靠性指标。用升秩表示每个任务的优先级，依次选取优先级最大的任务，然后将它调度到合适的处理器上执行，使得每个任务的可靠性指标得到满足，其中可靠性模型采用泊松分布来描述。

$$R_{goal}(t_j)=\frac{R_{goal}(G)}{\prod_{x=1}^{j-1} R(t_x,u_{proc(x)})\times\prod_{y=j+1}^{|T|} R_{max}(t_y)} \qquad (2.2)$$

其中$R_{goal}(t_j)$表示任务t_j要满足的可靠性，$R(t_x, u_{proc(x)})$表示任务t_x的最终实际可靠性，$R_{max}(t_y)$表示任务t_y的最大可靠性，通过遍历任务t_y在每个处理器上的可靠性得出。

2.2.3 能耗和硬件成本优化研究

在嵌入式系统中，资源通常是有限的，所以在设计阶段就应该把计算平台的设计优化好[18-19]。能耗优化是近年来绿色计算领域研究热门课题。降低能耗技术主要包括两种：动态电源管理(Dynamic Power Management, DPM)和动态电压频率调节(Dynamic Voltage and Frequency Scaling, DVFS)。动态电源管理是指在系统组件不工作的时候将其关闭或切换为休眠模式。动态电压频率调节是指通过调节处理器的电压和频率来优化能耗。

文献[20]中通过结合 DPM 和 DVFS 技术来降低应用的能耗。B.Zhao 和 H. Aydin 等人[21]提出一个方法，在时间和能源功耗限制条件下，通过 DVFS 技术运行时调整处理器频率，在硬能耗限制条件下，使可靠性达到最高。文献[22]中在截止时间约束下，通过向上和向下排序的方法，使用 DVFS 技术降低能耗。文献[23]中通过在满足实时的约束条件下，通过不断的关闭不必要的处理器来减少能耗。这些研究都不是在响应时间和可靠性目标这两个约束下优化能耗，不适宜应用在无人驾驶这类安全关键的应用中。

由于频繁使用 DVFS 技术改变处理器电压和频率，一是会提高处理器的瞬态硬件错误率，二是会降低处理器的寿命。因此在一些研究中不建议频繁改变处理器电压和频率，而是选择一个合适的频率长期运行[24]。同时能耗和可靠性之间存在着一个约束的关系。因此本文的能耗优化，是在固定处理器频率下进行的，在处理器不工作时，将其关闭或休眠从而降低能耗。

在嵌入式系统中，为了适应大批量生产，成本要求是非常严格的。文献[25]中针对异构嵌入式系统，提出了 PA（Path Assign）和 TA（Tree Assign）算法，通过基于概率性的策略解决了异构嵌入式系统中具有硬实时和软实时约束的硬件成本优化问题。文献[26]针对异构云计算系统，提出了自动调节计算实例的方法优化具有截止时间约束的 DAG 应用程序的硬件成本。文献[27]中分别介绍了整数线性规划（MILP）和分开与整合的启发式算法以及模拟退火（Simulate Anneal, SA）的智能优化算法，解决了异构分布式嵌入式系统中满足并行应用的实时和安全需求的同时优化硬件成本的问题，但它专注于安全性要求而非可靠性要求。

目前在一些高端汽车内，已经分布着 100 多个 ECU，而每个 ECU 的成本大约在 25-110 美元[28]，因此通过减少使用 ECU 个数来减少硬件成本是一个重要的手段。为了降低异构嵌入式系统硬件成本，G. Xie 在文献[28]等人提出了一

些算法（IHCO、EHCO、EEHCO、SEEHCO），在产品的设计阶段优化异构嵌入式系统硬件成本，EHCO 算法在满足硬实时和可靠性条件下，通过不断移除高硬件成本的处理器，让硬件成本得到优化。EEHCO 算法在 EHCO 算法基础上通过增强任务可靠性提高了应用的可靠性。SEEHCO 算法减少了 EEHCO 的时间复杂度，减少了算法执行时间。但是这些算法都为贪心算法，可能会陷入局部最优解，而且该文章没有考虑能耗，仅将硬件成本最优作为衡量标准。

现有的这些能耗和硬件成本优化研究，都只是考虑了实时、可靠性、硬件成本、能耗这些参数中的某几个，去进行优化，没有全部考虑进去。而在真实的无人驾驶汽车设计阶段，我们往往需要将这些参数全部考虑。

2.2.4 容错机制下可靠性保障研究

无人驾驶系统中，对计算的可靠性要求是非常高的，系统中某个计算节点宕机，导致系统部分功能缺失，可能会造成严重后果。通过对计算任务进行备容错，被广泛用于提升系统可靠性。对任务备份就是每个任务都有 0 个、1 个或多个备份任务。尽管通过任务备份可以大大提高应用的可靠性，但是在实践中应用的可靠性仍然不可能等于 100%。因此在实践中，如果某个应用满足可靠性目标（根据相应的标准），我们就认为该应用是可靠的。

任务复制备份可以分为主动复制和被动复制[29]。如果副版本任务不需要主版本任务出错就可以执行，只要有一个副本任务执行成功，则整个任务就执行成功，我们称之为主动复制。被动复制是指只有当主任务出错时，备份任务才激活成为新的主任务继续执行。被动复制存在的问题是需要主任务出错，才启动副版本任务执行，这个中间可能会存在一些时延，对于一些硬实时的场景不适用。同样主动复制存在的问题是同时执行多个副本任务会浪费资源。

在容错机制下，根据任务副版本数目是否固定，可以分为固定任务副版本数目复制和动态任务副版本数目复制。论文[30-31]中提出了一系列固定任务副版本数目的主动复制调度算法。固定的将任务复制 n 个副版本，可以实现同时容忍多个错误，从而实现较高的可靠性。但是这种盲目复制会造成任务过度冗余，付出了更多的资源和性能代价。不固定任务副版本数目，根据具体任务，复制准确副版本数目，可以减少冗余。下面我们主要介绍动态副版本数目的相关研究。

在可靠性保障相关研究中，一般通过预分配方法将应用的可靠性目标转换为单个任务的可靠性目标，然后保证每个任务的可靠性目标得到满足，最后实现整个应用的可靠性目标得到满足。一个并行应用如果想要提高可靠性，就有可能消耗更多的资源。Zhao L 和 Ren Y 等人[32]提出了一个 MaxRe 算法在满足可靠性的目标下，优化资源消耗，其中任务的副本数目是不固定的。通过将整个应用的可靠性目标开 n 次方根，转化为每个任务要满足的可靠性目标，

$R_{goal}(t_j)=\sqrt[|T|]{R_{goal}(G)}$，其中|T|为任务的总个数。每个任务的可靠性目标都是相等的。后来论文[33]提出的 RR 算法对 MaxRe 算法进行了改进，考虑了已分配任务对后续任务可靠性目标的影响,它的每个任务可靠性目标采用如下公式计算：

$$R_{goal}(t_j)=\sqrt[|T|-j+1]{R_{goal}(G)\Big/\prod_{x=1}^{j-1}R(t_x)} \tag{2.3}$$

这个可靠性分配方法由于考虑了前驱任务对当前任务的影响，能高效减少资源冗余。但是没有考虑后继任务的影响。

Xie G 和 Zeng G 等人[34]提出了一个 HRRM 算法，它不仅考虑了前驱任务对当前任务的影响，而且考虑了后继任务的对当前任务的影响，它将每个后继任务的可靠性预分配为任务的可靠性上界值，如下公式 2.4 所示：

$$R_{goal}(t_j)=\frac{R_{goal}(G)}{\prod_{x=1}^{j-1}R(t_x)\times\prod_{y=j+1}^{|T|}\sqrt[|T|]{R_{goal}(G)}} \tag{2.4}$$

针对 HRRM 对先后任务可靠性分配不平衡的问题，Xie G 和 Zhetao L 等人[35]提出了一个 GMFRP 算法，该算法采用几何平均值作为任务的可靠性预分配值，相比 HRRM 算法，GMFRP 算法的响应时间更短，但是 GMFRP 算法在高可靠性目标的情况下，不能保证可靠性目标始终得到满足。

2.3 基因算法与模拟退火算法

要设计一个分布式无人驾驶计算平台，满足无人驾驶应用的实时性，可靠性，低功耗和低成本的要求，是一个典型的多目标优化问题。基因算法被广泛应用于解决多目标优化问题，它模仿生物进化优胜劣汰的原理。将问题的求解方案编码成染色体，通过染色体交叉和变异，在足够次数的种群迭代之后，接近最优的染色体将被选择出来作为最终的结果。论文[14]运用自适应参数调节的基因算法解决了异构多处理器的静态调度问题，论文[36]中通过运用基因算法解决了云计算环境下的任务调度的负载均衡问题。其他的一些启发式算法也用于解决多目标优化问题，论文[37]中运用猫群算法和模拟退火算法解决了弹性云环境下的多目标任务调度问题。

基因算法被广泛应用于解决多目标优化问题，它模仿生物进化优胜劣汰的原理。通过自然选择、染色体交叉和变异，个体的适应能力不断提升。在种群迭代一定次数后，最优的个体将被选择出来作为最终的结果。由于基因算法有这强大的全局搜索能力，但是局部搜索能力不强，而模拟退火算法有着很强的局部搜索能力[38]。因此许多研究[39-42]都将基因算法和模拟退火算法结合来解决实际问题。如论文[39]就采用并行的模拟退火基因算法来解决任务调度问题。因此本文结

合了模拟退火算法来增强基因算法的局部搜索能力。模拟退火算法是一种基于蒙特卡罗迭代的优化算法。这个想法来自于固体物质退火的过程。模拟退火算法是从一个相对较高的温度开始进行解搜索，随着温度的不断下降，根据相应的概率接收准则，在解空间随机寻找目标函数的最优解。

2.4 小结

本章主要回顾了无人驾驶计算结构和计算平台的相关研究，其中计算平台呈现出分布式异构多核的特点。然后总结了目前在汽车电子系统上资源优化的相关研究。首先我们针对汽车电子系统进行抽象建模，将汽车内电子系统抽象为异构分布式系统，将汽车功能应用抽象为 DAG 应用。最后主要回顾了非容错下应用响应时间和可靠性优化、硬件成本和能耗优化、容错机制下的可靠性保障的相关研究以及基因算法与模拟退火算法。

第 3 章　无人驾驶计算结构

随着通信与信息技术的不断发展，无人驾驶领域有了巨大的进步。无人驾驶系统是一个非常庞大且复杂的软硬件系统，它涉及多个技术的集成。目前相关研究中对无人驾驶计算结构总结不全面，因此有必要对无人驾驶的计算结构进行一个全面的总结。

3.1 计算结构

无人驾驶系统根据计算所在实体可分为汽车移动端，边缘节点和云端。汽车移动端包括应用层、操作系统和硬件平台，主要负责运行包括面向传感、感知和决策等关键步骤的实时功能和算法以及人机交互等功能，人机交互主要是指乘客对车辆进行一些控制。边缘节点主要提供一个稳定的网络连接，实时数据处理。云端包括数据存储、自动驾驶模拟、高精度地图绘制，深度学习模型训练，云控制，监控及云诊断等功能。无人驾驶系统架构图如图 3.1 所示。接下来本文将从传感、感知、决策和远程服务四个方面来全面详细介绍无人驾驶的计算结构。

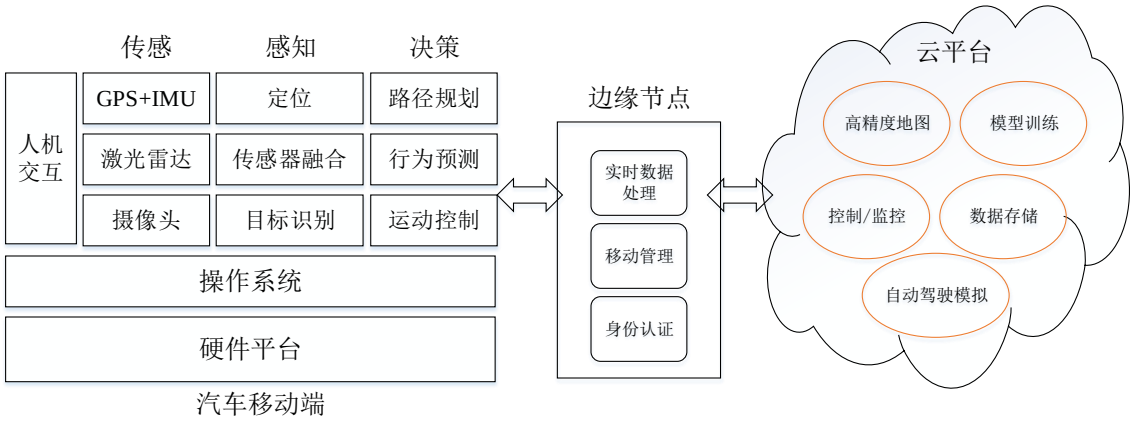

图 3.1 无人驾驶系统计算结构图

3.2 传感技术

在传感层面，无人驾驶汽车通过多种类型的传感器获取外部世界状态数据。现有的无人汽车大多使用 GPS 和惯性测量单元（Inertial Measurement Unit, IMU）提供位置信息，GPS 可以提供高精度位置信息，但更新速度慢，IMU 可以实时提供位置更新，但精度不大。将这两个传感器的数据进行融合，来获得高精度位置信息。通过 GPS 和 IMU 相结合的方式确保汽车能够适应各种情况。比如隧道等场景中，GPS 将不能提供服务，此时只能依靠 IMU 来提供位置信息。

在避障、运动控制领域的传感器，无人车主要采用雷达加摄像头的方案。雷

达主要包括 LIDAR（Light Detection And Ranging）激光雷达、超声波雷达和毫米波雷达等。雷达和摄像头这两种传感器互为补充，使得在某一种传感器出现故障的情况下，提供一定冗余度，确保汽车完成正常功能。

LIDAR 激光雷达是无人驾驶汽车中的一个非常关键的传感器，它主要用来产生高精度地图，以满足避障要求。LIDAR 传感器不断发射激光传感束，每秒可发送数百万光束，然后根据光束返回至传感器的时间测距。它可以推断出与周围任何物体的精确测量值，从而构建一个巨大的 3D 地图，可以真正地实现 3D 世界的可视化。随着 LIDAR 应用的更加广泛并且拥有更高的分辨物体的能力，LIDAR 绘制的地图不仅可以让无人驾驶汽车准确了解自己在世界的哪个位置进而进行导航。还可以识别和跟踪汽车，行人等障碍物。激光雷达相比其他传感器的优点是：

1. 激光雷达可以获得极高的角度、距离和速度分辨率。通常激光雷达的角分辨率不低于 0.1mard 也就是说可以分辨 3km 距离上相距 0.3m 的两个目标；距离分辨率可达 0.1m；速度分辨率能达到 10m/s 以内。如此高的距离、速度分辨率意味着激光雷达可以利用多普勒成像技术获得非常清晰的图像。
2. 抗有源干扰能力强。微波、毫米波雷达容易受到自然界中广泛存在的电磁波的影响，自然界中能对激光雷达起到干扰作用的信号源不多，因此激光雷达抗有源干扰的能力很强。
3. 测距距离大。以 Velodyne 公司的 VLS-128 激光雷达为例，最大范围 300 米。

但是激光雷达存在的缺点是：1、价格过于高昂；2、激光雷达对透明物体和黑色吸光物体无法检测；3、激光雷达产生的深度图为稀疏图，会损失一定的细节。

超声波雷达是通过发射超声波，通过发射和接受反射波之间的时间差计算与物体之间的距离。它的优点是不受光线的影响，比如大雾天气等情况下。毫米波雷达是通过发射波长在 1mm 到 10mm 之间的电磁波，来进行测距。它的优点是价格便宜，体积小。但缺点是会受天气的影响。

摄像头用来做物体识别及物体追踪，运用于车道线检测，交通标志识别等场景，主要运用在车道保持系统、全景泊车、行车记录等系统中。为保障汽车可靠性和安全性，车上一般装有八个摄像头。无人汽车上的多个摄像头以 60Hz 频率同时工作时，将产生 1.8GB/s 的巨额数据量[10]，这些传感器数据都必须在汽车端得到实时处理。 比如特斯拉 Autopilot 2.0 的硬件系统中有 3 个前视摄像头，分别为正常、长焦、广角摄像头， 3 个摄像头可覆盖更远距离和更宽的视野范围，探测精准度还安全性将大大提高。无人驾驶采用摄像头作为传感器的一些难

点如下：

1. 对于极端恶劣天气，或一些极端情况下可能会失效。比如大雾天气，或者突然遇到强光，摄像头会被致盲。
2. 摄像头产生巨大数据量，需要更强大的处理器去计算。
3. 传输高画质视频对网络的要求更加高效，以及带来成本的上升。

3.3 感知系统

在获得传感信息后，感知系统通过传感器数据融合以充分了解无人车所处周围环境以及建立起外部世界模型[43]。

无人驾驶汽车通过摄像头的图像数据，进行物体检测识别和跟踪来感知车辆周围环境，比如车道线检测，行人或车辆检测以及交通标志识别等。早期比较著名的车道线检测如 GOLD（General Obstacle and Lane Detection）算法[44]，它可以处理大多数情况，例如实线和虚线，强阴影和斑马线等。然而，该算法是一个没有 3D 信息的单眼算法，它难以区分白光灯杆和车道。但是通过数据融合模块，借助 SLAM（Simultaneous Localization and Mapping）地图和其他传感器，可以成功地消除这些误报。卷积神经网络（Convolution neural network, CNN）在图像识别领域有着非常优秀的表现以及广泛的应用。因此在无人驾驶领域，卷积神经网络已经广泛应用于目标分类识别，目标检测等领域[45-49]。

卷积神经网络工作原理如下：1、卷积层使用不同的滤波器从输入图像中提取不同特征；2、激励层决定是否启动目标神经元，并对数据进行正则化处理以及防止梯度消失；3、池化层压缩参数，减少训练计算量；4、最后通过一个全连接层连接到分类器，由分类器输出结果。一旦物体被 CNN 识别出来，下一步将自动预测进行物体追踪。卷积神经网络的图像数据采集由汽车移动端完成，模型训练由云端完成，最后将训练好的模型移植到汽车端用来进行目标识别等任务。

物体检测是无人驾驶感知的必不可少部分，它主要用于无人车发现前方障碍物类别和位置。在目标检测领域，广泛使用的是 R-CNN（Recursive-CNN）神经网络，针对 R-CNN 速度慢的缺点，后来又出现了 Fast R-CNN[50]和 Faster R-CNN[51]。Faster R-CNN 可以分为四个步骤：1、对整张图片输入给 CNN，得到特征图（Feature Map）；2、将特征图输入到区域方案网络(Region Proposal Networks, RPN），得到候选框的特征信息；3、对候选框中提取出的特征，使用分类器判别是否属于一个特定类；4、对于属于某一特征的候选框，用回归器进一步调整其位置，从而确定目标在图片中的位置。Faster R-CNN 原理图如图 3.2 所示：

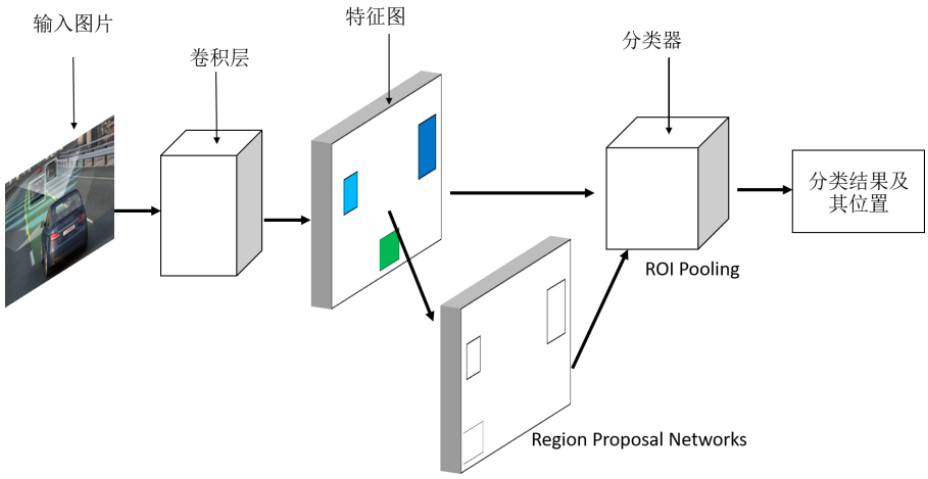

图 3.2 Faster R-CNN 结构原理

仅仅通过同类传感器来让无人驾驶汽车感知周围环境是不安全的，因为无法克服传感器自身的缺点。比如只用摄像头来测距或识别交通标识，在大雾天气就会失效。因此无人车必须通过融合多种传感器的数据来充分感知周围环境，融合多种传感器的系统结构如图 3.3。在数据融合计算领域有两种常用的算法：神经网络和卡尔曼滤波。

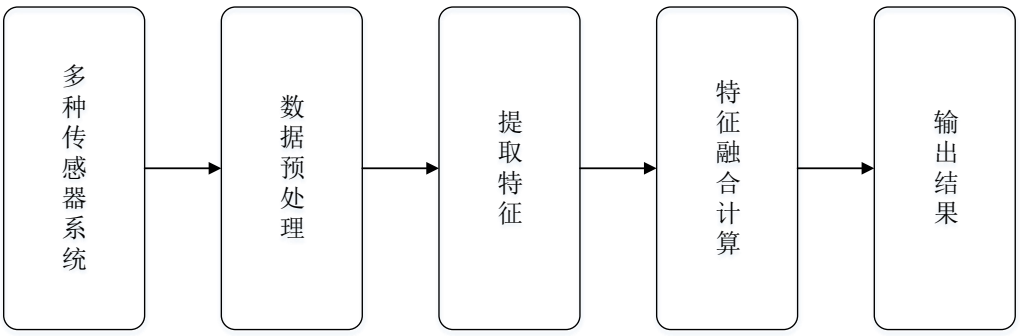

图 3.3 多传感器数据融合过程

神经网络由多层神经元组成，分为输入层，隐含层和输出层。我们可以从每种传感器中提取出必要的特征，把它作为输入数据输入给神经网络的输入层。从而达到数据融合的目的。

卡尔曼滤波（Kalman filtering）是一种利用线性系统状态方程，通过系统的输入输出观测数据，对系统状态进行最优估计的算法[52]。卡尔曼滤波在测量方差已知的情况下能够从一系列存在测量噪声的数据中，感知动态系统的状态。由于它便于计算机编程实现，并能够对现场采集的数据进行实时的更新和处理，因此卡尔曼滤波是目前应用最为广泛的滤波方法，在通信、导航、控制等多个领域得到了很好的应用。

3.4 决策

在决策阶段，无人车需要基于高精度地图进行路径规划、运动控制等操作。

典型的路径规划算法有迪杰斯特拉算法和 A*搜索算法[2]。与普通的地图导航不同，无人驾驶里的路径规划算法需要精确到车道级别。

无人驾驶中最重要、最具挑战的模块就是行为决策。传统的基于规则的驾驶决策系统是一个典型的有限状态机模型，只能采取非常保守的驾驶策略，需要人为设计精妙的规则来应对各种复杂的路况。一旦设计规则有疏忽，后果不堪设想。随着深度学习的兴起，越来越多的公司和研究者把强化学习应用到无人驾驶的行为决策中[53-56]。著名的无人驾驶汽车方案提供商 Mobileye 公司就是其中的典型代表，其设计的车辆模型已经能自如应对一些复杂的交通任务，如双向通道变线、复杂十字路口等场景。

如论文[55]采用深度强化学习的方法，提出了一个无人驾驶场景下的制动控制算法，原理如图 3.4 所示。其中状态由障碍物的相对位置和车辆速度给出，动作空间被定义为制动动作的集合，包括 1）无制动，2）弱制动，3）中等强度制动，4）强烈的制动。通过一个评价函数来为最终结果给出奖惩值，使得车辆尽可能避免碰撞以及使得碰撞后的损失最小。

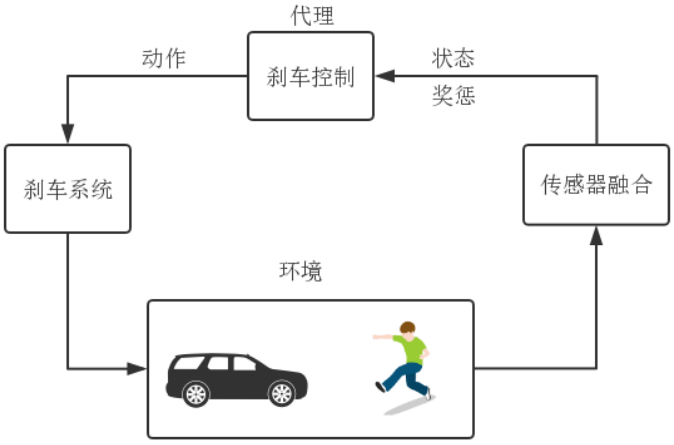

图 3.4 强化学习刹车控制系统

强化学习拥有一个能感知环境的自治代理（Agent），在每个步骤中，代理通过与环境的交互，学习一种策略，以优化长期回报。在每个步骤中，增强学习代理都得到关于其动作性能的评估反馈，从而改进后续动作的性能，Q-Learning 算法是增强学习里的一个经典算法。Q-Learning 算法使用表格来存储每一个状态，和在每个状态下每个动作所拥有的奖惩值。然而当今很多问题都很复杂，状态和动作的组合可以出现指数增长，比如下围棋。如果全用表格来存储它们，恐怕我们的计算机有再大的内存都不够，因此由 DeepMind 在 2013 年引入了深度 Q 神经网络（Deep Q Network）[57]。Deep Q Network 将状态当成神经网络的输入，然后经过神经网络计算后得到动作的奖惩值，这样就没必要在表格中记录奖惩值，而是直接使用神经网络生成奖惩值。Deep Q Network 的另一个优点是使用了经验回放，它克服了样本数据的相关性和非平稳分布的问题。

Deep Q Network 的结构图如图 3.5 所示：

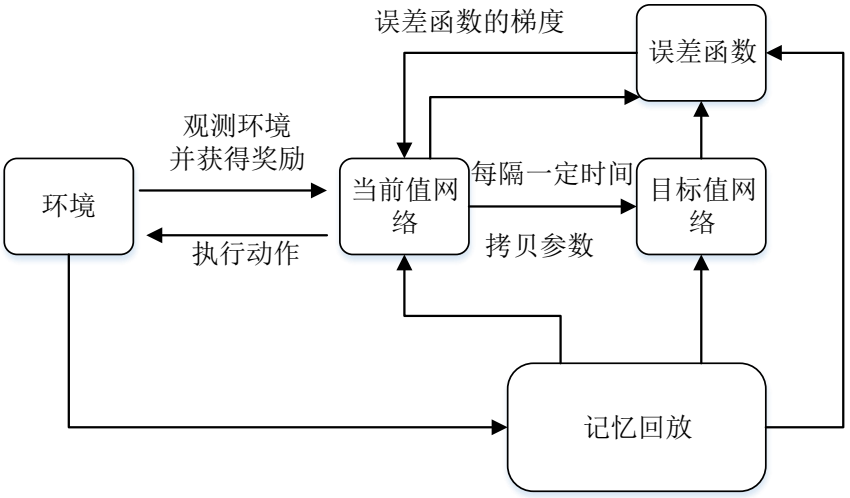

图 3.5 Deep Q Network 结构图

Deep Q Network 首先通过随机初始化当前值网络和目标值网络的参数，智能体根据当前值网络计算出来的值执行某个动作，执行动作后就会对环境造成影响。智能体通过观测当前的环境，通过评价函数获得一个回报（Q 值），这个回报可能是奖励或者惩罚，根据这个回报以及误差函数的梯度更新当前值网络的参数。隔一定时间，则将当前值网络的参数拷贝至目标值网络。

Deep Q Network 中有两个结构完全相同但是参数却不同的网络，预测 Q 估计的当前值网络使用的是最新的参数，而预测 Q 现实的目标神经网络参数使用的却是很久之前的，Q 值是指给予智能体的奖励或者惩罚值。当前值网络用来评估当前状态动作对的值函数。根据目标网络的输出，并根据误差函数计算误差梯度，更新当前值网络的参数，每经过一定次数的迭代，将当前值网络的参数复制给目标网络。引入目标网络后，在一段时间里目标 Q 值使保持不变的，一定程度降低了当前 Q 值和目标 Q 值的相关性，提高了算法稳定性。

3.5 边缘计算

由于基于以太网的消息具有时间不可靠性，云控制在实时性方面存在固有的挑战。由广域网内链路带宽差异、网络拥塞或远距离传输造成的延时，是控制稳定的一个重要问题。边缘计算和雾计算的兴起，正是为解决这一问题而提出。边缘计算和雾计算强调更接近数据产生地和使用地的信息处理，边缘计算处于云计算和移动设备计算之间。无人驾驶应用边缘计算的结构如图 3.6 所示。边缘节点在移动雾计算中可以对无人汽车进行极低延迟的数据处理，但边缘计算节点相比云计算中心在规定时间内计算能力是受限的。因此把边缘计算节点提供的稳定网络连接与云计算中心巨大的计算能力相结合，是解决无人驾驶汽车远程实时控制的绝佳组合。在传输速率和带宽方面，下一代移动通信网络技术标准 5G 技术可以为无人汽车提供强有力的保障，5G 的理论传输速度可达 10Gbps。

采用边缘计算的另外一个优点就是可以减少汽车端的计算能耗，目前电动汽车也正在蓬勃发展，续航里程是电动汽车的重要参数。由于无人驾驶汽车上的计算量巨大，计算系统消耗的能量也巨大，如果通过把一部分的计算任务从汽车端移到边缘节点上完成，可以有效降低汽车端的能耗，从而提高续航里程[58]。另外也可以通过适应节能驾驶行为来增加行驶距离，例如在接近红绿灯时，无人汽车通过与附近边缘节点通信，获取精确的速度建议，从而采用节能的速度到达红绿灯路口。

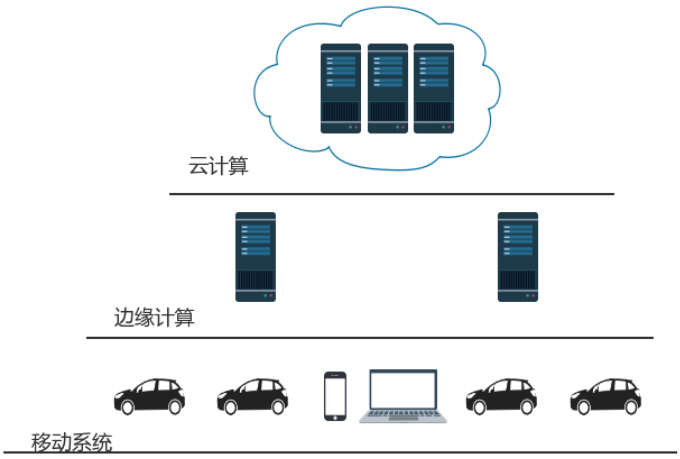

图 3.6 无人驾驶汽车云服务结构

无人驾驶汽车不但要与云平台通信，还要与一些基础设施进行通信，来获取一些必要的信息，如附近的交通流量信息以及停车场信息，来给无人驾驶系统选择合适的路线，避免拥堵。

深度学习计算和边缘计算相融合，也是近年来一些研究的趋势[59-60]。在无人车采集路况视频上传到云端进行训练的过程中，如果在边缘节点上对图像进行预处理，比如进行卷积和池化等操作，可以显著降低需要上传到云端的数据量。因为在卷积神经网络中，卷积和池化都可以压缩参数。

边缘节点中有三个部分：身份认证、移动管理和实时数据处理。

身份认证主要是指边缘节点和无人车的双向身份认证。包括边缘节点对无人车的身份进行识别和跟踪，同时边缘节点也要配合无人驾驶车辆对边缘节点进行身份识别认证。主要是为了防止黑客恶意攻击。

移动管理是为了给无人车提供连续性服务，当无人车驶出当前边缘节点服务范围时，就由这个移动管理模块完成切换操作。这里主要遇到的挑战是当边缘节点把计算结果返回给无人车时，而这时无人车已经驾驶出当前节点服务范围，怎么将结果正确发送给无人车。有两种解决方案：一个是由完成计算任务的边缘节点将计算结果广播给周围边缘节点，再它们完成计算结果返回。二是由正在给无人车服务的边缘节点向已完成计算的边缘节点发起请求，获取计算结果。

实时数据处理部分主要是在边缘节点上完成汽车端的一部分计算任务，减少汽车端的计算压力和能耗。解决与云计算中心通信远距离传输带来的时延问题。

3.6 云服务

无人车是移动系统，因此还需要云平台的支持。云平台主要从分布式计算及分布式存储对无人驾驶系统提供支持。在上汽通用汽车发布的企业车联网 2025 战略中，提出车联系统云服务的概念，并对 2017 到 2025 年企业在车联网方面的发展做出“精确到年”的详细规划。

无人车在未来还将支持手机远程控制，监控，云诊断等功能，我们可以假设这样的一个手机远程控制应用场景：通过手机给无人车发出指令，让它来机场接乘客。或者送乘客到机场后，通过手机控制无人车回家。这极大方便了人们的出行。云诊断是指汽车生产厂家通过无人汽车上传的一些历史日志数据，对汽车的健康情况进行分析，推荐车主对汽车进行维修和保养，预防故障的发生。这些功能都需要云平台的支撑。另外，无人驾驶系统中很多关键技术应用，包括用于验证新算法的仿真应用、高精度地图生成和深度学习模型训练都需要云平台的支持。比如深度学习模型的训练，它需要海量的路况数据，传感器数据和驾驶员操作的数据输入给深度学习模型，这么大的数据量需要借助云平台强大的计算能力才能完成。

高精度地图是无人驾驶汽车的核心技术之一，高精度地图可以起到类似于人脑对于空间的整体记忆与认知的功能，帮助无人车提前知道路面的复杂度，如坡度、方向等。与传统地图不同，高精度地图可以达到厘米级的精度，路面特征更加精准。高精度地图的生成过程非常复杂，涉及原始数据处理、点云生成、点云对齐、2D 反射地图生成、高精度地图标注、地图生成等阶段[2]，这些计算可提交给一个 Spark 集群来完成。与 Hadoop 不同，Spark 是基于内存的分布式计算，不同阶段间产生的中间数据不需要磁盘存储，因此可以加快运行速度。

在文献[61]中设计了一个统一的自动驾驶云平台。其中 alluxio 是一个基于内存的虚拟分布式文件系统，它可以作为缓存，为上层分布式计算提供数据存取服务，从而加快应用的读写速度[62]。同时还设计了一个由 CPU、GPU 和 FPGA 组成的异构计算层来提高计算能力，并采用 OpenCL 作为异构平台编写程序的框架。

无人驾驶的安全性和可靠性需要通过海量的功能和性能测试来保证。无人驾驶是一个复杂的系统工程，显然全部测试工作都集中在真车并且在实际道路上测试是一种成本非常高昂的方案。对无人驾驶进行仿真和模拟，需要保证模拟的真实和有效性，因此需要巨大的计算能力来完成，而云计算中心具备这种超大的计算能力。

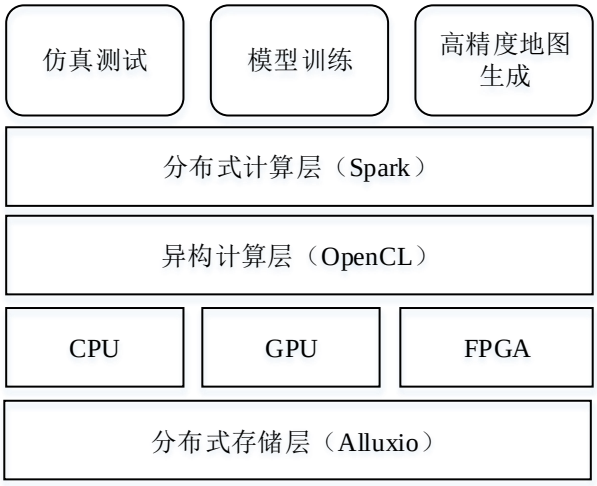

图 3.7 一个统一的自动驾驶云平台框架

3.7 本章小结

本章从无人驾驶的系统结构图出发，全面阐述了无人驾驶汽车集移动端、边缘节点和云端一体的计算结构，然后从传感、感知、决策以及边缘计算和云服务等方面详细介绍无人驾驶计算结构，并且介绍了无人驾驶汽车领域的一些关键算法。

第 4 章　硬件成本和能耗优化算法

无人汽车是一个异构分布式系统，异构处理器之间通过高速总线网络连接，比如 PCI 总线等。无人驾驶系统是一个典型的安全关键应用。无人驾驶系统必须满足以下三个要求：其一，实时性要求，系统必须确保捕捉到的传感器数据得到及时处理，截止时间对实时应用是一个很重要设计，错过这个截止时间，就有可能造成严重的后果；其二，系统必须确保高可靠性，确保所有计算任务正确完成；其三，系统必须在设计的能耗和资源限定下有效完成所有计算操作。

无人汽车属于工业嵌入式系统，汽车上分布着 100 多个处理单元，减少硬件成本可以为企业获取更多的利润空间，目前汽车里处理器成本在 25-110 美元之间，通过减少处理器个数来降低硬件成本具有重大意义。对于安全关键应用，必须满足安全标准，可靠性目标广泛用于安全标准。比如汽车电子安全标准 ISO26262，以及各类工业嵌入式安全标准 IEC61508。同时为了减少环境污染，电动汽车逐渐兴起，续航里程对电动汽车是一个重要的参数，通过减少能耗提升续航里程可以提高电动汽车的产品竞争力。过去的研究都只是考虑其中的某个约束条件，去优化另一个参数，没有将实时、可靠性、能耗、硬件成本这些参数统筹考虑。因此，在无人驾驶这类安全关键的汽车功能应用的设计阶段，如何在满足的应用的硬实时性和可靠性前提下，降低硬件成本和总能耗就尤为重要。

4. 1 模型

4.1.1 应用模型

本文所考虑的系统是一个异构分布式系统，多个异构处理器通过高速网络相连，每个处理器包含 CPU、RAM、非易失性存储器以及一个网卡。

本文用$U = \{u_1, u_2, \ldots, u_{|U|}\}$表示处理器集合，一个包含多个子任务的应用程序 G用有向无环图（DAG）来表示，如图 4.1 所示。DAG 里的节点表示任务，边表示任务之间的通信代价。G=(T,M), T表示任务节点集合，每个子任务$t_i \in T$。M表示通信边集合，这些边也表示了任务之间具有的优先约束。$m_{i,j} \in M$表示从任务t_i到任务t_j的通信代价。在分布式系统中，一个任务在一个处理器上执行完后，需要把数据传递给下一个任务执行，下一个任务可能位于另外一个处理器上执行，由此会存在通信时间。如果任务t_i和任务t_j在同一个处理器上执行，则通信代价$m_{i,j}=0$。每个任务在不同处理器上的执行时间用矩阵W表示，$w_{i,j} \in W$表示任务t_i在处理器u_j上的执行时间。如果某项任务不适宜调度到某个处理器上执行，则可将$w_{i,j}=\infty$。

接下来本文定义任务的最早开始时间EST（Earlist Start Time），和最早结束时间EFT（Earlist Finish Time）[63]。$EST(t_i,u_j)$和$EFT(t_i,u_j)$分别表示任务t_i在处理器u_j上的最早开始时间和最早结束时间。如公式(4.1)和公式所示(4.2)。

$$\begin{cases} EST(t_{entry},u_k)=0; \\ EST(t_i,u_k)=\max\left(avail_{u_k},\max_{t_x\in pred(t_i)}(AFT(t_x)+m_{x,i})\right) \end{cases} \tag{4.1}$$

$$EFT(t_i,u_k)=EST(t_i,u_k)+w_{i,k} \tag{4.2}$$

其中t_{entry}表示 DAG 中的入口任务，$avail_{u_k}$表示处理器u_k的最早可用时间，即处理器可以执行任务的时间。$AFT(t_x)$表示任务t_x的实际结束时间。当任务t_x调度到处理器上后，$AFT(t_x)$就可以计算出来。$pred(t_i)$表示任务t_i的前驱任务。

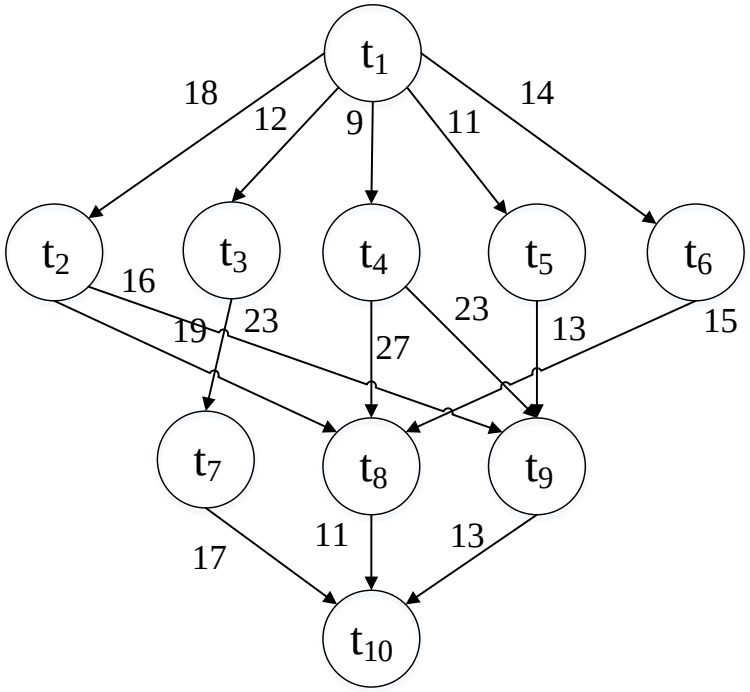

图 4.1 示例应用的 DAG 示例图

表 4.1 示例应用的各个在处理器上的执行时间

任务	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
u_1	14	13	11	13	12	13	7	5	18	21
u_2	16	19	13	18	13	16	15	11	12	7
u_3	19	18	19	17	10	19	11	14	20	16

本文使用$rank_u$来确定任务调度的优先级[13, 17, 23, 28]，$rank_u$可以使用如下公式 4.3 来计算。

$$rank_u(t_i)=\overline{w_i}+\max_{t_j\epsilon succ(t_i)}(\overline{m_{i,j}}+rank_u(t_j)) \tag{4.3}$$

$$\overline{w_i}=\sum_{k=1}^{|U|}w_{i,k}\Big/|U| \tag{4.4}$$

其中$\overline{w_i}$是任务t_i在处理器集合U_n中的平均计算代价，$succ(t_i)$是任务t_i的后继任务集合，$\overline{m_{i,j}}$是平均通信时间。通过对每个任务的$rank_u$进行降序排列，每次选择$rank_u$最大的任务优先调度。注意只有当t_i的所有先继任务执行完成后，t_i才会准备执行。根据公式，示例应用的任务优先级$rank_u$如表 4.2 所示，因此任务调度的优先级为$\{t_1,t_2,t_3,t_4,t_5,t_6,t_7,t_8,t_9,t_{10}\}$。

表 4.2 示例应用的各任务的优先级

任务	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
$rank_u$	108	77	80	80	69	63.3	42.7	35.7	44.3	14.7

任务调度有静态和动态之分，根据是否可中断低优先级任务执行，分为可剥夺和不可剥夺，本文研究的是硬实时系统下不可剥夺的静态调度。本文用$RT(G)$表示应用的响应时间，用$RT_{goal}(G)$表示应用所允许的最大响应时间（也叫做截止时间）。每个任务的响应时间等于计算时间代价加通信时间代价，$RT(t_i,u_k)$表示任务t_i在处理器u_k上的响应时间，$pred(t_i)$表示任务t_i的前驱任务，$proc(i)$表示任务t_i所在的处理器索引号。

$$RT(t_i,u_k)=w_{i,k}+\max_{t_x\in pred(t_i)}(m_{x,i}) \tag{4.4}$$

$$RT(G)=\max_{t_i\in T}AFT(t_i) \tag{4.5}$$

4.1.2 可靠性模型

类似许多论文一样，本文只考虑硬件的瞬态故障，不考虑永久故障和通信故障[16,22]。因为在一些功能安全标准中（ISO26262），只将瞬态故障和可靠性组合在一起。在汽车电子网络中，比如 Flexray 总线，提供了两个并行通道，来保障通信的高可靠性。通常一个 DAG 应用的任务的瞬态故障率服从泊松分布[16]，单位时间内的可靠性$R(t)=e^{-\lambda t}$, t是任务执行时间，λ是处理器的单位时间错误率。用如下公式(4.6)表示任务t_i在处理器u_k上执行的可靠性。

$$R(t_i,u_k)=e^{-\lambda_k w_{i,k}} \tag{4.6}$$

则整个应用的可靠性为：

$$R(G)=\prod_{t_i\epsilon T}R(t_i,u_{proc(t_i)}) \tag{4.7}$$

本文使用$R_{goal}(G)$表示应用的可靠性目标，必须要确保应用的可靠性大于可靠性目标根据安全标准，如下公式将整个应用的可靠性目标$R_{goal}(G)$转化为每个任务要满足的可靠性指标[16]。

$$R_{goal}(t_j)=\frac{R_{goal}(G)}{\prod_{x=1}^{j-1}R(t_x,u_{proc(x)})\times\prod_{y=j+1}^{|T|}R_{max}(t_y)} \tag{4.8}$$

其中$R_{goal}(t_j)$表示任务t_j要满足的可靠性，$R_{max}(t_y)$表示任务t_y的最大可靠性，通过遍历任务t_y在每个处理器上的可靠性得出，同时$R_{min}(t_y)$ 表示任务t_y的最小可靠性。

$$R_{max}(t_y)=\max_{u_k \epsilon U} R(t_y,u_k) \tag{4.9}$$

$$R_{min}(t_y)=\min_{u_k \epsilon U} R(t_y,u_k) \tag{4.10}$$

4.1.3 能耗和硬件成本模型

本文只考虑处理用于计算任务时所消耗的能量。并且固定处理器的频率，因为许多研究表明频繁的改变处理器频率，将会减少处理器的寿命[24]。因此在一些工业系统中不提倡频繁改变处理器频率和电压，而是选择一个合适的频率运行。

本文用$e_{i,j}$表示第 i 个任务在第 j 个处理器上执行所消耗的能量，$e_{i,j}$可由如下公式计算，其中$\overline{P_j}$为处理器u_j的平均功率，可由处理器的厂商提供。本文只考虑处理器用于计算任务时所产生的能耗，固定处理器的工作频率。因此整个应用的能耗$E(G)$可由以下公式计算。

$$e_{i,j}=\overline{P_j}\times w_{i,j} \tag{4.11}$$

$$E(G)=\sum_i^{|T|} e_{i,proc(i)} \tag{4.12}$$

本文用$Price(G)$表示所采用处理器的硬件总成本，$price(u_i)$表示处理器u_i的市场价格。

$$Price(G)=\sum_{u_i \epsilon U} price(u_i) \tag{4.13}$$

本文使用如下公式将能耗和硬件成本两个优化目标转化为一个优化目标，为了将能耗和硬件成本统一到一个参考系下进行优化，我们引入了权重系数c, 用于归一化硬件成本和能耗。$Optimize(G)$越大表示设计方案越优秀。

$$Optimize(G)=\frac{c_1}{E(G)}+\frac{c_2}{Price(G)} \tag{4.14}$$

4.2 问题描述

给定一个并行应用G，响应时间约束$RT_{goal}(G)$和可靠性约束$R_{goal}(G)$。给定一个已知参数的处理器集合U，U_n是集合U的非空子集，即$U_n \subseteq U$。本文提出的硬件成本和能耗优化算法的目标是寻找一个U_n，在满足$RT_n(G) \leq RT_{goal}(G)$且$R_n(G) \geq R_{goal}(G)$条件下，把任务调度到处理器集合$U_n$上执行，最小化$E_n(G)$和$Price_n(G)$。

4.3 硬件成本与能耗优化算法

由于 IHCO、EHCO 和 EEHCO 等算法是基于贪心策略，通过不断移除不必要的处理器使得Price(G)不断降低，这些算法有可能陷入局部最优解。因此本文设计了一个结合模拟退火的基因算法。算法从一组随机产生的初始解开始进行全局最优解搜索,通过基因算法中的选择操作淘汰适应度低的个体，通过交叉操作来产生一组新个体,然后针对每个新产生的个体采用模拟退火算法进行邻域搜索,并以其结果作为下一代种群中的个体，不断循环这两个过程，直到终止条件满足。将两个算法的优点充分结合起来,从而提高了成本优化问题的求解质量。

4.3.1 解的编码

基因算法中用一个染色体表示问题的一个可行解。每条染色体用二进制 01 序列编码成处理器的选择方案。由多个染色体组成一个种群，本文用一个二维数组Arr[N][|U|]表示种群，N表示总染色体个数，|U|表示处理器集合大小。数组中每一行表示一条染色体，即种群中的一个个体。如：Arr [n]=01000101,则表示共有 8 个处理器，由于该染色体的第 1,6,7 号位置为 1，故染色体 n 的处理器集合$U_n=\{u_1,u_5,u_7\}$。

4.3.2 适应度评价

适应度用来评价种群中每个染色体的生存能力，即对应解决方案的优劣。适应度越大，则表示该个体性能越优。本文使用公式（4.14）将硬件成本和能耗这两个优化目标转化为一个优化目标。因此本文使用如（4.15）公式来评价每个个体的适应度。

$$fit(n)=\begin{cases} -1 \quad , RT_n(G)>RT_{goal}(G) \text{ or } R_n(G)<R_{goal}(G) \\ \frac{c_1}{E_n(G)}+\frac{c_2}{Price_n(G)}, RT_n(G)\le RT_{goal}(G) \text{ and } R_n(G)\ge R_{goal}(G) \end{cases} \tag{4.15}$$

由于应用的响应时间和可靠性为硬性限制条件，所以若响应时间和可靠性不满足要求，则该个体适应度为-1，在种群迭代中将被淘汰。

对染色体进行适应度评价过程如下：根据染色体解码得到对应的处理器集合。在将任务调度到处理器上执行时，分为两个阶段，任务选择阶段和处理器选择阶段。在任务选择阶段，我们使用$rank_u$作为每个任务的优先级，在满足任务优先约束条件下，依次选取优先级最大的任务调度。在处理器选择阶段，通过遍历每个处理器，选择可以满足该任务可靠性目标且响应时间最短的处理器作为最终的执行处理器。在将任务调度完成后，根据公式（4.15）计算该染色体的适应度值。根据输入的染色体，计算适应度的算法如下：

算法 4.1: 计算适应度算法

Input：Arr[n] ,G, $R_{goal}(G)$, $RT_{goal}(G)$
Output：fit(n)
1:解码染色体 Arr[n]得到处理器集合U_n;
2:根据任务优先级$rank_u$，对任务排序后放到优先队列 task_list 中;
3:while(task_list 有未调度任务)
4: t_i= rank_task_list.out();
5: 根据公式计算$R_{max}(t_i)$, $R_{min}(t_i)$, $R_{goal}(t_i)$;
6: var $p_{proc(i)} = NULL$, $\mathrm{RT}(t_i, u_{proc(i)}) = \infty$, $\mathrm{R}(t_i, u_{proc(i)}) = 0$;
7: for(每个处理器 $u_k \epsilon \mathrm{U}_n$)
8: 计算$\mathrm{R}(t_i, u_k)$;
9: if $\mathrm{R}(t_i, u_k) \geq \max\{R_{min}(t_i), R_{goal}(t_i)\}$ then
10: 计算$\mathrm{RT}(t_i, u_k)$;
11: if $\mathrm{RT}(t_i, u_k) < RT(t_i, u_{proc(i)})$ then
12: proc(i)=k;
13: $\mathrm{R}(t_i, u_{proc(i)}) = \mathrm{R}(t_i, u_k)$;
14: $\mathrm{RT}(t_i, u_{proc(i)}) = \mathrm{RT}(t_i, u_k)$;
15: $e_{i,proc(i)} = \overline{P_k} \times w_{i,k}$;
16: end if
17: end if
18: end for
19:end while
20:计算 $\mathrm{R}_n(\mathrm{G})$, $RT_n(G)$, $\mathrm{E}_n(\mathrm{G})$, $\mathrm{Price}_n(\mathrm{G})$;
21: 计算 fit(n);

(1)第 1-2 行代码表示，对染色体进行解码，得到相应的处理器集合U_n。计算所有任务的$rank_{\mathrm{u}}$，并按优先级进行排序；

(2)第 3-6 行表示依次从任务集合中选取优先级大的任务t_i开始调度，并计算该任务的$R_{max}(t_i)$, $R_{min}(t_i)$, $R_{goal}(t_i)$，根据公式(4.9)(4.10)(4.8)；

(3)在第 7-18 行，算法遍历处理器集合，计算任务t_i在每个处理器上的可靠性，并在满足$\mathrm{R}(t_i, u_k) \geq \max\{R_{min}(t_i), R_{goal}(t_i)\}$条件下，选取使得任务$t_i$响应时间最短的处理器作为最终的处理器;

(4)第 19-21 行，根据公式(4.7)(4.5)(4.12)(4.13)计算得到$\mathrm{R}_n(\mathrm{G})$, $RT_n(G)$, $\mathrm{E}_n(\mathrm{G})$, $\mathrm{Price}_n(\mathrm{G})$，并返回最终的适应度；

适应度评价算法的时间复杂度分析如下：遍历每个任务需要$O(|T|)$时间，在选择处理器阶段，需要$O(|U|)$时间，因此整个适应度评价算法时间复杂度为$O(|T| \times |U|)$。

4.3.3 选择和交叉

计算出种群中每个个体的适应度后，对每个个体的适应度进行排序。保存好当前种群中适应度最好的个体，选取种群中适应度较好的前 40%个体生存下来，再从剩下个体中随机选择 10%适应度较差的个体生存下来，防止陷入局部最优解。其余个体被淘汰。再由这生存下来的 50%的个体，两两配对，通过随机生

成一个交叉位置，根据这个位置互换染色体的某部分来产生下一代个体。如图4.2所示：

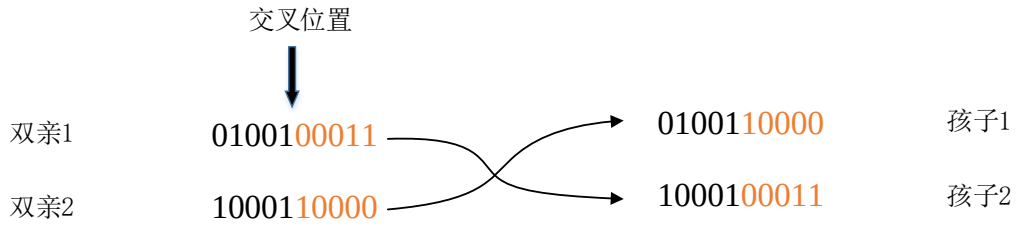

图 4.2 交叉示意图

4.3.4 模拟退火

本文用模拟退火算法来替代了基因算法的变异部分，因为模拟退火和变异都是用来提升算法的局部搜索能力。对种群中每个个体进行模拟退火操作，其中模拟退火操作如下：

1. 本文使用temp表示模拟退火的温度，模拟退火算法从初始温度$temp_0$开始，将交叉后的个体作为初始解S。

2. 对种群中每个个体的染色体以变异率p_m进行随机变异，产生新解S′。

3. 计算新解S′的适应度，如果$fit(S') - fit(S) > 0$，则接受这个新解，否则以一定的概率p_a接受这个新解，之后在新解基础上进行操作。

$$p_a = e^{\frac{fit(S') - fit(S)}{temp}} \tag{4.16}$$

4. 温度temp根据衰减因子α减少，然后判断temp是否大于 1。如大于 1 则返回第 2 步继续，否则结束退出。

在变异操作中，随机确定染色体中的某个二进制位是否进行变异。变异即将染色体中该二进制位的 0 替换为 1，1 替换为 0。如图 4.3 所示，变异前染色体为 0100000100，对第 2 和第 6 个基因进行变异，则变异为 0000010100，表示变异前的处理器集合为$U_n=\{u_1,u_7\}$，变异后为$U_n=\{u_5,u_7\}$。在变异操作后，需要对染色体的合法性进行检查，移除二进制位全 0 的染色体，然后随机生成新的染色体替换掉这些非法的染色体。因为二进制位全 0 表示处理器集合为空，这显然不符合实际。

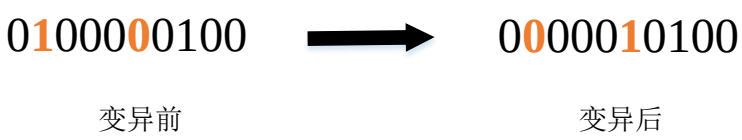

图 4.3 变异示意图

满足指定条件后，如适应度大于某一个值或迭代次数大于某一个值，则迭代终止，输出最好的染色体所代表的处理器子集合作为最终解决方案。整个算法流

程图如图 4.4 所示，伪代码如算法 4.2 所示：

算法 4.2: The HCECO Algorithm

Input： $U = \{u_1, u_2, \dots, u_{|U|}\}$,G, $R_{goal}(G)$, $RT_{goal}(G)$
Output： U_n
1:随机产生初始种群数组 Arr[N][|U|],种群大小为 N;
2:while(迭代次数不满足)
3:使用算法 1 评价种群中的每染色体,将适应度保存到 fit[N];
4:对 fit[N] 降序排列:
5:保存最好的染色体,将适应度较好的前 40% 染色体复制到临时数组 temp_arr,然后从剩下的染色体中随机选择 10%复制到 temp_arr:
6:对 temp_arr 中的染色体进行交叉操作，产生新的染色体体:
7:对temp_arr中的染色体进行模拟退火操作:
8:将 temp_arr 复制到 Arr;
9:用随机生成法替换 Arr 中的不合法染色体:
10:end while
11:解码最好的染色体，得到处理器子集合 U_n;
12:return U_n

(1) 第 1 行代码表示，随机初始化种群中的每条染色体；

(2) 第 2-10 行代码：当终止条件不满足的条件下，不断对种群进行适应度评价，选择，交叉，模拟退火操作；

(3) 第 11-12 行代码返回种群迭代过程中最好的染色体对应的处理器结合，作为最终方案；

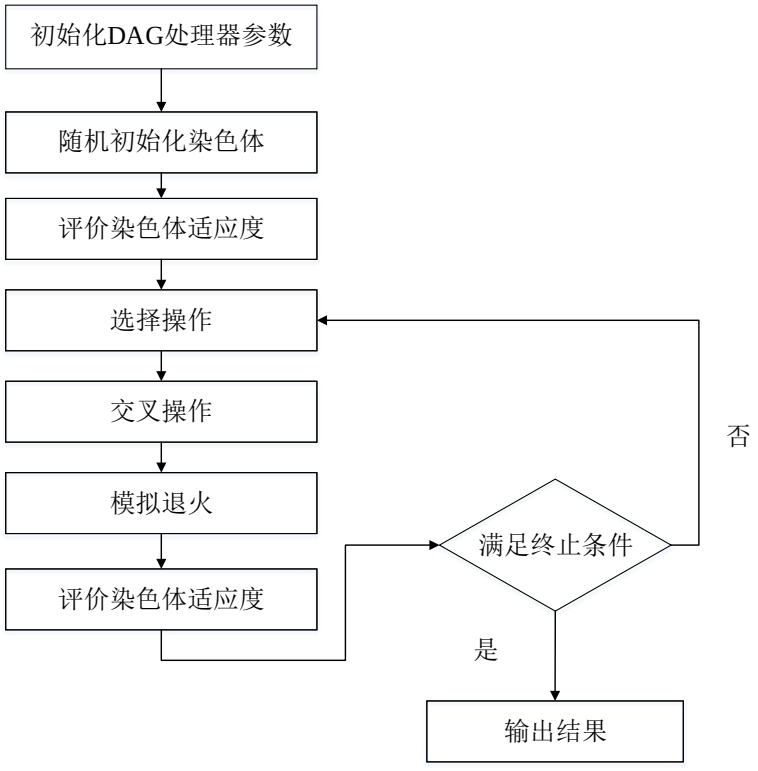

图 4.4 HCECO 算法流程图

整个算法的时间复杂度分析如下：设该算法总共需要迭代 m 次，由于适应度评价算法的时间复杂度为$O(|T|\times|U|)$，所以模拟退火操作时间复杂度为$O(temp\times|T|\times|U|)$。因此本文 HCECO 算法的时间复杂度为$O(m\times temp\times|T|\times|U|)$。

4.4 实验设计与评估

为了评价本文提出的 HCECO 算法，本文使用快速傅里叶变换（Fast Fourier Transform, FFT）[64]和高斯消元[65]这两个典型并行应用和一个真实汽车应用运行在我们的模拟系统上来测试算法性能。本文不讨论具体的截止时间和可靠性目标分析，因为本文主要聚焦于无人驾驶汽车的设计阶段，针对硬件成本和能耗进行优化。本文让这些测试应用程序运行在一个包含 64 个处理器的模拟异构分布式系统上。处理器计算每个任务的时间，任务之间的通信时间以及处理器的参数都是随机生成的。本文通过重复试验证明 HCECO 算法不会陷入局部最优解。

我们让$RT_{goal}(G)$和根据具体应用固定在一个合理的值，$R_{goal}(G)=R_{heft}(G)$，$R_{heft}(G)$表示 HEFT 算法的应用可靠性。$c_1=E_{heft}(G)$以及$c_2=Price_{heft}(G)$，$E_{heft}(G)$表示 HEFT 算法产生的能耗，$Price_{heft}(G)$表示 HEFT 算法产生的硬件成本，它们通过执行 HEFT 算法后得到。因为 HEFT 算法所产生的硬件成本和能耗都比其他算法的成本和能耗高，因此我们用 HEFT 算法的硬件成本和能耗来进行归一化。本文通过能耗，硬件成本这两个指标来评价 HCECO 算法优劣。试验的参数范围：$5ms\le w_{i,k}\le 100ms$，$5ms\le m_{i,j}\le 100ms$，$0.000001/ms\le\lambda_k\le 0.000009/ms$，$\$20\le Price_n\le\$110$[24]，$30w\le\overline{W_j}\le 200w$，基因算法种群大小设为 18，迭代次数设置为 400，模拟退火初始温度设置为 100，衰减因子$\alpha=0.8$。本文研究的是嵌入式系统的设计阶段，因此本文让并行应用执行在一个有 64 个仿真异构处理器上。处理器处理每个任务所需要的时间、任务间通信时间，以及处理器的参数采用随机生成的方式。

4.4.1 快速傅里叶变换应用实验

为了测试本文提出的算法针对复杂真实应用的优化能力，本次试验采用快速傅里叶变换这个典型的并行应用程序来进行实验[13, 17, 23, 28]。该应用的任务个数范围为 95 到 2559。在经过多次重复实验中，HCECO 算法实验结果如图 4.5-4.6 所示。

图 4.5 结果表明本文提出的 HCECO 算法的能耗总是最小的，相比 HEFT、IHCO 算法，HCECO 算法的能耗降低了 16-987（J）,比 EHCO,EEHCO 等算法

能耗降低了 32-915(J)，应用的任务个数越大，能耗优化越明显。图 4.6 结果表明本文提出的 HCECO 算法相比 HEFT 以及 IHCO 算法硬件成本降低了 3458-4283（美元）。但相比 EHCO、EEHCO 等算法本文的硬件成本上升了 46-289（美元），但 EHCO、EEHCO 等算法只针对硬件成本进行优化，并没有考虑能耗。然而，当任务个数为 95 的时候，HCECO 的硬件成本比 SEEHCO 低$17，尽管 SEEHCO 算法只优化硬件成本。

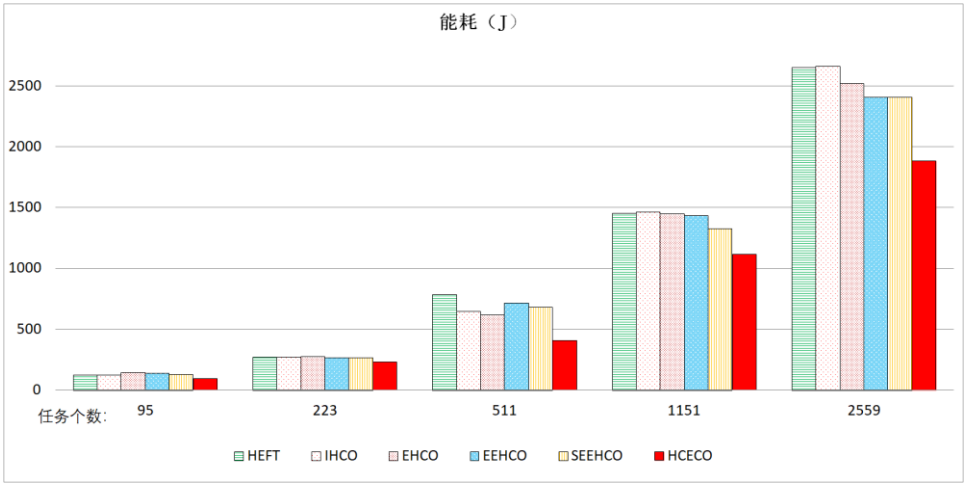

图 4.5 FFT 应用能耗

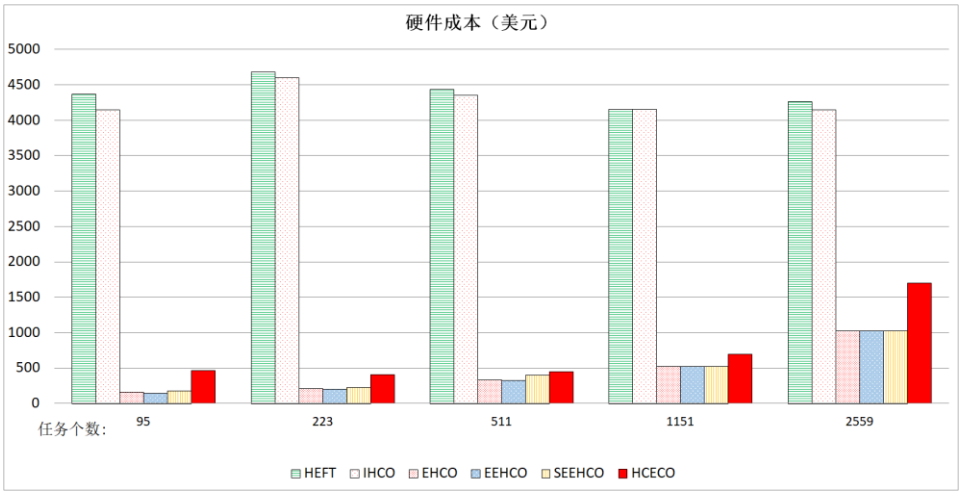

图 4.6 FFT 应用硬件成本

4.4.2 高斯消元应用实验

本次实验采用高斯消元这个典型的并行应用程序来进行实验[13, 17, 23, 28]，该应用的任务个数范围为 104 到 2484。实验结果如图 4.7-4.8 所示。

图 4.7 结果表明本文提出的 HCECO 算法能耗相比 HEFT、IHCO 算法降低了 6-1167（J）,比 EHCO,EEHCO 等算法降低了 11-1810(J)。图 4.8 结果表明本文提出的模拟退火基因算法相比 HEFT 以及 IHCO 算法硬件成本降低了 3354-4023（美元）。相比 EHCO 等算法、本文算法的硬件成本在任务个数为 104-

902 时硬件成本下降了 163-1163（美元）。因为实验参数是随机生成的，因此本文的算法比 EHCO 更接近最优解。

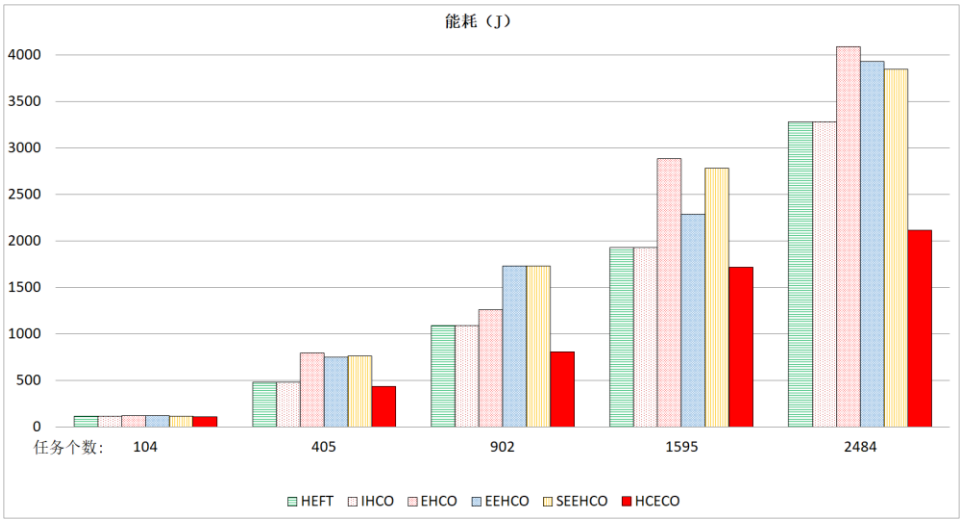

图 4.7 高斯消元应用能耗

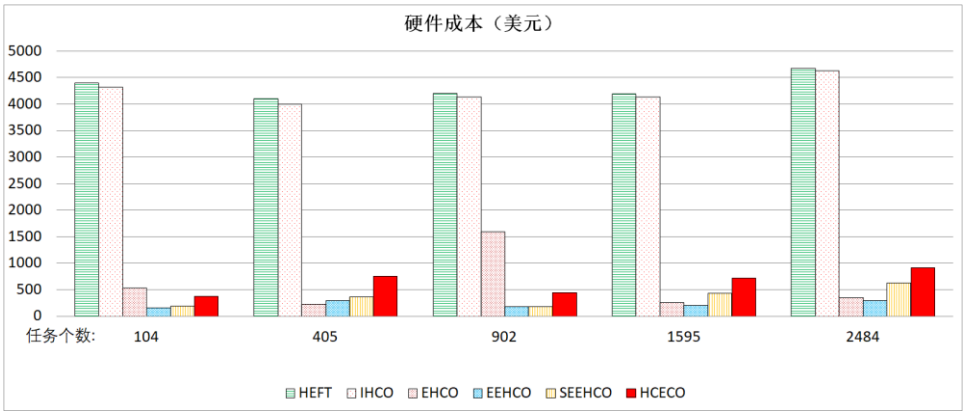

图 4.8 高斯消元应用硬件成本

4.4.3 汽车应用实验

为了测试 HCECO 算法针对真实汽车应用的优化能力，本次实验采用论文[27]中的一个真实汽车应用来进行实验。该应用的 DAG 如图 4.9 所示。该应用包含 6 个功能模块：具有 7 个任务的发动机控制(t_1-t_7)，具有 4 个任务的自动变速箱(t_8-t_{11})，具有 6 个任务的防抱死制动系统$(t_{12}-t_{17})$，具有 2 个任务的车轮角度传感器$(t_{18}-t_{19})$，具有 5 个任务的悬浮控制器$(t_{20}-t_{24})$，具有 7 个任务的车身控制器$(t_{25}-t_{31})$。本次实验的参数如下：$100\mu s\leq w_{i,k}\leq 400\mu s$，$100\mu s\leq m_{i,j}\leq 400\mu s$，$0.000001/\mu s\leq\lambda_k\leq 0.000009/\mu s$。根据安全评估这个应用的截止时间是1000μs，即 $(RT_{goal}(G)=1000\mu s)$。我们首先让$R_{goal}(G)=R_{heft}(G)$，然后$R_{goal}(G)=0.99$压力测试 HCECO 算法。

当$R_{goal}(G)=R_{heft}(G)$时，实验结果如图 4.10-4.11 所示，详细结果如表 4.3 所示。图 4.10 表明本文算法的能耗相比其他算法减少了 20-39(10^{-3}J)。本文仅统计了该应用运行一次时所产生的能耗，但是在真实汽车里，该应用是持续不断运行的，只要车辆处于启动状态。因此随着该应用的不断运行，本文算法节省的能耗将不断增加。图 4.11 表明 HCECO 算法相比 HEFT 和 IHCO 算法减少了\$649-\$810。与 EHCO 和 SEEHCO 算法比较，HCECO 算法的硬件成本减少了\$52-\$145,。相比 EEHCO 算法，HCECO 算法硬件成本提高了\$8。

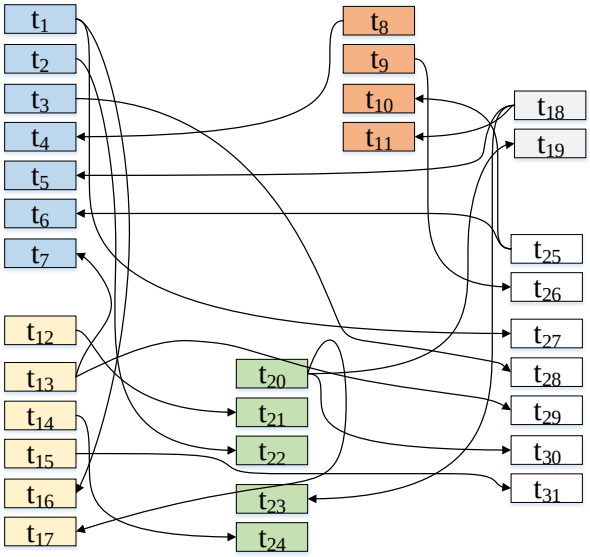

图 4.9 真实汽车应用 DAG

表 4.3 真实汽车应用详细实验结果

算法	Price(G)	E(G)	RT(G)	R(G)
HEFT	1104	264	554	0.979
IHCO	943	283	554	0.979
EHCO	346	283	992	0.979
EEHCO	286	271	952	0.981
SEEHCO	439	273	992	0.985
HCECO	294	244	975	0.981

本次实验结果表明，HCECO 算法的硬件成本比 EHCO 和 SEEHCO 都要低，尽管 EHCO 和 SEEHCO 算法仅考虑硬件成本优化。在采用不同的参数，经过多次重复实验，HCECO 算法相比其他算法，能耗平均减少了 14%-24%。相比 HEFT 和 IHCO 算法，硬件成本平均减少了 68%-73%，相比 EHCO，EEHCO 和 SEEHCO 算法，本文硬件成本减少了 16%-33%或者升高了 3%-17%。

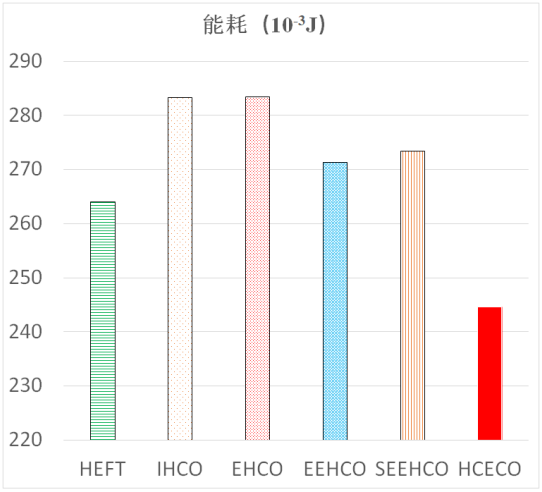

图 4.10 真实汽车应用实验能耗

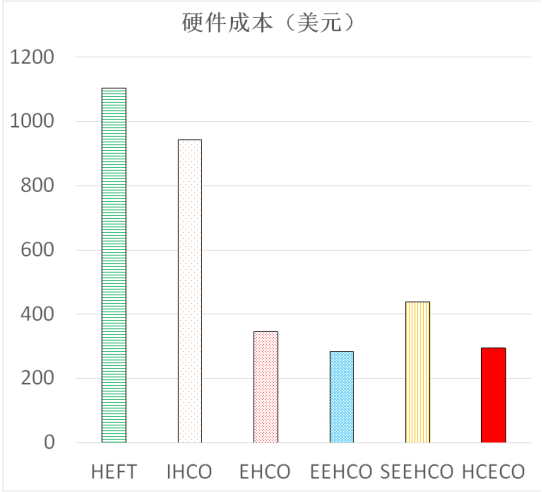

图 4.11 真实汽车应用实验硬件成本

为了压力测试本文提出的 HCECO 算法，我们采用汽车功能安全标准 ISO26262 中的可靠性相关的值来进行实验。如表 4.4 所示，在 E2 等级里，可靠性目标是 0.99，所以我们让$R_{goal}(G)=0.99$来进行实验。结果如表 4.5 所示，结果表明只有本文算法可以满足可靠性指标。基于这些实验数据，本文提出的 HCECO 算法可以在满足实时性和可靠性约束的前提下，优化硬件成本和能耗。但当可靠性目标大于 0.99 时，所有的算法都不能满足可靠性指标。

表 4.4 ISO26262 可靠性相关值

暴露等级	暴露率	可靠性需求
E1 非常低的概率	未指定	至少 0.99
E2 低概率	<1%	0.99
E3 中概率	[1%,10%]	> 0.9
E4 高概率	>10%	≤ 0.9

表 4.5 当可靠性目标为 0.99 时的实验结果

算法	Price(G)	E(G)	RT(G)	R(G)
HCECO	403	247	967	0.990

4.5 本章小结

本章我们针对无人驾驶汽车这类安全关键应用，提出了一个硬件成本和能耗优化算法。它通过结合模拟退火的基因算法减少处理器的使用数目，然后把应用的每个子任务调度到这些处理器上执行，在调度任务时，保证每个任务的响应时间最短并且满足任务的可靠性目标。最后通过三个应用的实验证明了本文提出的 HCECO 算法可以在满足硬实时和可靠性约束的前提下，有效地降低硬件成本和能耗。

第 5 章　可靠性保障及性能和能耗优化

系统可靠性是无人驾驶系统最重要的特性，因计算节点宕机或者随机硬件错误，导致计算任务执行不正确，可能会给无人驾驶汽车带来灾难性的后果。本章依旧将无人驾驶汽车电子系统抽象建模为一个异构分布式系统，在此基础上进行研究。对计算任务进行任务复制，从而提高系统可靠性是一个非常可行的方法。由于对计算任务进行复制，冗余会增加资源消耗，因此优化资源消耗对新型电动无人驾驶汽车具有重要意义。本文针对现有的容错可靠性保障调度算法进行了改进，提出了一个 PECORG(Performance and energy consumption optimization under reliability guarantee)算法。相比现有方法，PECORG 算法的能量消耗更低和应用执行时间更短。

5.1 模型

5.1.1 容错机制下的可靠性模型

采用容错机制时，一个任务可能会有多个副本，分布在不同的处理器上执行。为了保证容错的有效性，一个处理器不允许执行相同任务的多个副本。假设一个任务有$n(n \geq 1)$个副本，则任务t_i的可靠性可通过如下公式计算。

$$R(t_i)=1-\prod_{k=1}^{n}\left(1-R\left(t_{i,k},u_{proc(t_{i,k})}\right)\right) \tag{5.1}$$

其中$t_{i,k}$表示任务t_i的第 k 个副本，$u_{proc(t_{i,k})}$表示执行$t_{i,k}$任务的处理器。

在容错机制下任务的最大可靠性为在每个处理器上都有一个任务副本，因此任务最大可靠性计算公式为：

$$R_{max}(t_i)=1-\prod_{k=1}^{|U|}\left(1-R\left(t_{i,k},u_{proc(t_{i,k})}\right)\right) \tag{5.2}$$

文献[35]中提出了一个容错机制下的基于几何平均值的可靠性预分配方法，而不是使用最大任务可靠性作为预分配值，基于几何平均值可以有效避免分配不平衡，减少资源浪费。几何平均值是指|T|个值的乘积开|T|次方。我们对每个任务定义一个可靠性上界值，即

$$R_{up}(t_i)=\sqrt[|T|]{R_{goal}(G)} \tag{5.3}$$

然后整个应用的几何平均值可以表示为：

$$GM(G)=\sqrt[|T|]{\frac{R_{goal}(G)}{R_{min}(G)\times R_{up}(G)}} \tag{5.4}$$

其中$R_{min}(G)$和$R_{up}(G)$的计算公式如下；

$$R_{min}(G)=\prod_{i=1}^{|T|}R_{min}(t_i) \tag{5.5}$$

$$R_{up}(G)=\prod_{i=1}^{|T|}R_{up}(t_i) \tag{5.6}$$

那么任务的可靠性预分配值为

$$R_{pre}(t_i)=GM(G)\times R_{min}(t_i)\times R_{up}(t_i)$$

$$=\sqrt[|T|]{\frac{1}{R_{min}(G)}}\times R_{min}(t_i)\times R_{up}(t_i)$$

$$=\sqrt[|T|]{\frac{R_{goal}(G)}{R_{min}(G)}}\times R_{min}(t_i) \tag{5.7}$$

根据上述定义的未调度任务的可靠性预分配值，我们将应用的可靠性目标转换成具体每个任务的可靠性目标，如下公式所示：

$$R_{goal}(t_j)=\frac{R_{goal}(G)}{\prod_{x=1}^{j-1}R(t_x,u_{proc(x)})\times\prod_{y=j+1}^{|T|}R_{pre}(t_y)} \tag{5.8}$$

因为任务t_j受到任务可靠性上界的约束，所以$R_{goal}(t_j)$还需满足如下条件：

$$R_{goal}(t_j)=\min\{R_{up}(t_j),R_{goal}(t_j)\} \tag{5.9}$$

为了确保应用的可靠性目标得到满足，本文对任务的可靠性目标做了如下的改进，$R_{goal}(t_j)$还需满足如下条件：

$$R_{goal}(t_j)=\max\{R_{goal}(G),R_{goal}(t_j)\} \tag{5.10}$$

因为如果$R_{goal}(t_j)<R_{goal}(G)$，而且在实际中每个任务的可靠性必定小于 1，则整个应用的可靠性$R(G)$必定小于$R_{goal}(G)$，所以$R_{goal}(t_j)$必须满足公式（5.10）。

尽管这种基于几何平均的预分配方法可以防止分配不平衡，从而减少应用响应时间，但是由于存在$R_{goal}(t_j)>1$的可能性（通过举反例可以证明），而实际上任何一个任务的可靠性不可能大于 1，所以在这种情况下$R_{goal}(G)$将无法满足。为了防止这种情况，需要保证已分配任务的可靠性乘上当前正在分配任务t_j的可靠性不能小于应用目标可靠性，否则继续增加该任务副本数目，提高该任务的可靠性，如公式（5.11）所示：

$$\prod_{x=1}^{j-1}R(t_x,u_{proc(x)})\times R(t_j)>R_{goal}(G) \tag{5.11}$$

5.1.2 响应时间模型

由于在容错机制下，每个任务可能有多个副本，因此每个任务的响应时间将与上章不同。

在容错机制下，每个任务可能会有多个副本被执行，在实际应用中，只要一个副本执行成功，则该任务执行成功，就可以执行下一个任务。任务复制可以分为主动复制和被动复制。主动复制是指在没有发生故障的情况下预先生成副本任务执行。被动复制是指发生错误后再生成任务副本执行。容错调度还可以分为编译时严格调度和运行时通用调度。编译时严格调度是指每个任务要等其前驱任务的所有副本执行完成，才可以开始执行当前任务。与文献[31,33]相似，为了方便

获得可预测的结果，本研究采用主动复制和编译时严格调度。

因此任务$t_{i,k}$的最早开始时间和实际结束时间为

$$\begin{cases} \mathrm{EST}(t_{entry},u_j)=0; \\ \mathrm{EST}(t_{i,k},u_j)=\max\left(avail_{u_j},\ \max_{t_x\in pred(t_i)}(\mathrm{AFT}(t_x)+m_{x,i})\right); \end{cases} \tag{5.12}$$

$$\mathrm{AFT}(t_x)=\max_{k\in[1,n_x]}\{\mathrm{EFT}(t_{x,k})\} \tag{5.13}$$

其中n_x表示任务t_x有n_x个副本，因此应用G的最终响应时间为

$$\mathrm{RT}(G)=\max_{t_i\in T,k\in[1,n_i]}\mathrm{AFT}(t_{i,k}) \tag{5.14}$$

5.1.3 能耗模型

为了评价我们的算法比 GMFRP 算法可以减少能耗，我们仍然采用与上章的相似能耗模型，只是在容错机制下需作出相应的调整。

本文用$E(t_i)$表示第 i 个任务所有副本所消耗的能量，$E(t_{i,j})$表示任务t_i的第j个副本。可由如下公式（5.15）计算，其中$\overline{P_k}$为处理器u_k的平均功率，可由处理器的厂商提供。本文只考虑处理器用于计算任务时所产生的能耗，固定处理器的工作频率。因此整个应用的能耗E(G)可由以下公式（5.17）计算。

$$E(t_{i,j})=\overline{P_k}\times w_{i,k} \tag{5.15}$$

$$E(t_i)=\sum_j^n E(t_{i,j}) \tag{5.16}$$

$$E(G)=\sum_i^{|T|} E(t_i) \tag{5.17}$$

5. 2 可靠性保障及其性能和能耗优化算法

5.2.1 PECORG 算法

文献[35]中提出的 GMFRP 算法分为两个阶段，任务选择阶段和处理器选择阶段。在任务选择阶段选择向上排序值作为任务优先级，采用基于几何平均来计算任务的预分配值。在处理器阶段，选择让任务结束时间最早的处理器作为执行副本的最终处理器，如果可靠性目标没有得到满足，则生成任务副本，再次选择让这个副本结束时间最早的处理器，重复这个过程，直到任务的可靠性目标得到满足。

现有的 GMFRP 算法从原理上无法证明可靠性目标始终可以得到满足，即使每个任务的可靠性目标大于任务目标可靠性。因为任务的可靠性预分配值有可能会大于 1，而实际上任务的可靠性不可能大于 1，所以 GMFRP 算法在某些情况下无法保证可靠性目标得到满足。

现有的 GMFRP 算法在处理器选择阶段还存在不足，将导致资源浪费，从而增加应用调度时间。现举例说明现有 GMFRP 算法的不足。假设现有一个任务需要调度，它的目标可靠性为 0.98。该任务在三个处理器 A、B、C 上的最早结束时间分别为 45、47、40。在这三个处理器上的可靠性分别为 0.987、0.97、0.975。那么根据 GMFRP 算法，该任务应该先选择结束时间最早的处理器 C，因该处理器不能满足可靠性，然后生成该任务的一个副本，选择结束时间最早的处理器 A 执行，此时可靠性目标得到了满足，算法继续调度下一个任务。那么此时很明显，我们只需要把任务调度到处理器 A 上就能满足可靠性目标，且任务的结束时间与 GMFRP 算法一致。

在上述分析启发下，本文提出了一个支持容错的资源高效可靠性保障算法 PECORG 算法，PECORG 算法主要思想是基于几何平均值给未调度的任务分配可靠性，在将任务分配到处理器上时，不但要满足任务的可靠性目标还要满足已分配的所有任务可靠性乘积乘以当前任务可靠性不能小于应用的目标可靠性。在遍历处理器过程中，保存满足任务可靠性目标里结束时间最早的处理器。然后与迭代分配给结束时间最早的处理器方案相比，在结束时间相等前提下，取只有一个任务副本的方案为最终方案。具体算法步骤如算法 5.1 所示：

算法 5.1: PECORG 算法

Input： $U = \{u_1, u_2, \dots, u_{|U|}\}$, G, $R_{goal}(G)$
Output： R(G), RT(G), E(G)
1: 计算应用 G 中的所有任务的 $rank_u$，并根据 $rank_u$ 排序放入优先队列 rank_task_list 中;
2: 根据公式计算 $R_{min}(G), GM(G), E(G)$，并且 $assignedR = 1$;
3: 根据公式计算每个任务的可靠性预分配值 $R_{pre}(t_i)$;
4: for(i=1;i<rank_task_list.size();i++)
5: 取队列中优先级最大的任务 $R_{pre}(t_i)$= rank_task_list.out();
6: 根据公式计算 $R_{goal}(t_i)$;
7: var $eft = \infty, u_{best} = null$;
8: for(each processor $u_k \in U$) do
9: 根据公式计算 $R(t_i, u_k)$;
10: 根据公式计算 $EFT(t_i, u_k)$ 并保存到有序链表 list_EFT 中;
11: if($EFT(t_i,u_k) < eft$ && $R(t_i,u_k) \geq R_{goal}(t_i)$)
12: $eft = EFT(t_i, u_k)$;
13: $u_{best} = u_k$;
14: end if
15: end for
16: $R(t_i) = 0$;
17: while($R(t_i) < R_{goal}(t_i)$ && $R(t_i) \times assignedR < R_{goal}(G)$)
18: 从小到大遍历 list_EFT，将任务副本临时分配到对应处理器上;
19: 根据公式计算 $R(t_i)$，$AFT(t_i)$;
20: end while
21: if($AFT(t_i) \geq eft$)

22: 将任务t_i分配到处理器u_{best}上;
23: else
24: 将上述临时分配作为最终分配结果;
25: end if
26: assignedR = assignedR × $R(t_i)$;
27:end for
28:根据公式计算R(G),RT(G), E(G);

PECORG 算法的具体细节分析如下：

1) 第 1 行计算应用中左右任务的$rank_u$，并按从大到小的顺序保存到 rank_task_list 中。
2) 第 2-3 行根据公式计算$R_{min}(G)$，GM(G)以及每个任务的可靠性预分配值$R_{pre}(t_i)$，并将已分配任务的可靠性乘积assignedR赋初始值 1。
3) 4-6 行根据任务优先级顺序遍历所有任务，并计算任务t_i目标可靠性。
4) 7-15 行遍历所有处理器，计算任务可靠性和最早结束时间，并保存好满足目标可靠性且结束时间最早的那个处理器u_{best}。
5) 16-20 行生成任务副本，将任务临时分配到结束时间最早的处理器上，直到任务的可靠性目标得到满足且满足公式（5.11）。
6) 21-27 行比较任务在处理器u_{best}的最早结束时间 eft 和迭代分配最早结束时间的处理器上的$AFT(t_i)$，如果$AFT(t_i) \geq eft$，则说明迭代分配法结束时间更大，则将任务分配到处理器u_{best}，否则将算法之前的临时分配结果作为最终分配结果。
7) 28 行根据公式计算最终的应用可靠性、响应时间和能耗，并返回结果。

PECORG 算法的时间复杂度分析如下。遍历应用的所有任务的时间复杂度为O(|T|)，遍历所有处理器的时间复杂度为O(|U|)。因此算法的最终时间复杂度为$O(|T| \times |U|)$。

5.2.2 PECORG 算法示例

本例采用图 4.1 所示的应用程序来测试我们的算法，可靠性目标设定为 0.999，这是 GMFRP 算法无法满足的一个可靠性，此时 GMFRP 算法可靠性只有 0.997。三个处理器的故障率$\lambda_1 = 0.0002$，$\lambda_2 = 0.0005$，$\lambda_3 = 0.0009$。表 5.1 为 PECORG 算法下每个步骤的详细情况。图 5.1 所示为 PECORG 算法分配任务的结果。最终应用可靠性$R(G) = 0.99972$，响应时间为 132。

如图 5.1 所示，每个任务都有多个副本，而且副本分布在不同处理器上，这就能大大提高任务的可靠性。

PECORG 算法分配任务的详细情况如表 5.1 所示，以t_1为例，由于t_1在处理器 3 上的最早结束时间最小，因此首先将t_1的第一个副本分配在处理器 3 上执行，但由于$R(t_i, u_3) < R_{goal}(t_i)$，且不满足公式（5.11），因此需要再生成一个任务副

本，分配在处理器 1 上执行，因为此时处理器 1 的最早结束时间最小。

表 5.1 示例 DAG 在 PECORG 算法下的任务分配情况

t_i	$R_{goal}(t_i)$	$R(t_i,u_1)$	$EFT(t_i,u_1)$	$R(t_i,u_2)$	$EFT(t_i,u_2)$	$R(t_i,u_3)$	$EFT(t_i,u_3)$	$R(t_i)$
t_1	0.9999	**0.9972**	**14**	0.9920	16	**0.9919**	**9**	0.99998
t_3	0.9999	**0.9978**	**32**	**0.9935**	**39**	0.9830	45	0.99999
t_4	0.9990	**0.9974**	**45**	0.9960	47	**0.9848**	**40**	0.99996
t_2	0.9990	**0.9974**	**58**	**0.9905**	**58**	0.9839	58	0.99998
t_5	0.9990	**0.9976**	**70**	0.9935	71	**0.9910**	**50**	0.99998
t_6	0.9999	0.9974	83	**0.9920**	**74**	**0.9919**	**59**	0.99994
t_9	0.9990	**0.9964**	**92**	**0.9940**	**95**	0.9821	103	0.99998
t_7	0.9999	**0.9986**	**99**	0.9925	110	**0.9901**	**73**	0.99999
t_8	0.9999	**0.9990**	**104**	0.9945	106	**0.9875**	**103**	0.99999
t_{10}	0.9992	0.9958	135	**0.9965**	**123**	**0.9857**	**132**	0.99995

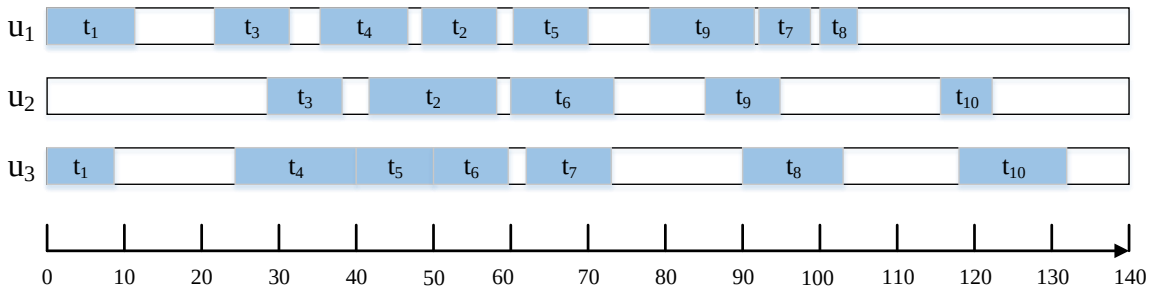

图 5.1 PECORG 算法调度示例

5.3 实验设计与评估

为了评价本文提出的 PECORG 算法，本文使用一个真实汽车应用以及随机生成的应用运行在我们的模拟系统上来测试算法性能。

由于本文是针对汽车电子系统的设计阶段，因此处理器处理每个任务所需要的时间、任务间通信时间，以及处理器的参数采用随机生成的方式。我们选用最新的研究成果 GMFRP 算法与本文的算法比较，因为 GMFRP 算法在论文[35]中已经证明比 HRRM 算法的消耗的资源更少，且 HRRM 存在一定的缺点。本研究目的在于满足高可靠性目标、减少应用响应时间和能耗，因此我们把应用的实际可靠性、应用响应时间和能耗作为评价指标。

5.3.1 汽车应用实验

本节采用跟上章一样的具有 32 个任务的真实汽车应用来进行实验。实验的参数范围：$100ms \le w_{i,k} \le 400\mu s$，$100\mu s \le m_{i,j} \le 400\mu s$，$0.000001/ms \le \lambda_k \le 0.000009/ms$，$30w \le \overline{P_j} \le 200w$，$0.98 \le R_{goal}(G) \le 0.99999999$。处理器个数为 16 个。通过随机生成在上述范围内的不同参数，重复实验 200 次。统计两个算法可以满足可靠性目标

的次数，所得实验结果如图 5.2 所示。

从图 5.2 中可以看出，本文提出的 PECORG 算法当可靠性达到 0.999999 时仍然能 100%满足可靠性指标，而 GMFRP 算法随着可靠性目标提高，可满足可靠性指标的次数逐渐降低。但 PECORG 算法始终能满足可靠性目标。

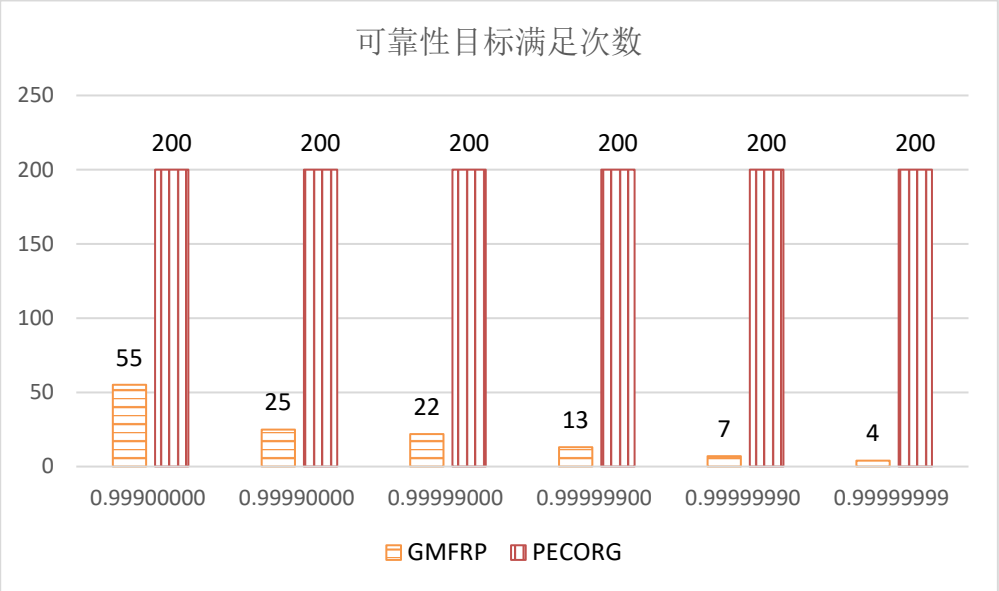

图 5.2 真实汽车应用实验可靠性结果

由于 GMFRP 算法并不能 100%满足可靠性目标，因此本文仅在 GMFRP 可以满足可靠性目标前提下，通过统计响应时间和能耗，计算出响应时间和能耗的平均值，得到如下结果，如表 5.2 所示：

表 5.2 真实汽车应用实验响应时间和能耗结果

可靠性目标	算法	GMFRP	PECORG
0.98	平均响应时间	8134	7932
	平均能耗	9326	9162
0.99	平均响应时间	11458	11447
	平均能耗	11567	11567
0.999	平均响应时间	8134	7932
	平均能耗	9326	9162
0.9999	平均响应时间	11458	11447
	平均能耗	11567	11567

由表 5.2 可知当应用可靠性目标小于 0.9999 时，本文提出的 PECORG 算法平均响应时间和平均能耗都比 GMFRP 算法小，说明本文的算法在性能和能耗上比 GMFRP 算法更加优秀。但是当可靠性目标大于 0.9999 时，PECORG 算法的平均响应时间和平均能耗与 GMFRP 一致，无法继续优化，原因是当可靠性大于 0.9999 时，没有任何处理器的可靠性可以直接满足任务可靠性目标。

5.3.2 随机应用实验

由于未来汽车电子系统将日益复杂，本节采用更加复杂的随机生成的应用来进行实验[16,34,66]，任务个数设定为 500 个，处理器个数设定为 32 个，任务之间的关系和处理器参数都在参数范围内随机生成。

本次实验参数范围：$10ms \le w_{i,k} \le 90ms$，$10ms \le m_{i,j} \le 90ms$，$0.000001/ms \le \lambda_k \le 0.000009/ms$，$30w \le \overline{P}_j \le 200w$，$0.98 \le R_{goal}(G) \le 0.99999999$。

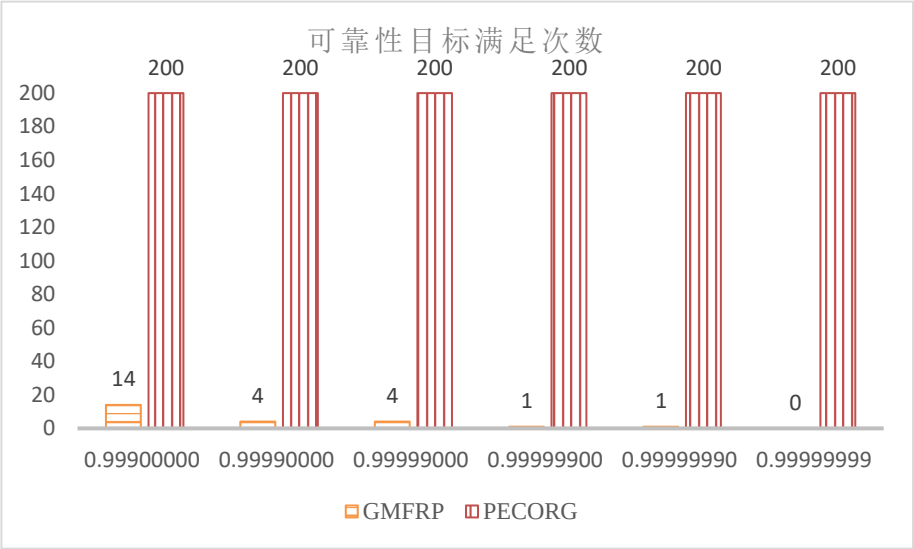

图 5.3 随机应用实验可靠性结果

从图 5.3 中可以看出，与上个汽车真实应用实验相比，在相同可靠性目标前提下，GMFRP 算法满足可靠性目标次数明显减少，原因是随机应用实验的任务个数更多，应用更复杂。本文提出的 PECORG 算法当可靠性达到 0.999999 时仍然能 100%满足可靠性指标，而 GMFRP 算法随着可靠性目标提高，可满足可靠性指标的次数逐渐降低。当可靠性目标达到 0.99999999 时，GMFRP 始终不能满足可靠性目标。

表 5.3 随机应用实验响应时间和能耗结果

可靠性目标	算法	平均响应时间	平均能耗
0.98	GMFRP	29111	6422
	PECORG	28872	4570
0.99	GMFRP	31749	6380
	PECORG	31630	5403
0.999	GMFRP	34249	6634
	PECORG	34187	6542
0.9999	GMFRP	34650	6619
	PECORG	34650	6619

本次实验同样仅在 GMFRP 可以满足可靠性目标前提下，通过统计响应时间和能耗，计算出响应时间和能耗的平均值。由表 5.3 可知，尽管应用的复杂度上升，当应用可靠性目标小于 0.9999 时，本文提出的 PECORG 算法平均响应时间和平均能耗仍然都比 GMFRP 算法小，说明本文的算法在性能和能耗上比 GMFRP 算法更加优秀。同真实汽车应用实验，当可靠性目标大于 0.9999 时，PECORG 算法的平均响应时间和平均能耗与 GMFRP 一致，无法继续优化，原因是当可靠性大于 0.9999 时，没有任何处理器的可靠性可以直接满足任务可靠性目标。

综上所述，本文提出的 PECORG 算法比 GMFRP 算法可以满足更高的可靠性目标，且应用的响应时间和能耗更低。

5.4 本章小结

本章介绍了本文提出的 PECORG 算法，算法分为三个步骤，任务排序、计算任务可靠性目标，生成任务副本确保可靠性目标得到满足。它通过保证当前调度任务的可靠性乘以已分配任务的可靠性乘积不小于应用的目标可靠性，来确保应用的可靠性目标始终能够满足。同时在保证任务最早结束时间不推迟的情况下，通过减少任务的冗余，减少应用响应时间和能耗。最后在真实汽车应用和随机生成的复杂应用实验表明，PECORG 算法比现有的 GMFRP 算法可以满足更高的可靠性目标，且在一定可靠性目标范围内，应用响应时间和能耗更低。

结 论

1．总结

随着计算机技术、通信技术以及人工智能技术的快速发展，无人驾驶越来越被学术界以及产业界所重视。汽车产业信息化程度也不断提升，传统的机械型汽车逐渐发展成高度协同、集成化和智能化的智能汽车。汽车已不再是一个简单的代步工具，无人驾驶汽车更像是一个集代步、娱乐、休息功能于一体的平台。

为了满足人们对汽车安全性、舒适性以及娱乐性的需要，汽车中的功能应用规模不断增长，并从过去的单一功能发展成分布式的功能。自动驾驶技术相关的机器视觉处理与深度学习需要大量的数据采集与处理，给无人车的计算能力带来了压力，同时对硬件成本和能量也进一步增加。道路车辆功能安全—ISO 26262标准对功能安全问题进行了详细的阐述，由此引出功能安全可靠性问题。2018年 12 月版的标准增加了容错相关的安全特性。四次工业革命给汽车工业带来了绿色环保的发展动力，新能源汽车已经成为未来汽车工业发展的方向，而其中又将以纯电动汽车为主，而电池技术的发展远远跟不上计算技术的发展，使得能量资源成为无人驾驶中重要的约束。另外无人驾驶汽车也是一个消费产品，要想进入大众市场，无人车成本必须要降低。因此无人车的设计必须在满足实时与可靠性前提下，尽可能减少硬件成本，对能量进行合理分配与使用。

本文分析了无人驾驶汽车的计算结构与计算平台，计算结构以硬实时，计算任务种类多为特点，计算平台呈现分布式异构多核的趋势，然后从调度的角度，在满足实时性、可靠性前提下，研究分布式功能应用中任务的硬件成本和能量资源优化，并研究了如何通过任务备份容错来提高系统可靠性，减少应用响应时间和能耗，提出了一系列的调度方案，并通过实验进行验证。主要的工作及成果如下：

（1）总结分析了一个集汽车移动端、边缘节点、云端的无人驾驶汽车计算结构，并全面系统的阐述了无人驾驶的计算结构。调查研究了无人驾驶汽车的计算平台。

（2）将无人驾驶汽车电子系统抽象成异构分布嵌入式系统，从任务调度出发，对有关可靠性保障、硬件成本优化、能耗优化相关的调度研究进行详细的总结和归纳，重点分析以 DAG 抽象建模的并行应用调度问题。

（3）研究了无人驾驶汽车中功能应用在有性能和可靠性约束下的硬件成本和能耗优化问题。现有的研究没有将这些约束统筹考虑，因此本文提出结合模拟退火的基因算法的调度算法。通过快速傅里叶变换、高斯消元、真实的汽车电子

功能应用实验，并和现有相关的一些算法进行对比。

（4）面对无人驾驶汽车对高可靠性的需求，本文针对其中计算节点宕机或者随机硬件错误带来的故障。现有的研究存在不能满足高可靠性需求，同时存在资源浪费情况，本文提出了一个基于任务备份的高可靠性保障算法。通过真实汽车应用和随机生成的应用实验，和现有相关算法进行对比，证明了本文提出的算法真实有效。

综上所述，本文针对无人驾驶计算结构，功能安全约束下的硬件成本和能耗优化，可靠性保障的性能和能耗优化这三个方面进行了研究。

2．研究展望

无人驾驶系统是一个非常庞大且复杂的软硬件系统，它涉及许多个技术的集成，无人驾驶系统中还有许多问题需要进一步研究，后续可针对下面几个方面进行：

（1）本文只研究了非容错机制下，在满足实时性和可靠性目标前提下对硬件成本和能耗进行优化。但由于在非容错机制下，不能满足高可靠性的需求，不能解决因计算节点宕机带来的功能失效问题。因此未来可以研究在容错机制下，满足实时性和可靠性目标的硬件成本优化和能耗优化。

（2）本文提出了无人驾驶结合边缘计算的系统结构，通过将无人车上的一些计算任务移到边缘节点上完成，从而降低无人车的能耗。因此将来还可以研究如何通过把计算任务调度到边缘节点上，在保证实时性前提下，减少能耗。同时还可以研究在无人车驶出当前边缘节点服务范围时，怎么保证把计算结果正确，并且实时的返回给无人驾驶汽车。

（3）无人驾驶车中的信息安全研究。关于安全方面，本文主要研究的是系统瞬时故障或计算节点宕机造成的功能可靠性降低。无人车是一个需要和因特网互连的系统，汽车在行驶过程中将时刻与边缘节点、云计算中心、以及周围其它车辆进行信息交互，因此也就面临和传统互联网类似的信息安全问题，外部的信息攻击（如木马病毒、信息篡改等）会影响无人车的正常运行。同时，由于无人驾驶汽车运用了各种不同类型传感器以及异构网络和处理器，为黑客提供了许多的攻击点。因此有必要从信息安全的角度研究无人汽车的安全性。

参考文献

[1] autonomous car, 2018-01-19. https://en.wikipedia.org/wiki/Autonomous_car.

[2] 刘少山,唐浩,吴双,李力耘,第一本自动驾驶技术书,北京:电子工业出版社,2017年6月.

[3] 宋金林.能量感知的 ACPS 调度研究：[湖南大学硕士学位论文]. 长沙： 湖南大学，2017，3-23.

[4] 苏昶玮.探究无人驾驶汽车的发展趋势.通讯世界，2019(2):311-312.

[5] 袁娜.新一代汽车电子系统功能安全可靠性目标保障研究：[湖南大学硕士学位论文]. 长沙： 湖南大学，2017，1-11.

[6] International Organi-zation for Standardization in ISO 26262. "Iso 26262 road vehicles-functional safety",2018.

[7] 刘樑骄. 异构分布式混合关键级系统调度优化研究[湖南大学博士学位论文]. 长沙： 湖南大学. 2016，1-2.

[8] Winner H, Prokop G, Maurer M. Automotive Systems Engineering II || Towards a System-Wide Functional Safety Concept for Automated Road Vehicles，2018, 123-145.

[9] Behere S . A Functional Architecture for Autonomous Driving.In: International Workshop on Automotive Software Architecture. ACM, 2015.

[10] Liu Shao-shan, Tang Jie, Zhang Zhe, et al. Computer Architectures for Autonomous Driving. Computer, 2017, 50(8):18-25.

[11] Mariusz B , Davide-Del T, Daniel D,et al, End to End Learning for Self-Driving Cars ,2018-01-19, http://www.nvidia.cn/object/drive-px-cn.html.

[12] Zeng W , Khalid M , Chowdhury S . In-Vehicle Networks Outlook: Achievements and Challenges. IEEE Communications Surveys & Tutorials, 2017, 18(3):1552-1571.

[13] Topcuoglu H , Hariri S , Wu M Y . Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3):260-274.

[14] S. Ding, J. Wu,”Hybrid Heuristic-Genetic Algorithm with Adaptive Parameters for Static Task Scheduling in Heterogeneous Computing System” 2017 IEEE Trustcom BigDataSE ICESS, 2017.

[15] V. Vyatkin, “Software engineering in industrial automation: State-of-theart review,” IEEE Trans. Ind. Informat, vol. 9, no. 3, pp. 1234–1249,Aug. 2013.

[16] SHATZ, S. M, Wang J. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. IEEE Transactions on Reliability, 2002, 38(1):16-27.

[17] Xie G , Chen Y , Liu Y , et al. Resource Consumption Cost Minimization of Reliable Parallel Applications on Heterogeneous Embedded Systems. IEEE Transactions on Industrial Informatics, 2016:1-1.

[18] Ovatman T . Analysis for Embedded Systems: Experiments with Priced Timed Automata. Electronic Notes in Theoretical Computer Science, 2010, 238(6):81-95.

[19] Qiu M , Sha H M . Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems.. Acm Transactions on Design Automation of Electronic Systems, 2009, 14(2):1-30.

[20] Nogues E , Pelcat M , Menard D , et al. Energy Efficient Scheduling of Real Time Signal Processing Applications through Combined DVFS and DPM.In: Euromicro International Conference on Parallel. IEEE, 2016.

[21] Zhao B , Aydin H , Zhu D . On Maximizing Reliability of Real-Time Embedded Applications Under Hard Energy Constraint. IEEE Transactions on Industrial Informatics, 2010, 6(3):316-328.

[22] Tang Z , Qi L , Cheng Z , et al. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. Journal of Grid Computing, 2016, 14(1):55-74.

[23] Xie G , Jiang J , Liu Y , et al. Minimizing Energy Consumption of Real-Time Parallel Applications using Downward and Upward Approaches on Heterogeneous Systems. IEEE Transactions on Industrial Informatics, 2017:1-1.

[24] 黄晶. 异构多核嵌入式系统的性能与能量优化[湖南大学博士学位论文]. 长沙： 湖南大学. 2018.4-6.

[25] Qiu M, Sha H M. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. Acm Transactions on Design Automation of Electronic Systems, 2009, 14(2):1 – 30.

[26] Mao M, Humphrey M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proc of High

PERFORMANCE Computing, Networking, Storage and Analysis. 2011, 1－12.

[27] Gu Z, Han G, Zeng H, et al. Security-aware mapping and scheduling with hardware co-processors for flexRay-based distributed embedded systems. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(10):3044－3057.

[28] Xie G, Chen Y, Li R, et al. Hardware Cost Design Optimization for Functional Safety-Critical Parallel Applications on Heterogeneous Distributed Embedded Systems. IEEE Transactions on Industrial Informatics, 2017, 14(6):2418-2431.

[29] 谢国琪, 李仁发, 刘琳, et al. 异构分布式系统 DAG 可靠性模型与容错算法. 计算机学报, 2013, 36(10):2019-2032.

[30] Benoit A, Hakem M, Robert Y. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms.In: IEEE International Symposium on Parallel & Distributed Processing. 2008.

[31] Benoit A , Hakem M , Robert Y . Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. Parallel Computing, 2009, 35(2):83-108.

[32] Zhao L , Ren Y , Xiang Y , et al. Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems.In: IEEE International Conference on High Performance Computing & Communications. IEEE, 2011:434-441.

[33] Zhao L , Ren Y , Sakurai K . Reliable workflow scheduling with less resource redundancy[M]. Elsevier Science Publishers B. V. 2013.

[34] Xie G , Zeng G , Chen Y , et al. Minimizing Redundancy to Satisfy Reliability Requirement for a Parallel Application on Heterogeneous Service-oriented Systems. IEEE Transactions on Services Computing, 2017:1-1.

[35] Xie G ,Zhetao L,Na Y,et al.Toward Effective Reliability Requirement Assurance for Automotive Functional Safety. ACM Transactions on Design Automation of Electronic Systems (TODAES).2018.

[36] Shyam Singh Rajput , Virendra Singh Kushwah,“A Genetic based Improved Load Balanced Min-Min Task Scheduling Algorithm for Load Balancing in Cloud Computing”, 2016 8th International Conference on Computational Intelligence and Communication Networks.

[37] Danlami Gabi , Skudai, Johor, “Cloud Scalable Multi-Objective Task Scheduling Algorithm for Cloud Computing Using Cat Swarm Optimization and Simulated Annealing”, 2017 8th International Conference on Information Technology (ICIT).

[38] Zhou Kang, Zhiyi Qu”Application of BP Neural Network Optimized by Genetic Simulated Annealing Algorithm to Prediction of Air Quality Index in Lanzhou”, 2017 2nd IEEE International Conference on Computational Intelligence and Applications.

[39] Zheng S , Shu W , Gao L . Task Scheduling using Parallel Genetic Simulated Annealing Algorithm.In: IEEE International Conference on Service Operations & Logistics. IEEE, 2006.

[40] Wang Z , Cui D . A Hybrid Algorithm Based on Genetic Algorithm and Simulated Annealing for Solving Portfolio Problem.In: International Conference on Business Intelligence & Financial Engineering. IEEE, 2009.

[41] Xin-Hua T , Xu C , Zhi-Feng F . A Multi-objective Genetic Algorithm Based on Simulated Annealing. 2012.

[42] 孟凡超, 初佃辉, 李克秋, et al. 基于混合遗传模拟退火算法的 SaaS 构件优化放置. 软件学报, 2016, 27(4):916-932.

[43] Behere S , Torngren M . A functional architecture for autonomous driving. Information & Software Technology, 2016, 73:136-150.

[44] Zong Wen-hao, Zhang Chang-zhu, Wang Zhu-ping, et al. Architecture Design and Implementation of an Autonomous Vehicle. IEEE Access, 2018, 6:21956-21970.

[45] Wu B, Iandola F, Jin P H, et al. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving.In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops,2016:446-454.

[46] Sun Yi-you, Su Tong-hua, Tu Zhi-ying. Faster R-CNN based autonomous navigation for vehicles in warehouse.In: IEEE International Conference on Advanced Intelligent Mechatronics. IEEE, 2017:1639-1644.

[47] Kim H, Hong S, Son H, et al. High speed road boundary detection on the images for autonomous vehicle with the multi-layer CNN.In: International Symposium on Circuits and Systems. IEEE, 2003:V-769-V-772 vol.5.

[48] Gao Hong-bo, Cheng Bo, Wang Jiang-qian, et al. Object Classification using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment. IEEE Transactions on Industrial Informatics, 2018, 14(9):4224-4231.

[49] Aditya Tewari , Bertram Taetz ,A Probabilistic Combination of CNN and RNN Estimates for Hand Gesture Based Interaction in Car.In:2017 IEEE International Symposium on Mixed and Augmented Reality Adjunct Proceedings.

[50] Girshick R . Fast R-CNN.In: 2015 IEEE International Conference on Com puter Vision (ICCV). IEEE, 2016.

[51] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, 39(6):1137-1149.

[52] 金绍敏. 四轴飞行器的设计与算法研究[中南民族大学硕士学位论文]. 武汉：中南民族大学. 2015，31-33.

[53] Shalev-Shwartz S, Shammah S, Shashua A. Safe,Multi-Agent,Reinforcem ent Learning for Autonomous Driving[EB/OL],2018-01-19.arXiv preprint arXiv:1610.03295.

[54] Kiumarsi B, Vamvoudakis K G, Modares H, et al. Optimal and Autonomo us Control Using Reinforcement Learning: A Survey. IEEE Transactions o n Neural Networks & Learning Systems, 2017, PP(99):1-21.

[55] Chae H, Kang C M, Kim B D, et al. Autonomous braking system via deep reinforcement learning.In: IEEE, International Conference on Intelligent Transportation Systems. IEEE, 2017:1-6.

[56] 夏伟, 李慧云. 基于深度强化学习的自动驾驶策略学习方法. 集成技术, 2017，6(3)：29-40.

[57] Mnih V , Kavukcuoglu K , Silver D , et al. Playing Atari with Deep Reinf orcement Learning. Computer Science, 2013.

[58] Mach P, Becvar Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. IEEE Communications Surveys & Tutorials, 2017, PP(99):1628-1656.

[59] Jian Yu, Bin Fu, Ao Cao, et al.EdgeCNN: A Hybrid Architecture for Agile Learning of Healthcare Data from IoT Devices, .In: 2018 IEEE International Conference on Parallel and Distributed System (ICPADS). IEEE, 2018.

[60] Hongshan Li, Chenghao Hu, Jingyan Jiang, et al,JALAD: Joint Accuracy-and Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution， .In: 2018 IEEE International Conference on Parallel Distributed System (ICPADS). IEEE, 2018.

[61] Liu S , Tang J , Wang C , et al. A Unified Cloud Platform for Autonomous Driving. Computer, 2017, 50(12):42-49.

[62] 邓湘军.异构分布式集群的视频转码与优化：[湖南大学硕士学位论文]. 长沙：湖南大学，2017，18-19.

[63] 谢国琪. 面向汽车的异构网络化嵌入式系统多 DAG 调度研究[湖南大学博士学位论文]. 长沙： 湖南大学. 2014，9-34.

[64] Hascoet J , Jean-François Nezan, Ensor A , et al. Implementation of a Fast Fourier Transform Algorithm onto a Manycore Embedded System.In: DASIP 2015 Conference on Design & Architectures for Signal & Image Processing. IEEE, 2015.

[65] Mladenov T , Nooshabadi S , Kim K . Implementation and Evaluation of Raptor Codes on Embedded Systems. IEEE Transactions on Computers, 2010, 60(12):1678-1691.

[66] 陈月昆.面向工业 CPS 应用的成本优化与设计研究[湖南大学博士学位论文]. 长沙： 湖南大学. 2019,30-60.

附录 A 攻读硕士学位期间发表的学术论文

[1] Wenchao Zou, Renfa Li, Wufei Wu, Lining Zeng. Hardware cost and energy consumption for safety-critical Applications on Heterogeneous Distributed Systems. In: Proc of 2018 IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2018:527-536. (CCF C).

[2] 邹文超，李仁发，吴武飞.适应于自动驾驶的计算结构及其平台综述[J]，计算机工程与科学 ,2019,41(3):130-137.

附录 B 攻读硕士学位期间所参与的项目

[1] 国家自然科学基金 [61672217]：新一代汽车嵌入式系统功能安全的建模与算法研究.

[2] 湖南航天捷诚电子装备有限责任公司：基于AI的雷达自动目标识别系统，2018-2019.

致　谢

研究生三年过得很快。2016 年送别了本科的同学与朋友，我最终还是选择了继续读研。但时间总是过得那么快，转眼之间又到了毕业的季节，即将结束二十年的学校生涯，终于要奔向工作岗位。在此，我要感谢这些年来帮助和鼓励过我的老师、同学和家人，是他们让我的学习生涯不再孤独，并且顺利地完成学业。

首先我要感谢我的导师李仁发教授。李老师渊博的知识、前沿的视野以及务实的学风深刻地影响着我。从论文的选题到再到写作环节，李老师给我提供了非常专业且严谨的指导，实验室例会上的点评为我拨开迷雾，开阔视野，让我能把握住问题的关键所在。生活上，您对学生无微不至的关怀让我们铭记于心，时刻为学生着想，让我们能一心一意地进行学术研究。在此表示对恩师深深地敬意和谢意。

感谢实验室的谢国琪老师。谢老师给我的学术论文中的实验以及撰写提供了非常专业的帮助。同时感谢实验室的吴武飞师兄，感谢师兄每次细心地帮我审稿、提修改意见，使我的论文得到不断完善和补充。

感谢所有在研究生阶段教过我课程的老师，是你们让我补充了许多计算机领域的专业知识，完成了从通信工程专业向计算机专业的转型。

感谢实验室的吴武飞师兄、黄晶师兄、李万里师兄、周佳师兄、白洋师姐，黄一智师兄。博士师兄师姐们都平易近人，不仅在学习上给予我启迪，在生活中也十分关照，让我感受到实验室的温暖。感谢实验室同门关俊杰、汪继龙、马温红、李桂芬、何妍、杨远达、段宇、张开厂、彭浩、陈毅杰，和你们一起学习让我倍感快乐。感谢实验室一群带我打乒乓球的小伙伴，是你们让我的学习生涯丰富多彩。特别重点感谢王栩同学对我的关心和大力支持。

特别感谢我的父母以及亲戚朋友，你们对我无私的爱与支持让我不断前进，让我顺利地完成学业，我将永远记得你们为我付出的心血。

最后衷心感谢评审专家以及答辩组老师们在百忙中对本文的审阅与指导。