2024 华为智联杯·无线程序设计大赛

亲和任务调度系统——任务书

文档版本	V0.1
发布日期	2024-05-31

华为技术有限公司

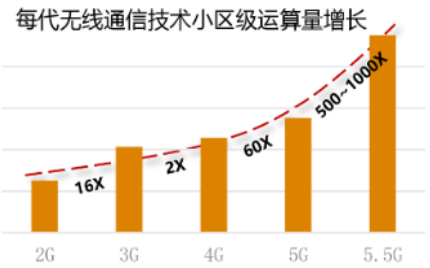
目 录

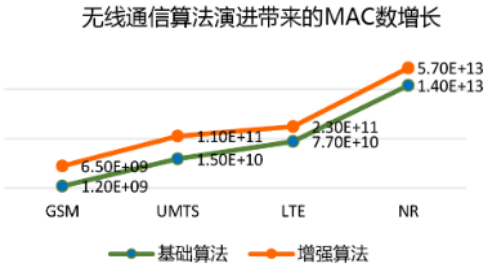
1 修订记录

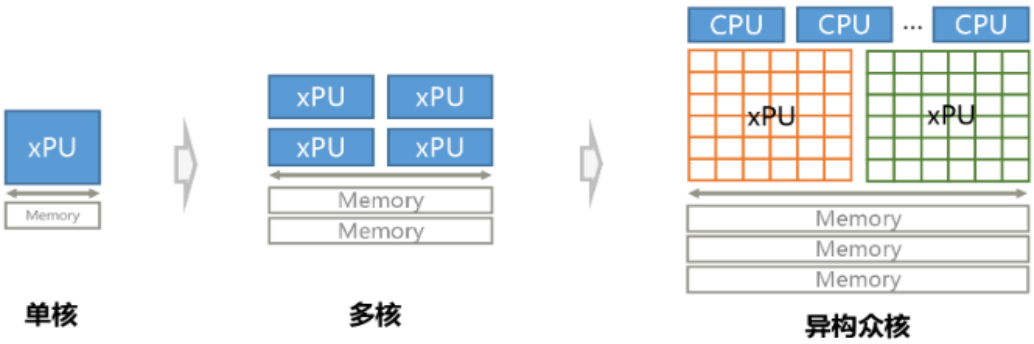
版本	修改说明	发布时间
V0.1	初稿	2024-05-31

2 背景信息

随着物联网、大数据、AI 时代的到来，时延、可靠性等指标要求越来越高，海量的数据分析、大量复杂的运算对 CPU 的算力要求越来越高。CPU 内部的大部分资源用于缓存和逻辑控制，适合运行具有分支跳转、逻辑复杂、数据结构不规则、递归等特点的串行程序。在集成电路工艺制程将要达到极限，摩尔定律快要失效的背景下，基站系统芯片架构从单核演变到多核、众核时代。







在已经给定的复杂硬件架构下，如何确定整个系统的任务调度，软件如何实现最优的任务调度和资源分配、并将控制开销降到最低以趋近上限，越来越难以通过人工分析获得结果，需要通过数学建模和理论分析，辅助 AI 等手段，寻找上限和优化方向。面向未来业务演进的芯片设计，基于无线业务特征的调度策略演进方向、对应的硬件架构设计的方向，需要系统的理论分析来指导。

在无线领域，利用 AI 技术对任务准确建模、多核系统任务最优调度等问题都是非常有价值的算法难题。本次比赛通过软件模拟了多核运行系统，由选手来挑战这些有价值的软件难题。

期待您的精彩解决方案！

3 题目

3.1 术语

名词	解释
机器	执行任务的核
工作	由一个或多个任务组成的序列
任务	可以在机器上执行的最小单元
亲和性	核连续处理的任务类型如果不同，可能会导致核频繁切换加载不同指令，导致 CacheMiss 提高，降低核的处理效率

3.2 题目信息

3.2.1 问题抽象

基站作为多核任务处理系统，既要最追求任务处理能力，也要追求处理效率。

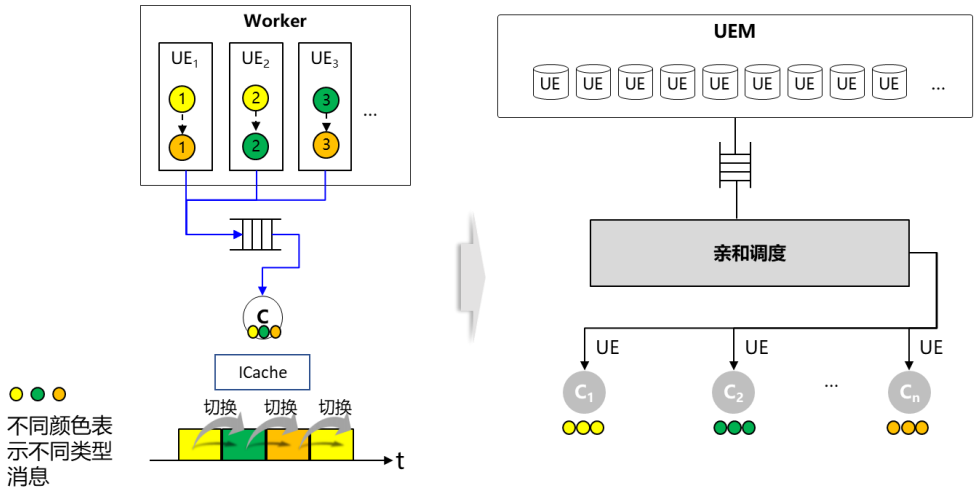
在基站系统中，共有 M 台机器和 N 项任务待处理，任务定义为= <*MsgType, UsrInst, ExeTime, DeadLine*>，携带以下几项信息：

- *MsgType：基站处理的任务类型，范围*$0 \le MsgType \le 200$
- *UsrInst：基站接入的用户实例号，范围*$0 \le UsrInst \le 10000$
- *ExeTime：处理本任务耗时，简化为整数，其范围*$1 \le ExeTime \le 4000$*；*
- *DeadLine：任务进入系统即开始计时，在此之前需要处理完，其范围* $1 \le DeadLine \le 4294967295$*;*
- *保证<MsgType, UsrInst>唯一，即每条任务都是独一无二的*

你需要将 N 个待处理的任务，依照时间顺序分配到 M 个机器上。分配的约束要求包括：

- 同一用户实例的任务顺序不能乱序，只有当某任务的前序任务均完成时，该任务才可以启动
- 一个机器同时只能处理一个任务
- 当一个任务启动时，必须执行完毕
- 不同核的执行时序无法保证，核内的执行时序能保证为:先进先出，即同一个用户的任务不能分配到不同的核上，因为无法保证执行时序
- 必须完成所有任务的分配

核在执行任务处理任务时，如果任务类型不断变更，则核需要不断切换加载不同指令，导致 CacheMiss 提高。



[注] 不同色块表示处理不同消息的指令块

- 不断切换加载不同的指令,iCahce Miss较高
- 优先调度与核有亲和关系的任务（消息处理），减少CacheMiss

在满足以上约束条件的情况下，任务分配结果才被认定为可行解。在可行解的前提下，尽可能提高任务调度的整体效率，并提高核处理任务的亲和性以降低 CacheMiss。

3.2.2 赛题任务

3.2.2.1 任务场景分类

在实际抽象建模分析的过程中，调度系统中任务的耗时实际上是通过估测得到的。实际做法是根据通信信号数据完成场景分类，依据分类结果辅助经验数据，来确定每一条任务的真正处理耗时。而在这个处理过程中，场景分类的信号检测和处理是非常复杂的，随着 AI 技术的进一步发展，AI 将有机会使能无线通信的软件性能优化，提高基站系统的处理性能。你需要完成以下 AI 分类任务：

- **线下训练任务：**在本赛题中，将提供 5400 条通信信号和其对应的业务场景，业务场景共 11 种。你需要利用这 5400 条数据(带标签)作为训练集训练自己的 AI 模型，尽可能的提高模型场景预测准确率。

- **线上推理任务**：在比赛期间，将给出若干条通信信号(无标签)，你需要利用你的AI 模型得到这些通信信号的业务场景分类结果，并在赛事平台提交时，同时上传**分类结果+模型文件+模型定义+训练/推理源代码**。

在本任务中，根据通信信号数据进行场景分类是非常重要的。对于任务而言，若任务所属的业务场景分类错误，将导致业务处理中的逻辑分支差异较大，影响任务的处理耗时估计，从而影响多核系统的整体调度。在本题中，为了简化问题抽象如下：

- 每条任务归属于唯一业务场景，编号[0, 10]，赛题输入中将不会给出任务的业务场景归属
- 每条通信信号样本归属于唯一业务场景
- 选手设计的 AI 模型预测准确率将**直接影响所有任务的耗时预估**。

例如存在一个任务 Msg = <2, 1, 500, 2000>，代表该任务类型为 2、归属用户 1、执行时间为 500、最晚完成时间为 2000。如果 AI 模型预测准确率为 95.5%，则选手在 3.2.2.2 最优任务调度环节，获取到的输入信息中，所有任务的耗时都将提高。Msg 任务的输入信息将更正为 Msg = <2, 1, 544, 2000> (解释：500 * (2 – 95.5% * 95.5%) = 543.98≈544, 四舍五入)。

因此，请尽可能的优化你的 AI 模型，提高模型的预测准确率，来降低对“最优任务调度”环节的负面影响。

3.2.2.2 最优任务调度

在此环节中，你将设计多核调度器，将输入任务分配到不同核上。任务信息如下：

- **输入**

长度为 N 的任务序列，N 的范围为$1 \le N \le 100000$，每条任务定义见 3.2.1 释义，其中任务的耗时字段将受你的 AI 模型准确率影响；

给定 M 个核， M 的范围为$1 \le M \le 30$；

给定系统最大处理时长 C，其范围为$1 \le C \le 4294967295$；

- **输出**

按格式输出每个核上的任务分配顺序结果

Core[0] :=TaskNum, <MsgType, UsrInst>, … , <MsgType, UsrInst>

…

Core[M-1] := TaskNum, <MsgType, UsrInst>, … , <MsgType, UsrInst>

- **优化目标**

- **亲和性评价(AffinityScore)**：影响处理效率。当一个 Core 连续处理两条相同类型的任务时亲和性较好，亲和得分+1，如出现连续处理多条相同类型任务时，分别计算每两个相邻任务的亲和得分并累加。
- **处理能力评价(CapabilityScore)**：处理能力，给定系统最大处理时长 C 以及每个任务的最晚结束时间，处理不超时的任务总数即为处理能力得分。

基于以上评价维度，选手单个用例的得分评价公式定义为：

$$\text{选手得分} = \textit{亲和性得分} + \textit{未超时的任务数}$$

为了平衡各个用例集规模导致的得分差异，需要进行归一化处理。针对此评分公式，我们可以得出一个绝对上界：

$$\text{绝对上界} = \textit{任务数}N + \textit{任务数}N$$

基于此进行得分的归一化和规格扩充，选手单用例的最终得分评价为：

$$\text{单个用例调度得分} = 100000 * log10(N) * \left(1 + \frac{\textit{选手得分} - \textit{绝对上界}}{\textit{绝对上界}}\right)$$

4 输入说明

4.1 任务场景分类输入

训练集会给出 11 种场景的通信信号，共 5400 条样本数据。

- **命名规范：**训练集给定数据命名规则为 Data_label_*_.bin, 其中*为对应场景的 label 编号。相同的 label 表示数据来源为相同的场景，不同的 label 表示数据来源为不同的场景。
- **文件格式：**二进制文件
- **数据格式：**CFLOAT16 * 15360，每个四字节代表一个复数点，高 16bit 为实部，低 16bit 为虚部，数据格式为 ieee 标准的 float16

4.2 最优任务调度输入

线上评测时，输入数据从标准输入读入。格式如下：

- **第 1 行：** 共 3 个整数，分别代表任务个数 N，核的个数 M，最大处理时间 C；
- **第 2~N+1 行：**每行 4 个整数，代表任务携带的信息<*MsgType, UsrInst, ExeTime, DeadLine*>，释义见 3.2.1，**其中 *ExeTime* 字段值受选手“任务场景预测任务”准确度的影响，不同选手获取到的值可能不一致**。

一个例子如下：

```
7 3 100
1 1 5 10
1 2 5 22
2 1 6 8
2 2 6 9
2 3 11 40
3 4 25 30
3 1 12 32
```

5 输出说明

5.1 任务场景分类输出

练习赛(线上)将提供 2800 条测试样本用于调测

正式赛(线下)将提供新的 2800 条测试样本用于评测

请将对每个样本的业务场景预测结果依次输出到 result.csv 文件中，每个样本占用一行，如：

```
1
2
3
5
2
0
11
…
…
…
…
…
8
7
1
0
```

请将生成的预测结果 csv 文件与模型文件+模型定义+训练/推理源代码，与“最优任务调度”的源代码，一起打包上传评测，模型源代码要求可复现预测结果，上传示例请

见赛事页面相关附件。提交判题时，若 csv 文件行数不匹配、csv 文件无法解析等错误，则准确率直接计为 0%。

5.2 最优任务调度输出

输出共 M 行，代表每个核上分配的任务二元组信息

每一行第一个整数代表该核上分配的任务数 N，紧接着有 2*N 个整数，每两个整数为一组，代表唯一标识任务的<任务类型，用户实例号>

Core[0] :=TaskNum, <MsgType, UsrInst>, … , <MsgType, UsrInst>

…

Core[M-1] := TaskNum, <MsgType, UsrInst>, … , <MsgType, UsrInst>

以下是 4.2 中提供的输入的一种可能的输出结果(共 3 个核，每行依次输出核上分配的任务)：

```
5 1 1 1 2 2 1 2 2 3 1
1 3 4
1 2 3
```

调度结果必须保证 3.2.1 中描述的约束可行性，如不可行则得分计 0 分。以下给出可行性约束的两组示例：

样例 1：调度 OK(同一用户实例任务顺序不能乱序，不同用户实例任务可乱序)

输入任务序列：<<1, 2,1,50>, <2, 2,1,50>, <3,1,1,50>, <2,1,1,50>>

Core0 执行 <1, 2,1,50>, <2, 2,1,50>

Core1 执行 <3,1,1,50>, <2,1,1,50>

样例 2：调度不 OK(不同 Core 的执行时序无法保证，Core 内的执行时序能保证为:先进先出)

输入任务序列：<<1, 2,1,50>, <2, 2,1,50>, <3,1,1,50>, <2,1,1,50>>

Core0 执行 <1, 2,1,50>

Core1 执行 <3,1,1,50>, <2,1,1,50>, <2, 2,1,50>

6 评分规则

1. 判题程序会从选手程序标准输出读取分配方案，计算调度总得分并记录程序运行时间（单位为 ms），总得分高的方案胜出。
2. 如果不同选手的输出方案的总得分相同，则先提交代码者胜出。
3. 判题采用多组数据，得分依据多组结果求和后进行排名。
4. 对于单个用例，选手的程序所有计算步骤（包含读取输入、计算、输出方案）所用时间总和**不超过 4 秒**。若程序运行超时、运行出错或输出不合法的解（包括调度分配方案不满足题目约束或解格式不正确），则判定无成绩。
5. **禁止在代码中执行 shell 命令、使用多线程**等影响判题机器运行与公平性的行为。此行为在赛后的最终测评阶段也将无法得分！
6. 比赛结束后将进行代码查验，如发现代码重复或违规等情况，将取消该团队参赛资格和现有成绩。

⚠ 注意

因为进程调用存在一定的时间开销，用时统计在判题程序侧和选手程序侧可能存在细微差异。建议选手控制算法用时的时候要留有一定的冗余

---

7 附录

7.1 AI 读取输入数据的示例代码(Python)

```
def preprocess(folder_path):
    labels = []
    data = []
    cnt = 0
    for filename in os.listdir(folder_path):
        cnt += 1
        if filename.endswith(".bin"):
            match = re.search(r'label_(\d+)_', filename)
            if match:
                label = int(match.group(1))
            else:
                continue
            with open(os.path.join(folder_path, filename), 'rb') as file:
                data_row_bin = file.read()
                labels.append(label)
                data_row_float16 = np.frombuffer(data_row_bin, dtype=np.float16)
                data_row_float16 = np.array(data_row_float16)
                data.append(data_row_float16)

    return data, labels
```