基于 RISC-V 的人工智能处理器描述语言雏形设计 - 附录 1

佚名

2025 年 7 月 24 日

附录 1：异构多核支持：扩展对多核协同和任务调度的描述

随着人工智能（AI）应用复杂性的增加，单一核心处理器难以满足高性能、低功耗和多样化计算需求。异构多核架构通过集成不同类型的核心（如整数核心、向量核心、专用 AI 加速核心）实现任务并行和性能优化。基于 RISC-V 的人工智能处理器描述语言（PDL）通过扩展支持异构多核架构，提供了对多核协同和任务调度的描述机制。本附录详细阐述了 PDL 如何描述异构多核处理器，包括核间通信、同步机制和任务调度策略，以满足 AI 工作负载的需求。

0.1 异构多核架构描述

异构多核架构由多种类型的核心组成，每种核心针对特定任务优化（如通用计算、向量运算、AI 加速）。PDL 通过 `HeterogeneousProcessor` 关键字定义多核架构，支持指定核心类型、数量和互连拓扑。其语法格式如下：

```
HeterogeneousProcessor <name> {
    cores: [<core_type1> x <count>, <core_type2> x <count>, ...]; // 核心类型和数量
    interconnect: <topology>; // 互连拓扑，如Mesh、Ring
    sharedMemory: <memory_config>; // 共享内存配置
    communication: <protocol>; // 核间通信协议
}
```

例如，定义一个包含 2 个整数核心、1 个向量核心和 1 个 AI 加速核心的异构处理器，采用 Mesh 互连拓扑：

```
HeterogeneousProcessor AIHeteroProc {
    cores: [IntegerCore x 2, VectorCore x 1, AICore x 1];
    interconnect: Mesh;
    sharedMemory: { size: 1MB, accessLatency: 5 };
    communication: MessagePassing;
}
```

该描述指定了核心组成、互连方式和共享内存配置，便于后续任务分配和协同优化。

0.2 核间通信机制

核间通信是异构多核架构的关键，PDL 支持以下通信机制：

- **共享内存**：通过共享缓存或主内存实现数据交换。
- **消息传递**：通过专用通道传递数据或指令。
- **同步机制**：如锁、信号量，确保数据一致性和任务协调。

PDL 通过 `InterCoreCommunication` 关键字定义通信机制，语法格式如下：

```
InterCoreCommunication <name> {
    type: <comm_type>; // 通信类型：SharedMemory, MessagePassing
    latency: <cycles>; // 通信延迟
    bandwidth: <rate>; // 带宽，单位为GB/s
    synchronization: [<sync_mechanism>]; // 同步机制，如Lock, Semaphore
}
```

示例：定义一个基于共享内存的通信机制，配合锁同步：

```
InterCoreCommunication SharedMemComm {
    type: SharedMemory;
    latency: 10;
    bandwidth: 32;
    synchronization: [Lock, Barrier];
}
```

该机制支持多核间高效数据共享，锁和屏障确保同步正确性。

0.3 任务调度策略

任务调度决定如何将 AI 工作负载分配到不同核心，以优化性能和能效。PDL 支持静态和动态调度策略：- **静态调度**：在设计时指定任务到核心的映射，适合固定工作负载。- **动态调度**：运行时根据负载和核心状态动态分配任务，适合复杂 AI 应用。

PDL 通过 `TaskScheduler` 关键字定义调度策略，语法格式如下：

```
TaskScheduler <name> {
    type: <scheduler_type>; // Static, Dynamic
    policy: <scheduling_policy>; // 调度策略，如RoundRobin, PriorityBased
    tasks: [<task_type>]; // 支持的任务类型，如MatrixMul, Convolution
    coreMapping: [<core_type> -> <task_type>]; // 核心-任务映射
}
```

示例：定义一个动态调度器，优先分配矩阵乘法任务到向量核心：

```
TaskScheduler AIScheduler {
    type: Dynamic;
    policy: PriorityBased;
    tasks: [MatrixMul, Convolution, Activation];
```

```
    coreMapping: [VectorCore -> MatrixMul, AICore -> Convolution];
}
```

该调度器根据任务优先级动态分配，优化 AI 任务的执行效率。

0.4 多核协同优化

为实现高效的多核协同，PDL 支持以下优化机制：

- **负载均衡**：通过调度器分配任务，确保核心利用率最大化。
- **数据局部性**：优化共享内存访问，减少跨核数据传输。
- **低功耗设计**：支持动态电压频率调整（DVFS），降低空闲核心功耗。

示例：定义一个支持负载均衡和 DVFS 的协同机制：

```
MultiCoreOptimization OptConfig {
    loadBalancing: Enabled;
    dataLocality: CacheAware;
    powerManagement: DVFS { minFreq: 800MHz, maxFreq: 2GHz };
}
```

该配置通过缓存感知的数据分配和动态频率调整优化性能和能效。

0.5 应用示例

以下是一个完整的异构多核处理器描述，展示多核协同和任务调度的应用：

```
HeterogeneousProcessor DeepLearningProc {
    cores: [IntegerCore x 2, VectorCore x 2, AICore x 1];
    interconnect: Mesh;
    sharedMemory: { size: 2MB, accessLatency: 6 };
    communication: SharedMemComm;
    scheduler: AIScheduler;
    optimization: OptConfig;
}

InterCoreCommunication SharedMemComm {
    type: SharedMemory;
    latency: 8;
    bandwidth: 64;
    synchronization: [Lock, Semaphore];
}

TaskScheduler AIScheduler {
    type: Dynamic;
    policy: PriorityBased;
```

```
    tasks: [MatrixMul, Convolution];
    coreMapping: [VectorCore -> MatrixMul, AICore -> Convolution];
}

MultiCoreOptimization OptConfig {
    loadBalancing: Enabled;
    dataLocality: CacheAware;
    powerManagement: DVFS { minFreq: 1GHz, maxFreq: 2.5GHz };
}
```

该示例描述了一个深度学习处理器，包含 2 个整数核心、2 个向量核心和 1 个 AI 核心，通过动态调度和共享内存通信实现高效协同，优化矩阵乘法和卷积运算的执行。

0.6 实现与验证

异构多核支持的实现依赖于 PDL 与 ANTLR 和 MLIR 的集成：- **ANTLR 解析**：扩展语法规则以支持 `HeterogeneousProcessor` 和 `TaskScheduler` 等关键字。例如：

```
grammar AIPDL;
heterogeneousProcessor: 'HeterogeneousProcessor' ID '{' (cores |
    interconnect)* '}';
taskScheduler: 'TaskScheduler' ID '{' (type | policy | tasks)* '}';
```

- **MLIR 优化**：将多核描述转换为中间表示，优化任务分配和通信逻辑，生成高效的 Verilog 代码。- **验证方法**：- 语法验证：测试多核描述的正确性和错误用例。- 功能仿真：使用 SystemC 仿真多核协同，验证矩阵乘法任务的分配正确性。- 性能评估：测量任务调度效率，确保核心利用率 $>90\%$，通信延迟 <10 周期。

0.7 总结

本附录扩展了 PDL 对异构多核架构的支持,通过 `HeterogeneousProcessor`、`InterCoreCommunication` 和 `TaskScheduler` 等机制，实现了多核协同和任务调度的灵活描述。这些机制支持共享内存、消息传递和动态调度，优化了 AI 工作负载的性能和能效。未来可进一步扩展对实时 AI 任务和边缘设备低功耗场景的支持，推动 PDL 在复杂 AI 处理器设计中的应用。