DFC ver0.2 软件任务小组
任务分解

2020-10-31

1. 任务描述：

DFC 编译及运行环境进行改进增强：

- ✧ （黄义凯）动态构图能力；
- ✧ （李克钹）分层图形输入前端工具；
- ✧ （文成元）多文件、分层编译、多机任务分发。
- ✧ （李泽铨）添加调度算法。
- ✧ （王树泉）

测试？gcc dejagun llvm lit 自动化测试用例

2. Ver 0.2 概述

2.1. DFC ver0.1 构图方案

原来的构图方式包括两个内容：

1、 声明全局性的主动数据（图中的边）；

2、 声明数据流函数（图中的节点）。

它们的关系在原代码编写的时候就已经确定，而且所生的的代码中函数与数据之间的绑定已经无法改变。

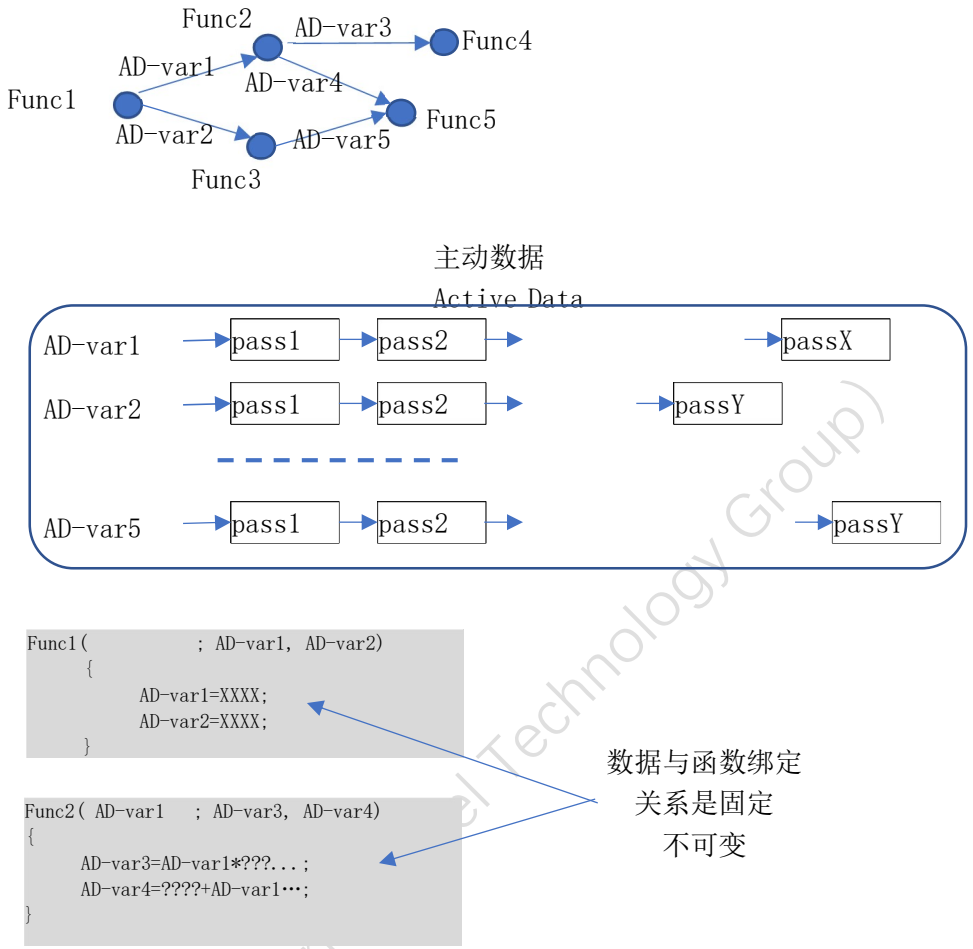


图 2-1 DFC ver 0.1 版本的构图方案

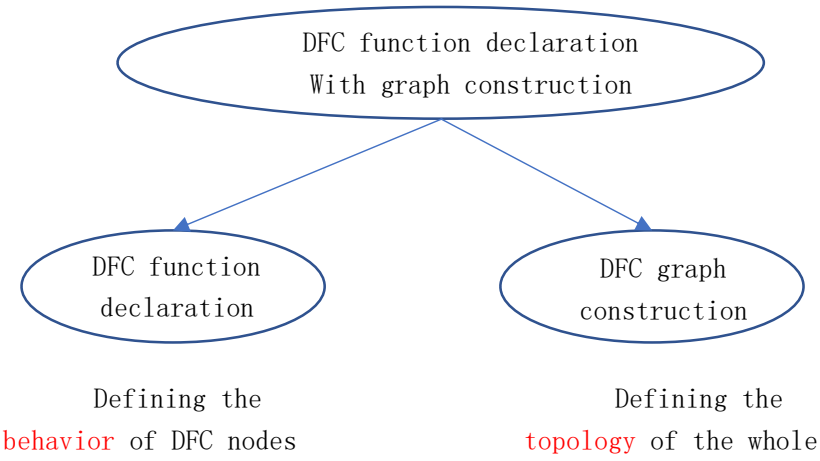
2.2. 动态构图能力

编译器生成的代码中，需要有一个图的描述数据作为中间，该结构需要成为一个可以动态修改的图结构。

2.2.1. 核心修改——节点功能和构图的分离

通过语法扩展，将 DFC ver0.1 的 DFC 函数同时负责函数声明和图的构建，两个功能。Ver 0.2 将下面两个概念进行区分，从而获得更灵活的构图能力：

2. 数据流函数的定义；
3. 图的定义进行区分；



因此，需要增加一个新的语法元素，勇于从源代码创建“初始”计算图的能力。此时，源代码仍具有对主动数据的“类型声明”和专用的构图语法构件——“#pragma DFC_GRAPH”。

2.2.2. DFC ver0.2 动态构图方案

动态构图方案包括两个方面的内容：

1）如何表示拓扑中的边和节点的关系；

2）如何改变这些关系；

可变拓扑的表示

虽然 DFCver0.2 的初始拓扑可以通过原代码制定，但是它们的连接拓扑关系，使用一个动态创建的“中间”数据结构来表示。主动数据和节点函数各自都通过一个“实例”列表来实现——从而可以获得动态的映射能力。具体包括：

1） 主动数据实例链表，其中每个元素取代原来一个主动数据；

2） 节点函数实例列表，其中每个元素取代原来一个节点 dfc 函数；

仍以上面的例子来讨论，此时的主动数据加入一个“引用”列表，函数也就入一个“构图”表。

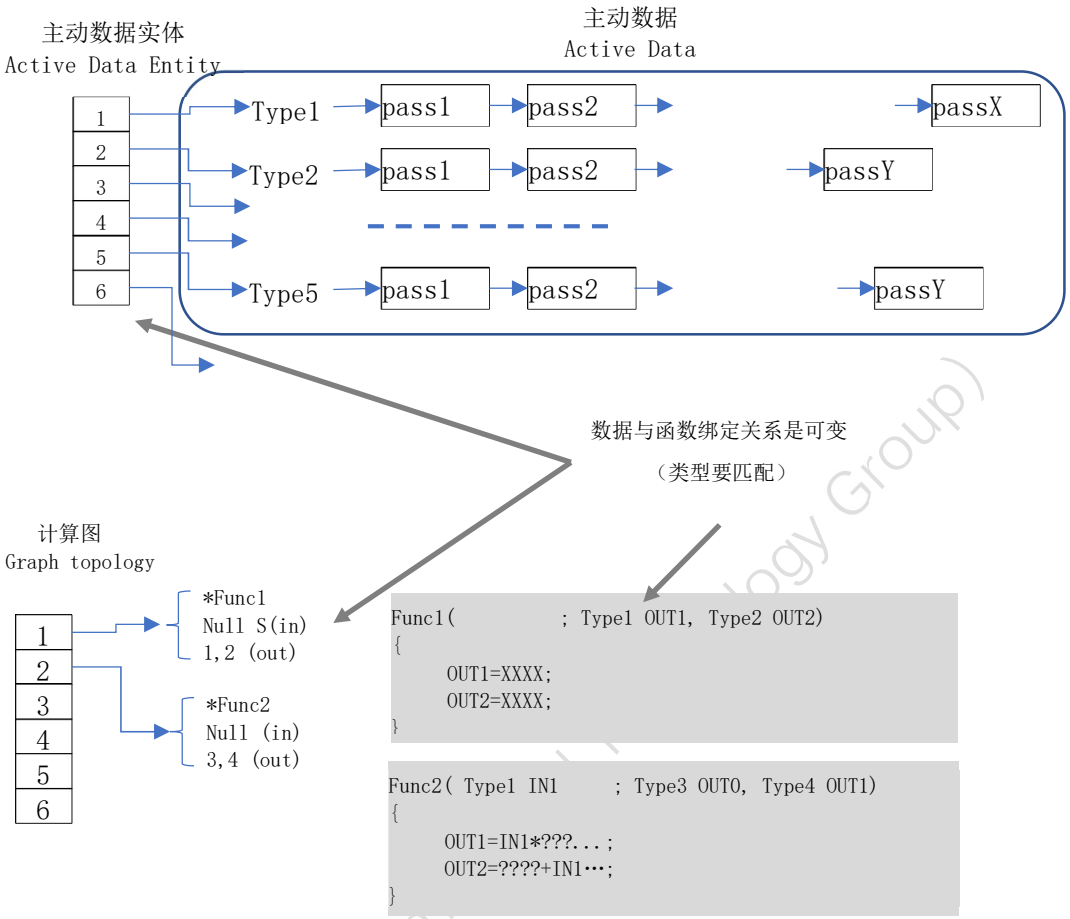


图 2-2 DFC ver 0.2 **动态构图方案——动态映射**

构图 API（Runtime）

编译器在变换构图源代码的 AST 时，需要借助 Runtime 提供的服务，建立主动数据实例列表和函数实例列表。对应的有函数：

1. 增加两个函数 DFC_ADD_AD()用于插入一个新的主动数据实例，其类型实现有的类型。反之，需要一个 DFC_DEL_AD()用于删除一个主动数据实例。类似地有插入节点函数 DFC_FUNC_INSERT()和 DFC_FUNC_DELETE()。
2. 需要增加新的节点函数，则需要进行注册。提供 DFC_REGISTER_AD()和 DFC_DISMISS_AD(),对于函数有 DFC_REGISTER_FUNC()和 DFC_DISMISS_FUNC()。

修改拓扑关系

修改拓扑关系，表现为一个 API 函数集。下面先分析所需的拓扑修改操作，以及实现方案。修改拓扑关系具有一个约束条件：

■ 修改的子图输入输出部分不变，只运行改变内部结构。

修改过程分成两个步骤：

1） 删除子图；
2） 新建替代用的子图；
为了修改拓扑关系，将使用“构图 API”完成。

*分布式与分层问题

分布式支持

这四个函数根据系统的状态不同，需要区分为单机和分布式两种版本，使用函数指针列表的方式加以初始化和引用。

```
struct Graph_setup_ops{
        DFC_ADD_AD();
DFC_DEL_AD();
DFC_REGISTER_AD();
DFC_DISMISS_AD();
}
```

区分为两种函数，其名字前缀为 sp_DFC_XXXXXX 和 mp_DFC_XXXX,两组不同的函数。
同理，主动数据列表和函数列表将加入子图分配信息,每个实例上面都需要记录所属处理器——此例有两个处理器 c0 和 c1：

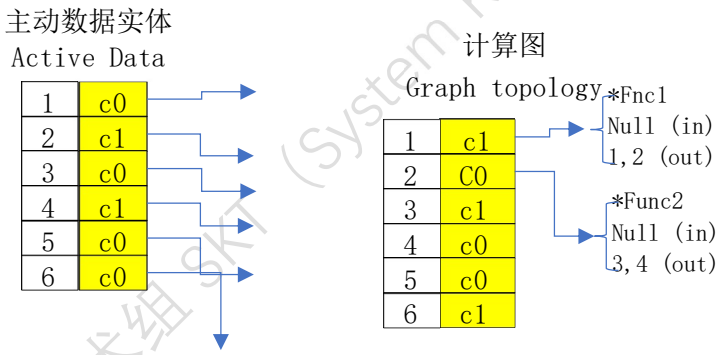


图 2-3 带有子图划分属性的主动数据和节点函数

多层/子图支持

动态构图过程中，可以先考虑子图插入问题。毕竟子图插入和主动数据+函数的插入方式本质相同，只属于功能封装。

2.3. 分层描述

提供独立的语法结构或独立文件

2.3.1. 顶层描述

比如一个顶级描述 top.dfc 或者起名为专用文件 top.graph，内含子图。

```
#pragma DFC_GRAPH
{
    声明  AD1;AD2;

    DFC_func1( AD1-1, AD1-2…  ; AD1-i, AD1-i+1…  );
    DFC_func1(                                   );
//同一函数可以出现多次，每一次出线的实例都代表一个节点
    DFC_func2(                )
    …
    DFC_funcX(                )

DFC_SUB_GRAPH subgraph-XXX( AD…            ;AD…            );
}
```

编译之后，上述的主动数据，都被转换成主动数据的注册函数，插入到数据流的初始操作中；同理 DFC 函数也将通过 runtime 中的函数注册插入到系统中。

对于每一个子图，则变换成子图注册函数（子图注册函数）

2.3.2. 子图描述

子图文件 subgraph-XXX.dfc，内容如下：

```
#pragma DFC_SUB_GRAPH(  AD…        ;AD…       )
{

}
```

子图描述经过转换后，成为子图注册函数（在顶层图代码中调用），内部为主动数据的注册和 DFC 函数的注册操作。

2.4. 分布式支撑

DFC ver 0.1 的 runtime 局限于单一系统/单一 OS 内，以共享内存的多线程方式在工作。Ver 0.2 将支持分布式运行环境。需要实现以下几个功能：

1） 集群信息管理，维护通信域
2） 子图划分（涉及划分算法，将来的研究生选题）
3） AD 数据的传送

2.4.1. 集群管理

集群管理功能包括：

1） 主机基本信息收集；

2) 通信基础及相关协议

收集主机信息，可以利用现有工具，比如 clush 或自行设计一套规则，登记必要信息——例如主机 IP、主机名、处理器类型和数量、物理内存总量和 NUMA 拓扑等。

采用集中式管理，设立管理节点，接受计算节点的接入和登记。子图划分后，分析出那些节点将具有通信需求，并将上述信息发布到具体节点上，然后又具体节点按照前后依赖关系创建 TCP 连接。

2.4.2. 子图分割

要能对子图进行分割，然后一个应用程序可执行文件，其 runtime 可以有选择地运行一部分数据流任务。子图分割的机制和算法作分离，分割作为机制是稳定不变的基础，算法根据优化目标的不同而可以选择不同的算法——当前算法近用作示例。分割包括两部分：

1) 子图划分还包括对应的数据的划分，因此系统启动时，需要将边界主动数据和普通主动数据进行区分管理。

2) 节点函数只需要列出子图内的函数即可，与单机没有区别。

除了图 2-3 给出的在主动数据和节点函数上进行子图划分标记以外，可以单独构件一个本子图所涉及的主动数据和节点函数的快速索引，以便于编程或阅读。

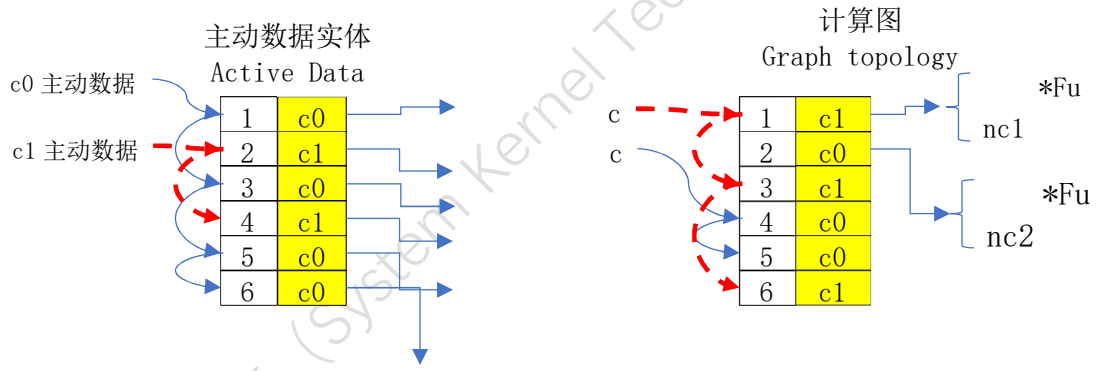


图 2-4 子图划分辅助数据结构

2.4.3. AD 数据传送

为了跨主机节点之间传递 AD 数据，需要在各主机之间建立起通信链路。根据节点间是否存在数据传送而建立 TCP 连接。

在主动数据的读取操作函数中（ver0.1 是什么？）将区分边界数据和非边界数据，边界数据的操作将和本地主动数据有所不同

1) 边界 Input AD，将使用网络进行接受

2) 边界 Output AD，将使用网络进行传送

2.5. 用时测量

李锦荣，补充