北京航空航天大學
BEIHANG UNIVERSITY

端到端的跨平台科学计算软件框架

报告人：韩斌 龚煜涵

2024年8月19日

1 背景

硬件架构的发展

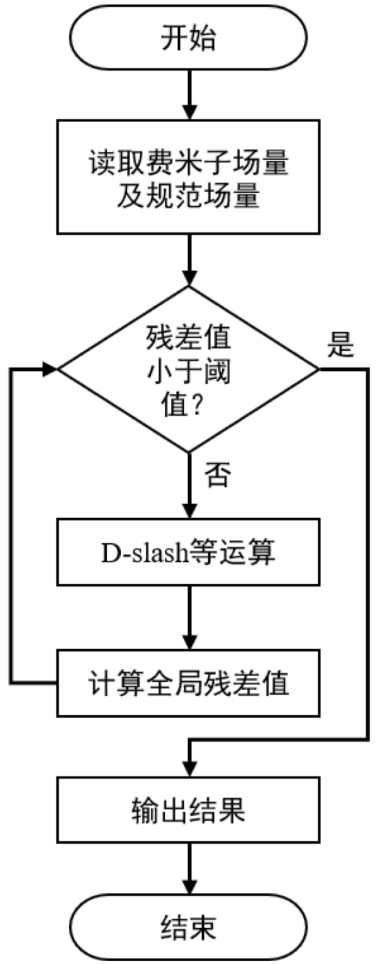
- 后摩尔时代，为了进一步提升总体性能，异构多核处理器越来越成为主流
- 国产超算选择不同的架构追求性能提升，处理器架构展现出显著的差异性

国产超算生态不完善

- 大量科学计算的软件需要移植和开发
- 现有科学计算软件普遍支持CPU/GPU，但不支持国产超算的处理器
- 移植优化科学计算软件困难且成本高

科学计算软件的特点

- 以格点QCD计算为例
- 复杂的张量计算表达式
- 循环迭代等控制流、数据搬移



$$\text{D-slash}=\sum_{\mu}\left[(1^{(4)}-\gamma_{\mu})U_{\mu}(\vec{X})\delta_{\vec{x}+\mu,\vec{y}}+(1^{(4)}+\gamma_{\mu})U_{\mu}^{\dagger}(\vec{X}-\mu)\delta_{\vec{X}-\mu,\vec{y}}\right]$$

2 目标

端到端的跨平台科学计算软件框架

- 方便使用的脚本/领域特定语言——跨平台运行
- 复杂的张量表达式
  - 自动代码生成
  - 自动优化
- 控制流、数据搬移等
  - 以算子为单位，将程序组织为算子图
  - 实现算子的自动并发/并行，提高硬件的利用率
- 使用调度器跨平台自动调度算子

3 框架设计

X语言

- 领域特定语言
- 用户友好的张量计算表达方式
- 语法树的形式表示张量计算
- 利于张量变换/并行

```
B =     [D]
|       [U] - B
|       B - [n-m]
|       [R] - Bs
Bs =    B
|       [U] - Bs
|       Bs - [n-m]
|       Bs & Bs
|       Bs - [G]
```

标识符	描述
[D]	张量
[R]	规约运算：+ * ...
[U]	一元运算：- / sin ...
[G]	下标生成器
[n-m]	导线
&	并联（对下标求交集）
-	串联

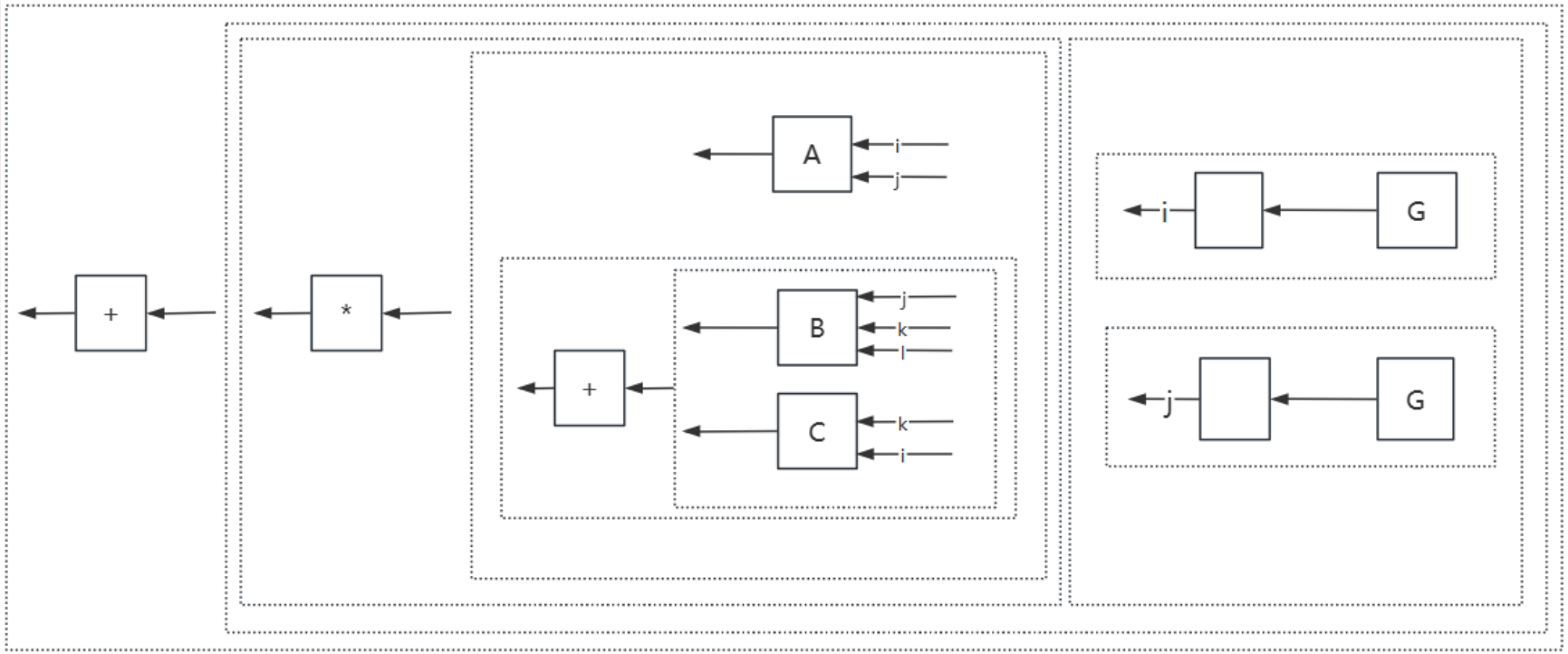
3 框架设计

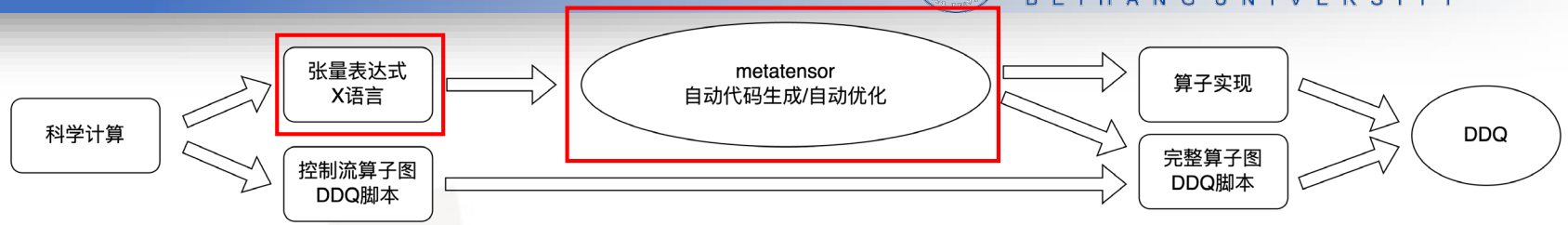
X语言

- 领域特定语言
- 用户友好的张量计算表达方式
- 语法树的形式表示张量计算
- 利于张量变换/并行

需要计算的张量表达式：

$$result_{k,l} = \sum_{i,j} A_{i,j} * (B_{j,k,l} + C_{k,i})$$

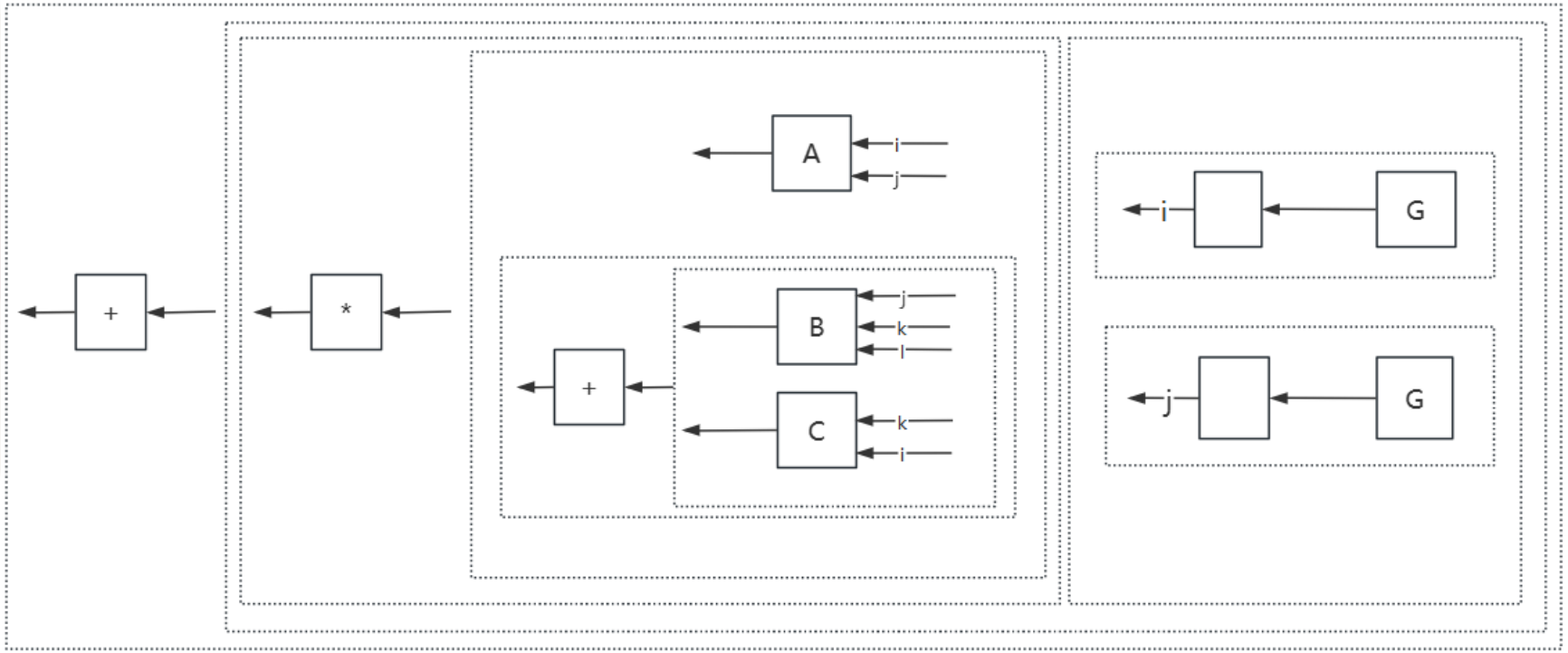




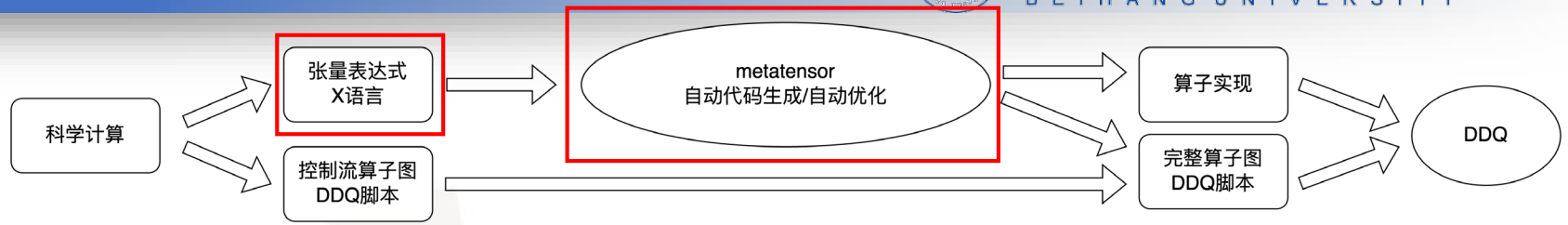
代码生成

- 转换到规范型
- 从规范型生成代码
- 原始语法树

$$res_{k,l} = \sum_{i,j} \left(A_{i,j} \times \left(B_{j,k,l} + C_{k,i} \right) \right)$$



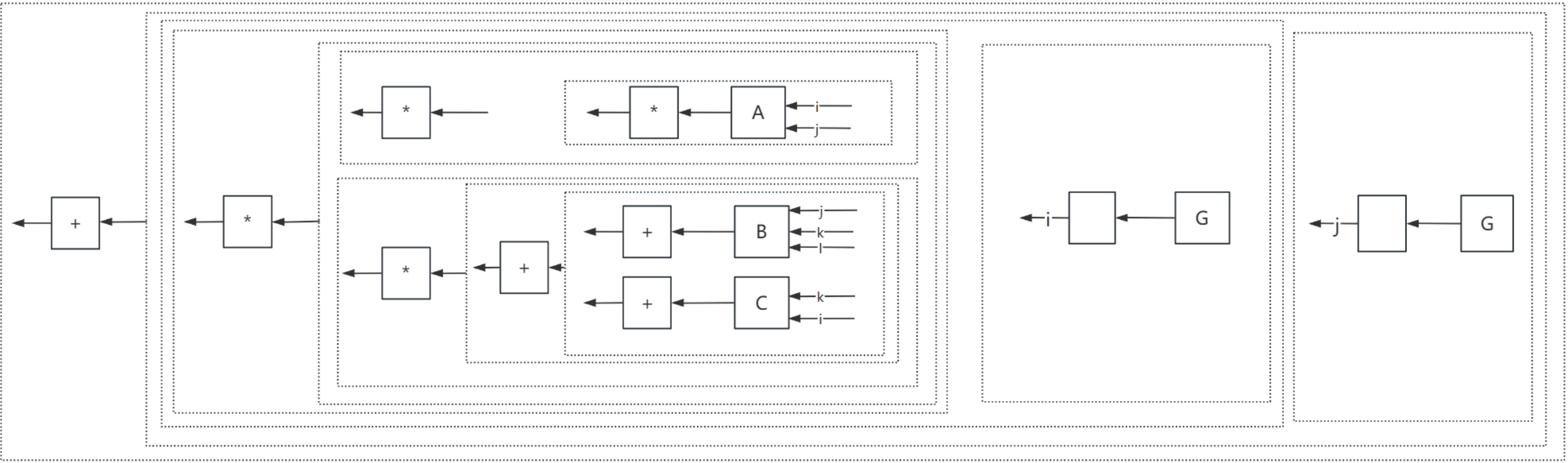
3 代码生成



代码生成

- 转换到规范型
- 从规范型生成代码
- 规范型语法树

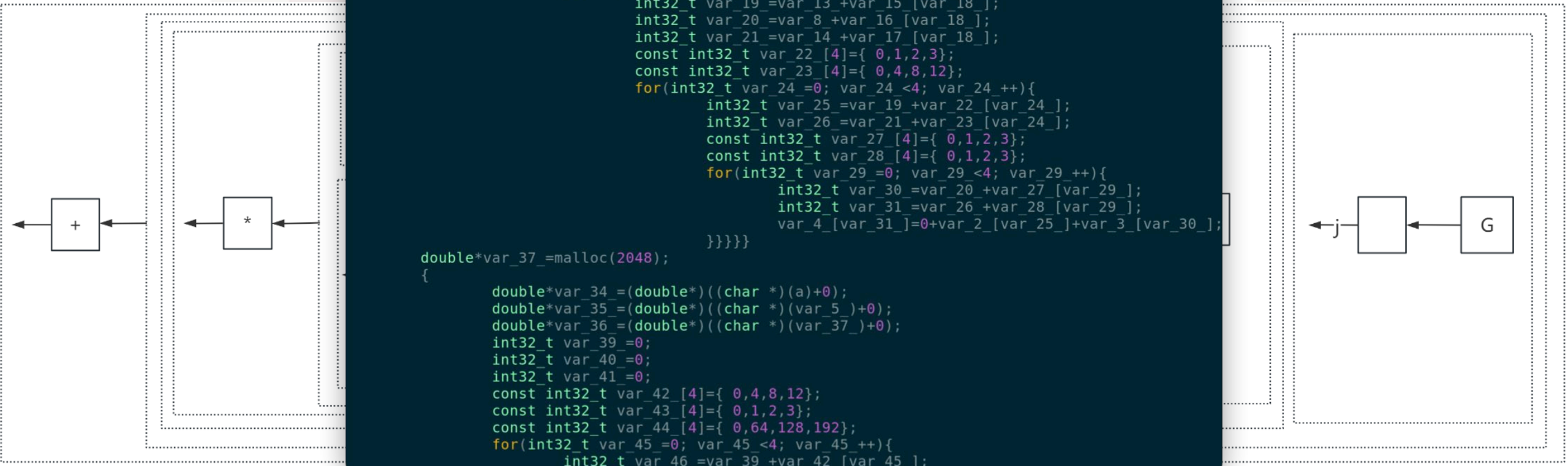
$$res_{k,l} = \sum_{i,j} \left(A_{i,j} \times \left(B_{j,k,l} + C_{k,i} \right) \right)$$



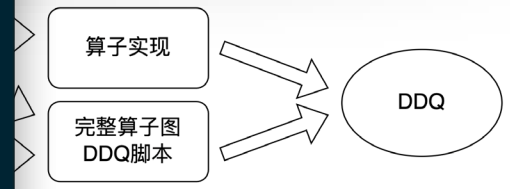
3 代码生成

代码生成

- 转换到规范型
- 规范型语法树




```
#include <stdint.h>
#include <stdlib.h>
void calc(void *res, void *a , void *b, void *c){
        double*var_5_=malloc(2048);
        {
                double*var_2_=(double*)((char *)(b)+0);
                double*var_3_=(double*)((char *)(c)+0);
                double*var_4_=(double*)((char *)(var_5_)+0);
                int32_t var_7_=0;
                int32_t var_8_=0;
                int32_t var_9_=0;
                const int32_t var_10_[4]={ 0,16,32,48};
                const int32_t var_11_[4]={ 0,64,128,192};
                for(int32_t var_12_=0; var_12_<4; var_12_++){
                        int32_t var_13_=var_7_+var_10_[var_12_];
                        int32_t var_14_=var_9_+var_11_[var_12_];
                        const int32_t var_15_[4]={ 0,4,8,12};
                        const int32_t var_16_[4]={ 0,4,8,12};
                        const int32_t var_17_[4]={ 0,16,32,48};
                        for(int32_t var_18_=0; var_18_<4; var_18_++){
                                int32_t var_19_=var_13_+var_15_[var_18_];
                                int32_t var_20_=var_8_+var_16_[var_18_];
                                int32_t var_21_=var_14_+var_17_[var_18_];
                                const int32_t var_22_[4]={ 0,1,2,3};
                                const int32_t var_23_[4]={ 0,4,8,12};
                                for(int32_t var_24_=0; var_24_<4; var_24_++){
                                        int32_t var_25_=var_19_+var_22_[var_24_];
                                        int32_t var_26_=var_21_+var_23_[var_24_];
                                        const int32_t var_27_[4]={ 0,1,2,3};
                                        const int32_t var_28_[4]={ 0,1,2,3};
                                        for(int32_t var_29_=0; var_29_<4; var_29_++){
                                                int32_t var_30_=var_20_+var_27_[var_29_];
                                                int32_t var_31_=var_26_+var_28_[var_29_];
                                                var_4_[var_31_]=0+var_2_[var_25_]+var_3_[var_30_];
                                        }}}}}
        double*var_37_=malloc(2048);
        {
                double*var_34_=(double*)((char *)(a)+0);
                double*var_35_=(double*)((char *)(var_5_)+0);
                double*var_36_=(double*)((char *)(var_37_)+0);
                int32_t var_39_=0;
                int32_t var_40_=0;
                int32_t var_41_=0;
                const int32_t var_42_[4]={ 0,4,8,12};
                const int32_t var_43_[4]={ 0,1,2,3};
                const int32_t var_44_[4]={ 0,64,128,192};
                for(int32_t var_45_=0; var_45_<4; var_45_++){
                        int32_t var_46_=var_39_+var_42_[var_45_];
                        int32_t var_47_=var_40_+var_43_[var_45_];
```



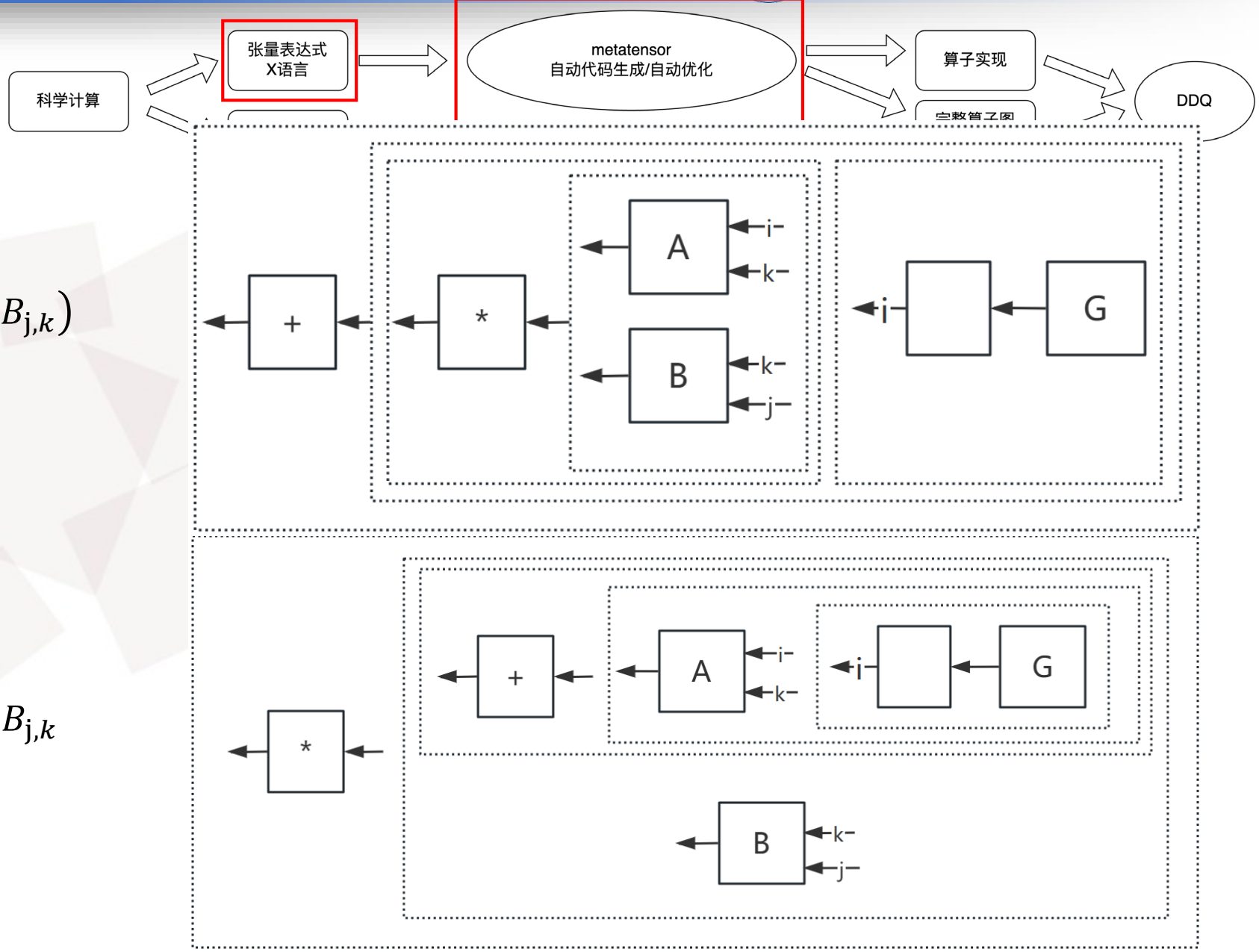
代码自动优化

$$res_{k,\mathrm{j}} = \sum_i \left(A_{i,k} \times B_{\mathrm{j},k}\right)$$

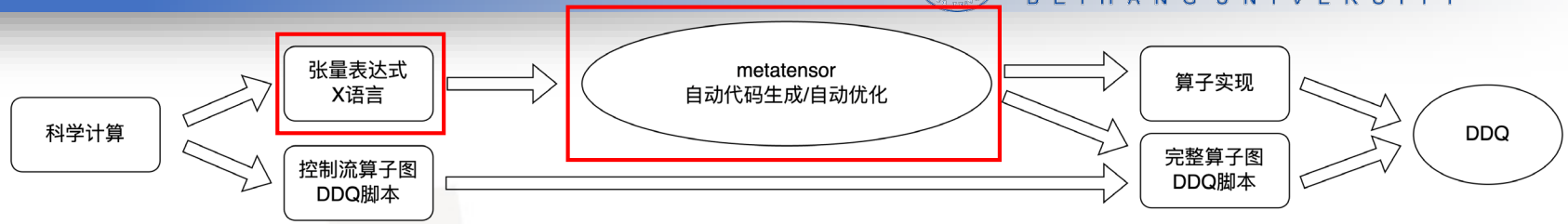
原始语法树

$$res_{k,\mathrm{j}} = \sum_i \left(A_{i,k}\right) \times B_{\mathrm{j},k}$$

优化语法树

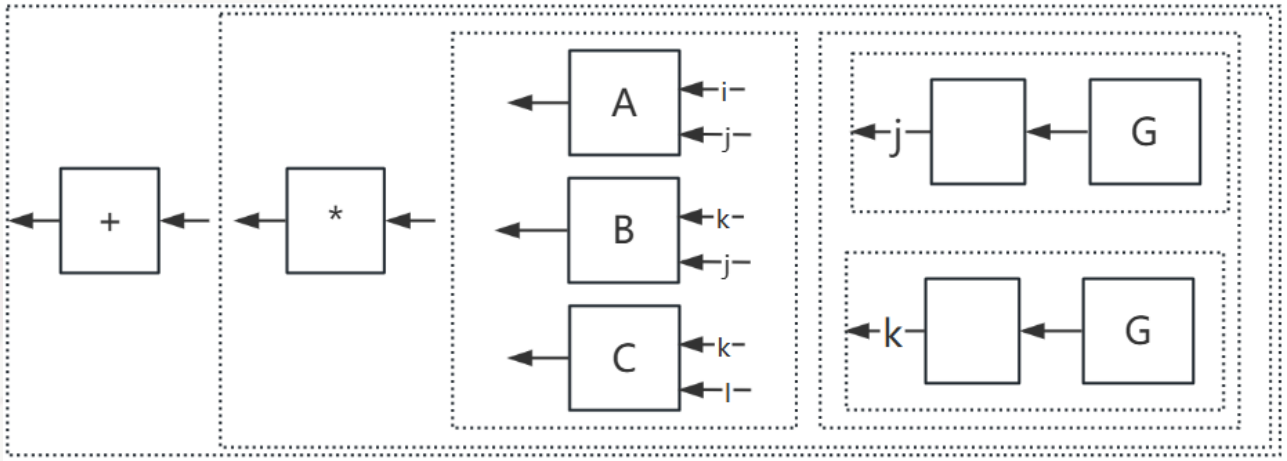


代码自动优化



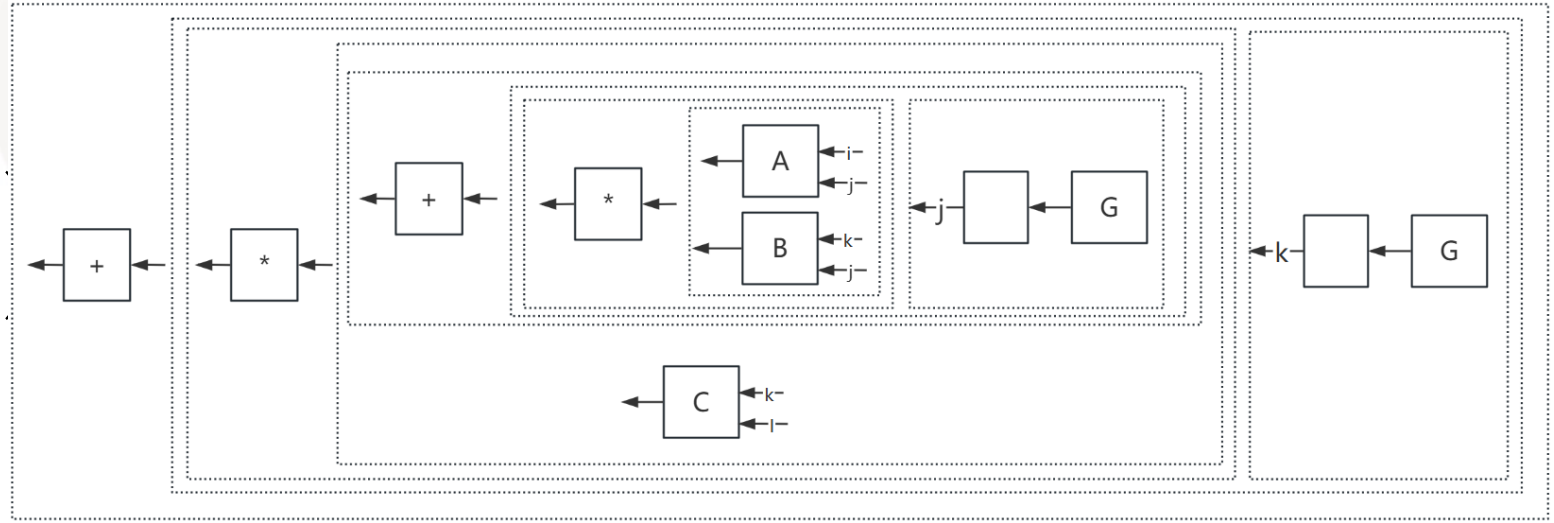
$$res_{i,l} = \sum_{j,k} \left(A_{i,\mathrm{j}} \times B_{\mathrm{j},k} \times C_{\mathrm{k},l} \right)$$

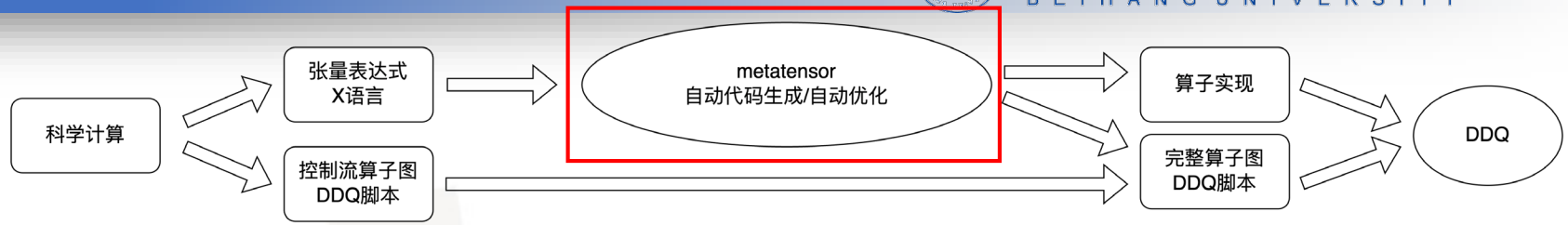
原始语法树



$$res_{k,\mathrm{j}} = \sum_{k} \left(\sum_{j} \left(A_{i,\mathrm{j}} \times B_{\mathrm{j},k} \right) \times C_{\mathrm{k},l} \right)$$

优化语法树





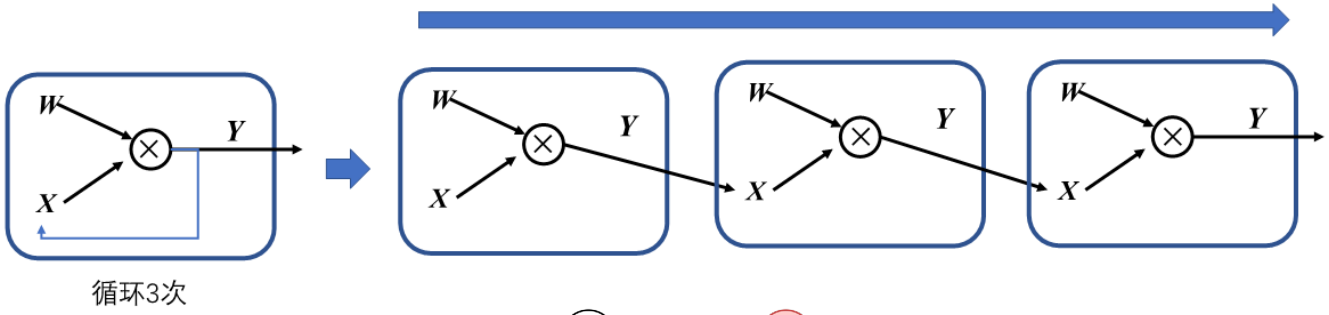
张量描述：datadesc

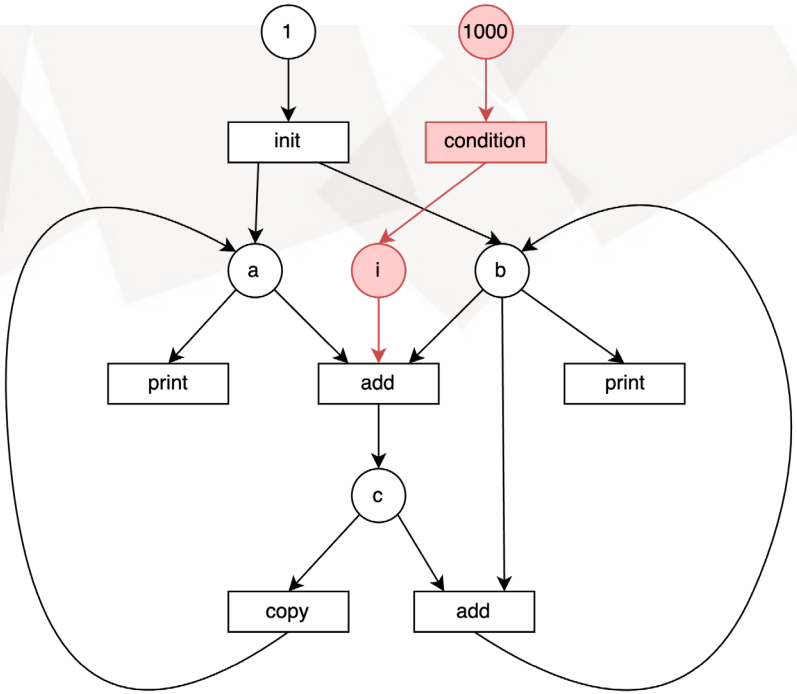
- 维度描述：dimdesc

维度属性	描述
int_t n_tuple	dimdesc 描述的真实维度数量
int_t n_entry	张量在该维度下的尺寸
int_t *indices	维度下所有合法的下标取值（或下标组合取值）
int_t *offsets	维度下所有合法的下标取值对应的偏移量
int_t dims[]	dimdesc 描述的各真实维度在全局的编号

算子图

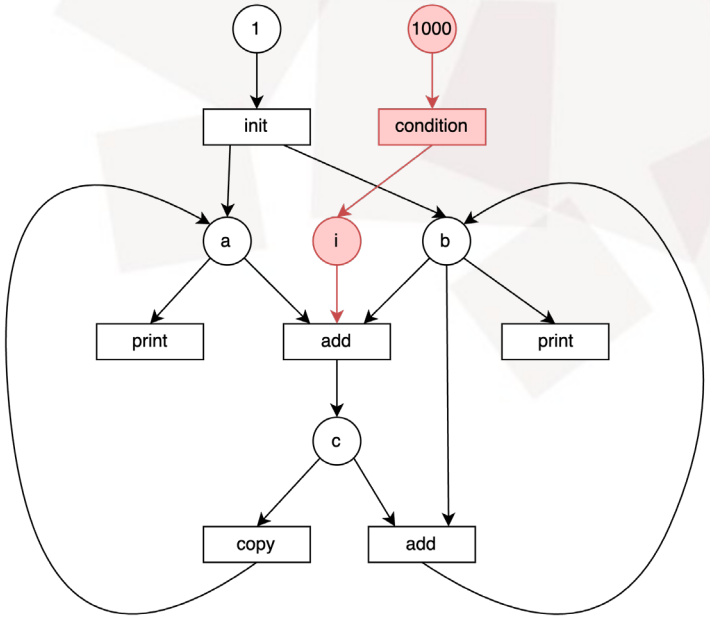
- 包含环状结构
  - 有环图vs无环图
    - 科学计算循环次数不确定，根据条件确定
    - 循环展开导致节点过多
  - 算子与数据共同为节点
    - 数据具有状态
    - 内存层次复杂，数据搬移对性能影响大





DDQ脚本(算子图描述脚本)

- 读写友好且计算机易于分析
- 算子库的开发较为灵活
- 指令顺序无关




```
load_op(std/int/print) < [a] < load_op(std/int/copy) < [c] < load_op(std/int/add) < [a b]
load_op(std/int/print) < [b] < load_op(std/int/add) < [b c]
[a] < load_op(std/int/init) < [1]
[b] < load_op(std/int/init) < [1]
```

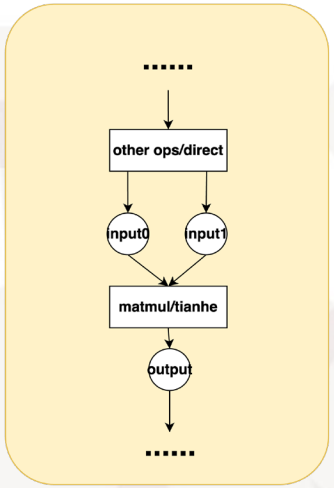
举例：计算圆周率 π

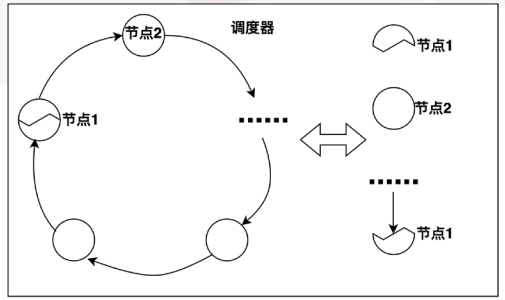
$$\frac{1}{4}(\pi-3)=\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{2k(2k+1)(2k+2)}=\frac{1}{2\cdot3\cdot4}-\frac{1}{4\cdot5\cdot6}+\frac{1}{6\cdot7\cdot8}-\cdots$$

```
load_op(std/real/print) < [pi] < load_op(std/real/add) < [pi term]

[term] < load_op(std/real/mul4) < [s ia ib ic]

[s] < load_op(std/real/mul) < [s -1.0]

[ia] < load_op(std/real/div) < [1.0 a]
[ib] < load_op(std/real/div) < [1.0 b]
[ic] < load_op(std/real/div) < [1.0 c]

[a] < load_op(std/real/add) < [a 2.0]
[b] < load_op(std/real/add) < [b 2.0]
[c] < load_op(std/real/add) < [c 2.0]

[pi] < load_op(std/real/init) < [3.0]
[s] < load_op(std/real/init) < [4.0]
[a] < load_op(std/real/init) < [2.0]
[b] < load_op(std/real/init) < [3.0]
[c] < load_op(std/real/init) < [4.0]

load_op(std/time/sleep) < [1]
```

DDQ(跨平台调度器)

- 调度器根据算子图进行自动调度
- 算子之间可以实现并发/并行执行
- 调度器主体代码跨硬件可复用
- 实现对算子的跨平台调度





后端		
mpi	direct	fork
cuda	tianhe	pthread

DDQ(跨平台调度器)

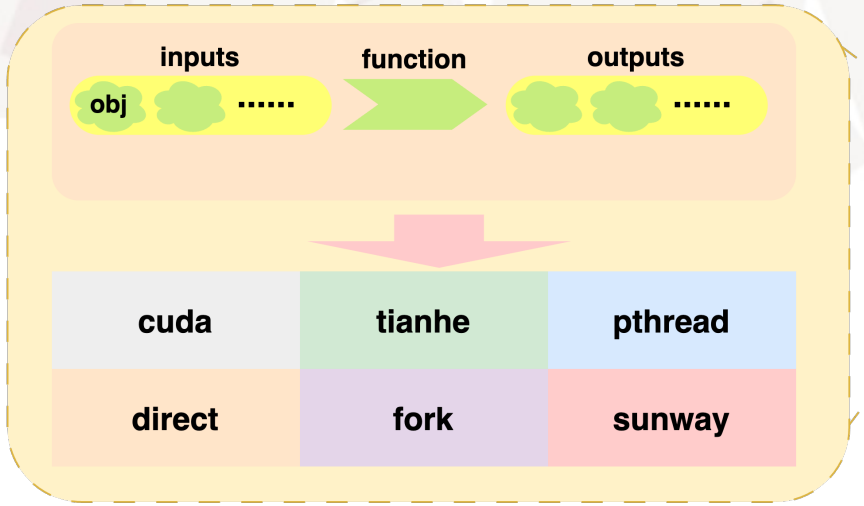
- 统一算子接口
- 任意数量和类型的输入和输出
- 灵活支持多种操作

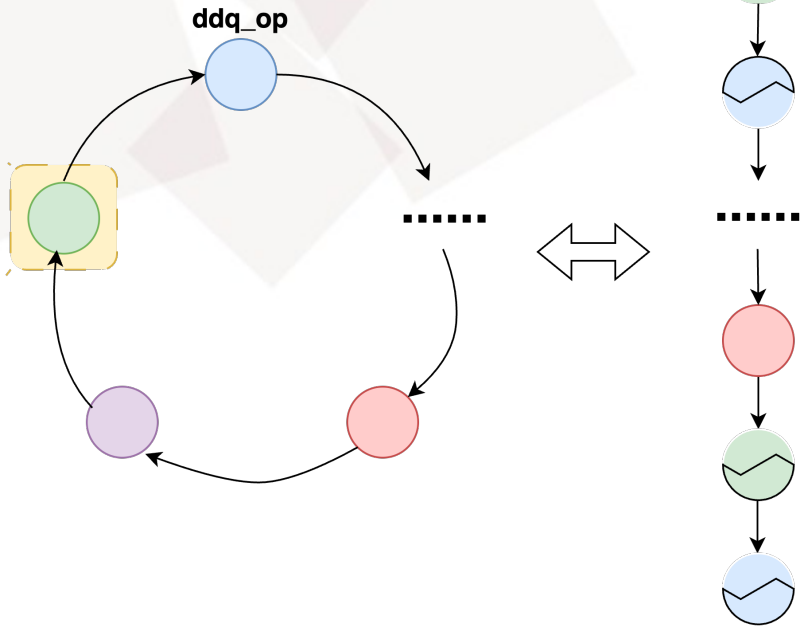
```
typedef enum
{
task_ret_ok = 0, // 任务完成，算子保留下次使用

task_ret_done, // 任务完成，算子不必保留

task_ret_again, // 任务未完成，可以暂时移交控制权以等待并发工作

task_ret_error, // TODO: 错误有哪些？怎么处理？

task_ret_kernel, //需要调用kernel函数

// ...

task_ret_last
} task_ret;

typedef task_ret task_f(void **inputs, void **outputs);
```

DDQ(跨平台调度器)

- DDQ的主体包括：算子，对象（具有状态），节点（由算子和对象共同组成）
- 程序循环遍历节点组成的环，某个节点满足执行条件自动调度执行
- 每一个节点即一个协程（协作式线程），节点的算子为执行完返回task_ret_again，将计算资源的控制权移交给其他节点实现算子并发

ddq_op_t
obj f obj * inputs obj * outputs ddq_op next ddq_label pos





DDQ(跨平台调度器)

- 基于协作式线程（协程）设计
- 一个节点对应一个协程
- DDQ提供一些协程相关原语
- 通过不同运行环境的后端代码，实现算子跨平台调度

ddq_op_t

```
obj f
obj * inputs
obj * outputs
ddq_op next
ddq_label pos
```

```
ddq_start();

ddq_do();

o_v(is_done) = 0;

if (pthread_create(&o_v(pt), NULL, run_pthread, ddq_p))
error("processor_pthread : pthread_create() fails.\n");

ddq_waitfor(o_v(is_done));

if (pthread_join(o_v(pt), NULL))
warning("processor_pthread : pthread_join() fails.\n");

ddq_while( o_v(ret) == task_ret_ok );

ddq_finish();
```

```
void * run_pthread(void *args)
{
struct processor_pthread_t *p = args;

p->ret = ((task_f *)(p->head.f->p)) (p->head.p_inputs, p->head.p_outputs);
p->is_done = 1;

return NULL;
}
```

4 总结

端到端的跨平台科学计算软件框架

- 方便使用的脚本/领域特定语言——跨平台运行
- 复杂的张量表达式
  - 使用领域特定语言表示张量表达式
  - 自动代码生成/自动优化
- 控制流、数据搬移等
- 使用调度器跨平台自动调度算子

研发贡献者（统计至：2024/08/17）

- 中国科学院高能物理研究所

  宫明、陈莹、刘朝峰、毕玉江、孙玮、施春江、蒋翔宇

- 北京航空航天大学

  栾钟治、韩斌、王御臣、肖敏毅、房歆哲、龚煜涵、马世清、刘星宇、李根、王逸杰、李亦白

- 中国科学院计算机网络信息中心

  徐顺、张克龙、韩秉豫、张术飞

- 中国科学院理论物理研究所

  王建成

欢迎各位感兴趣的老师同学一起参与贡献！

THANKS

姓名：韩斌 龚煜涵

2024年8月19日