格点 QCD 计算新软件框架

中科院高能所
宫明

2024.05.21

研发贡献者

- 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、刘星宇、李根、王逸杰

- 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

格点 QCD 的基本公式

$$\text{人} + \text{软件} + \text{计算资源} = \text{科研成果}$$

+

+

国内外格点 QCD 计算软件现状

USQCD SciDAC 软件集（Chroma、QUDA、QDP++、……）

+ 国际普遍使用
+ 模块化，支持 intel、nvidia 等架构

- 原版不支持国产超算
- 单任务，中小规模运行

Bridge++、Grid、tmLQCD、openQCD、……

+ 代码量较小，易于扩展和修改
+ 适应各自特定需求

- 功能单薄
- 跨平台性较差
- 不支持国产超算

各研究组的其他软件等

+ 实现通用软件不支持的功能

- 规模小、功能单一、大多不跨平台
- 大多不开源

国产新一代软硬件环境的挑战

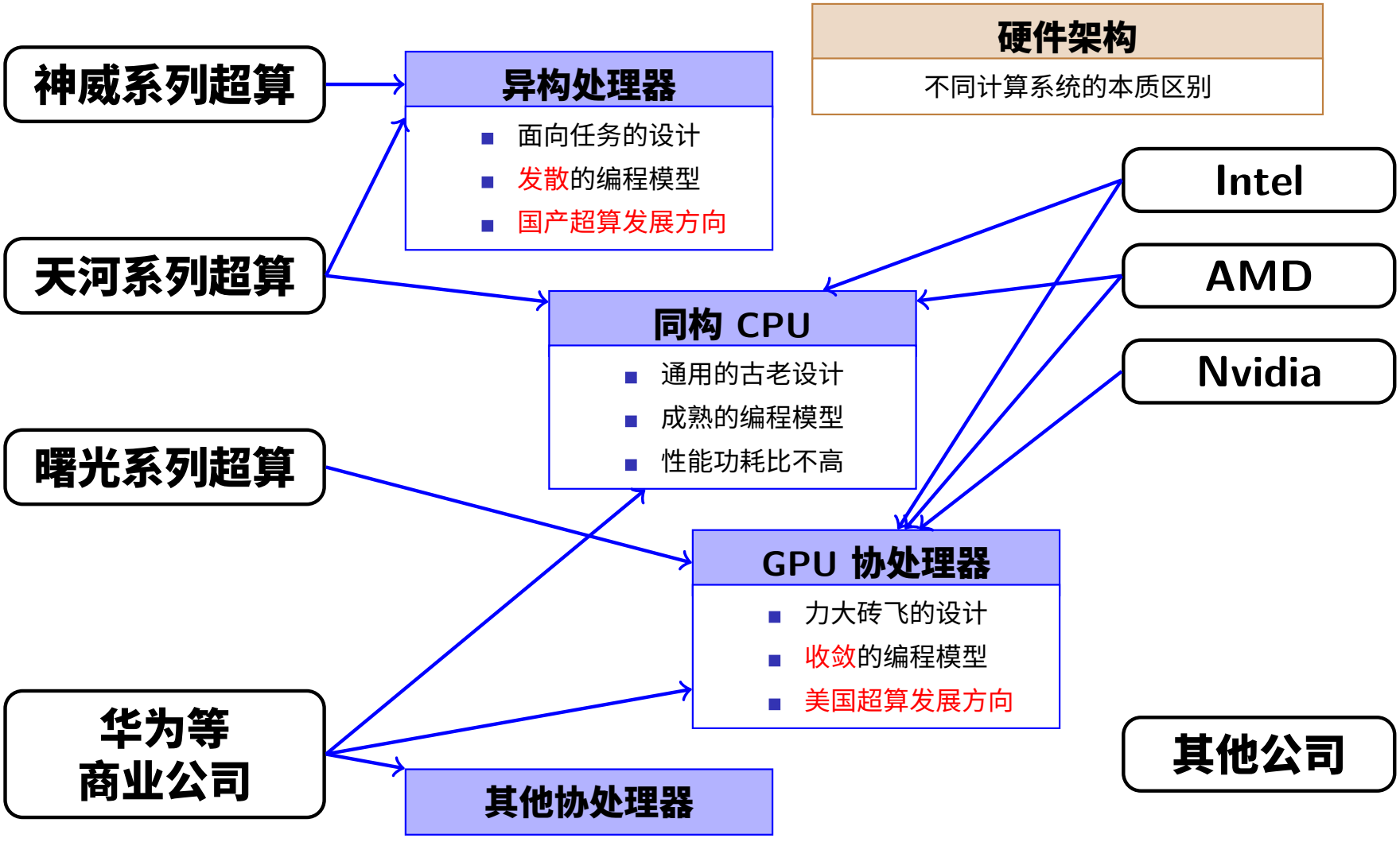
硬件的多样性

- 神威、天河、曙光三个架构都非常不同
- 同系列下一代超算架构变化也非常大
- 未来超算仍处于快速演化过程中
- \+ 所以软件设计必须挖掘共性特点、面向未来趋势。

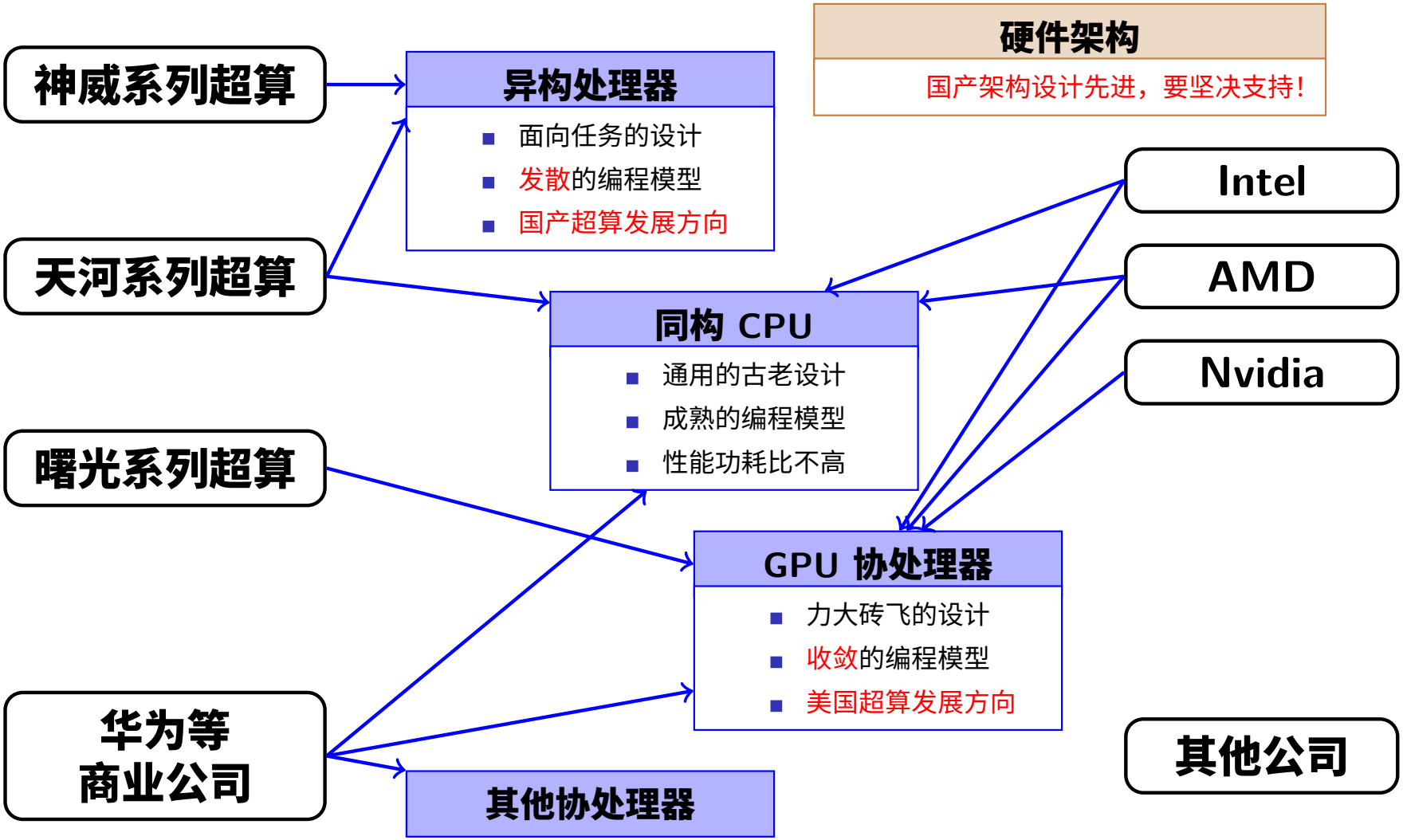
基础软件刚刚起步

- 新天河系统、神威系统仍不完善，急需大量基础软件支持。
- 曙光系统有开源社区支持，但也缺少很多基础软件。
- \+ 所以软件研发需要从基础软件做起，但同时也抛弃了历史包袱：一张白纸绘新图。

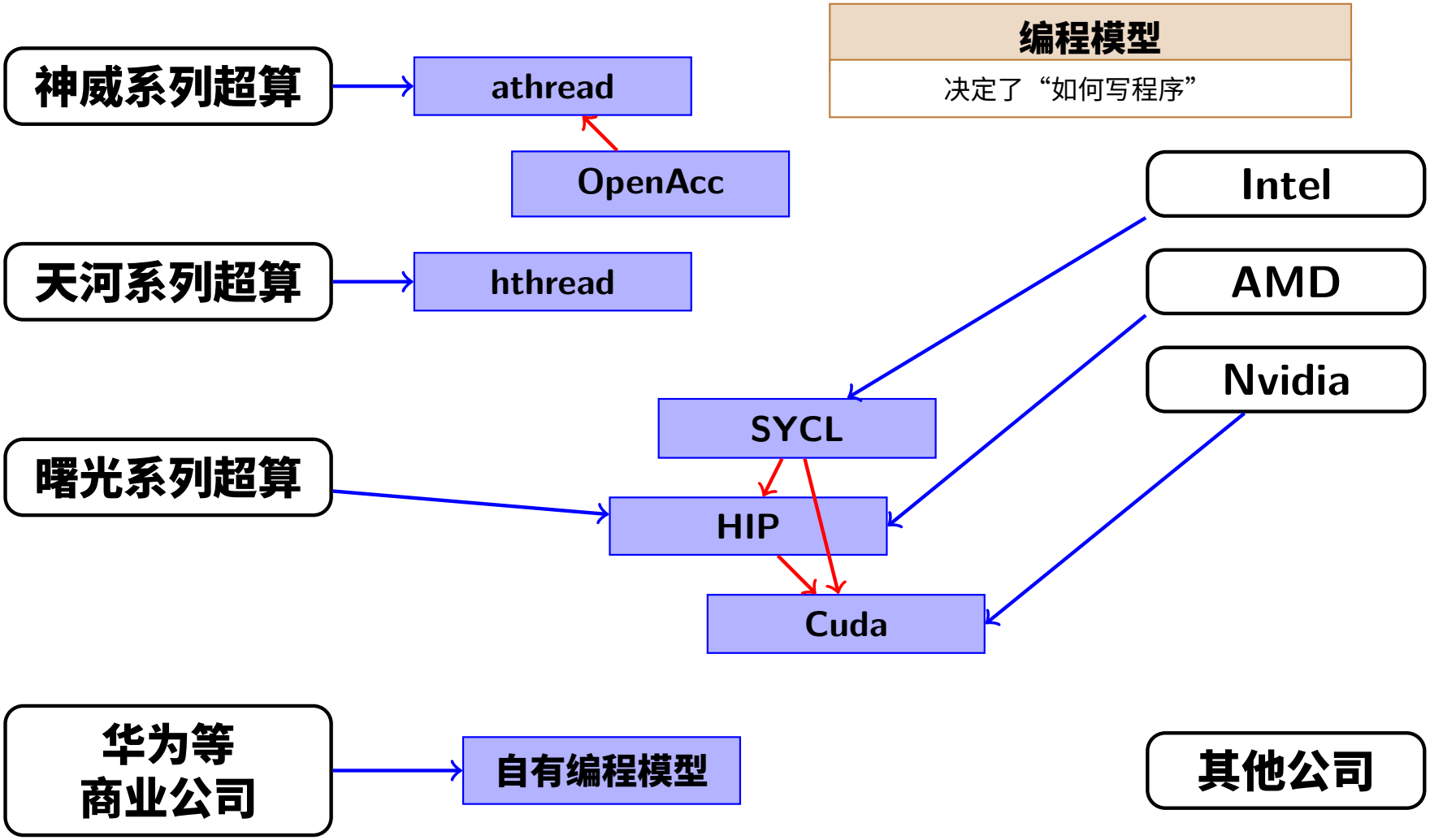
超级计算机的主要处理器平台



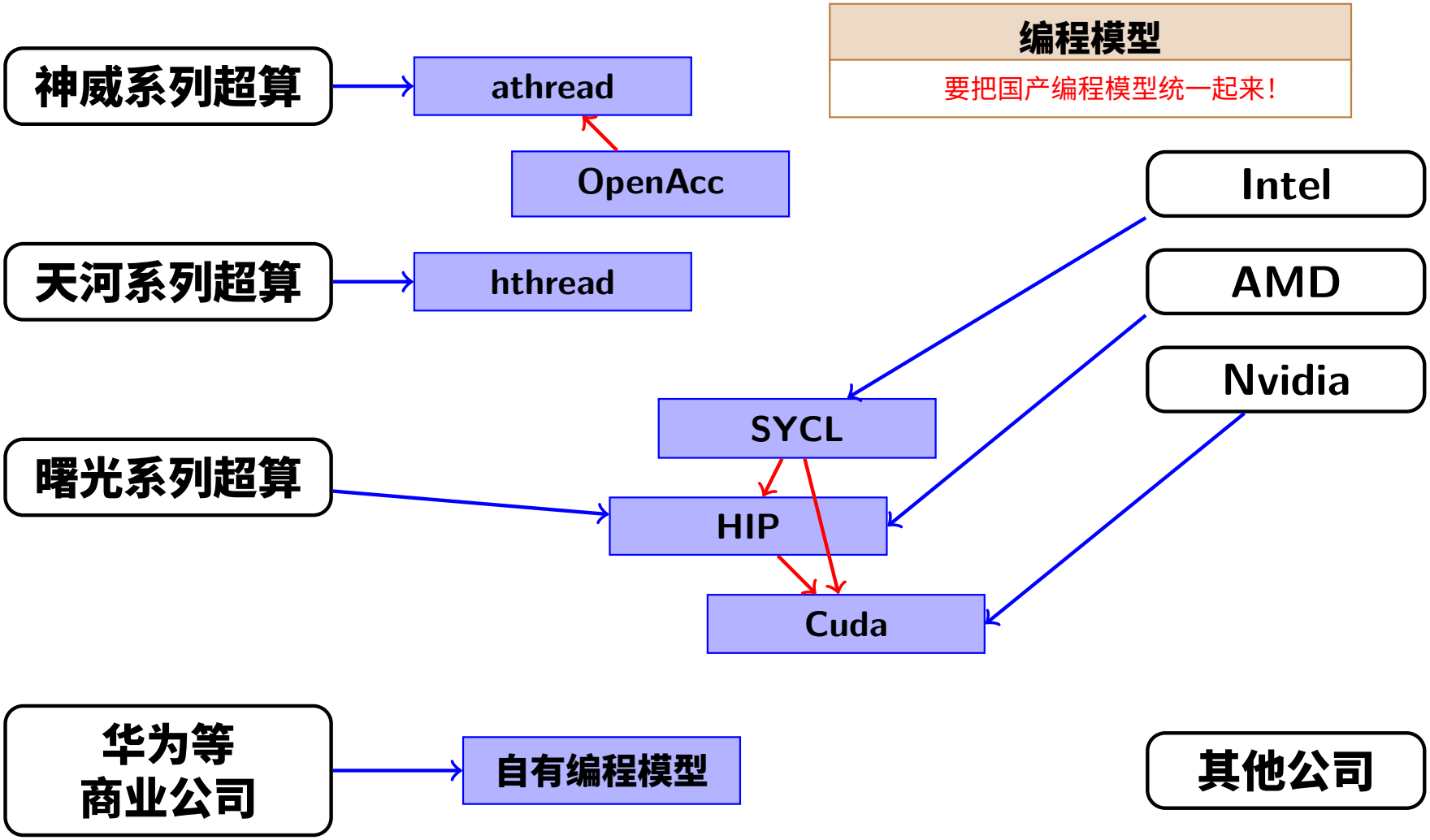
超级计算机的主要处理器平台



超级计算机的主要处理器平台



超级计算机的主要处理器平台



并行编程模型回顾

常用并行编程模型

- 格点 QCD 定制并行模型：QDP、Grid
- 进程并行：MPI
- 核心函数线程并行：CUDA、HIP、athread
- 匿名函数线程并行：SYCL、KOKKOS、RAJA
- 代码段线程并行：OpenMP、OpenAcc
- 指令并行：SIMD

- 以上编程模型在结构上分为两大类：对称、主从。
- 格点 QCD 软件通常混合采用对称和主从模型实现二级并行，代码结构很复杂。
- 以上模型中，除了 KOKKOS 外，大多数模型没有系统地处理多层次异构内存。
- 需要根据彻底的异构前提，设计新的编程模型。

未来科学计算技术的四个发展方向

自动代码生成

- 这是当前热点：元编程技术
- 路径：公式 $\rightarrow$ 中间表示 $\rightarrow$ 代码
- 是其他三个发展方向的基础设施
  - 参数化生成、面向平台生成、低代码界面

跨平台部署和调度

- 大数据框架是关键基础设施
  - Spark、Map-Reduce、Pregel、……
- 传统科学计算软件在技术上已落后
- 急需“高性能 + 可移植”的两全设计

自动代码优化

- 人工智能相关技术正在跃进
  - 科学计算软件要准备好借东风
- 优化理论和智能算法有发展潜力
- 需要完整的支撑平台来测试和部署

适合科研用户的人机界面

- “物理的归物理，计算的归计算”
  - 这是目前科学计算的最大痛点
- 面向物理公式的编程模型创新
- 面向科研工作的敏捷开发平台

格点 QCD 在四个方向上的需求和解决方案

MetaTensor：自动代码生成

- 物理计算基于复杂的公式
- 所有公式都可以转写为广义的“张量表达式”

张量表达式计算

DDQ：高效跨平台调度器

- 格点 QCD 的算法结构主要是循环迭代
- 需要大规模并行、需要高效数据交换

循环迭代

拆解算法流程

格点 QCD 计算软件

拆解计算程序

纯计算部分

- 与物理结果无关，即为优化空间
- 参数化此空间，开展自动优化研究

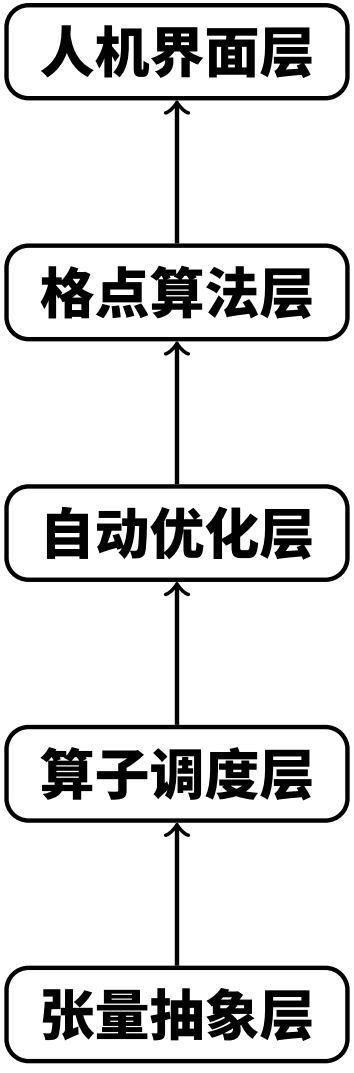
Trene：通用自动优化示例

纯物理部分

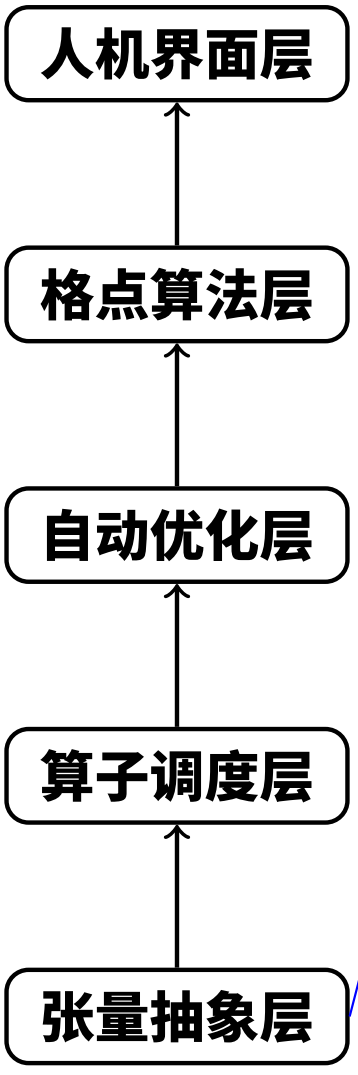
- 隔离计算细节，面向物理公式
- 适合做科研用户界面的编程模型

x 语言：图形化编程语言

格点 QCD 新软件框架



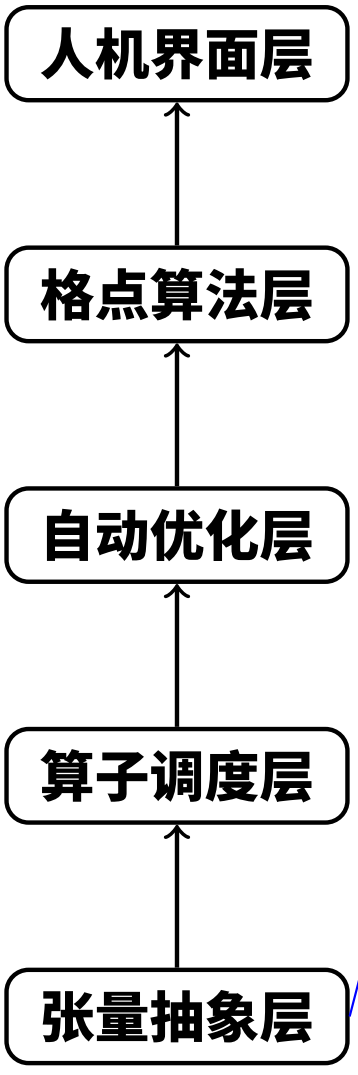
格点 QCD 新软件框架



编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

格点 QCD 新软件框架



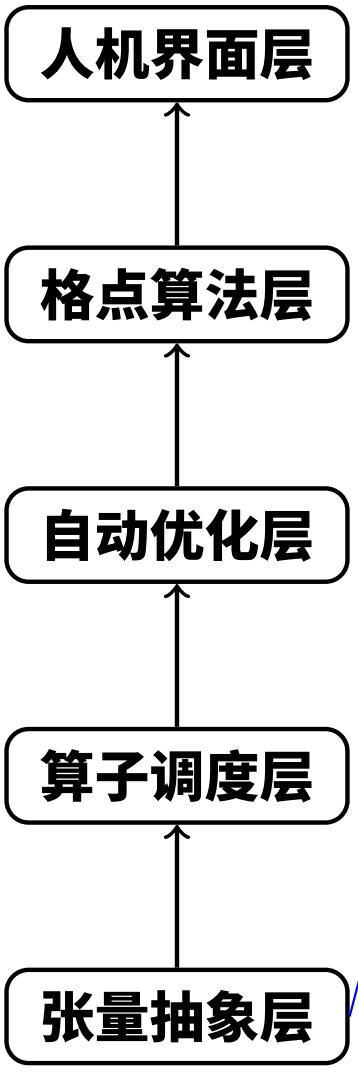
编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

面向张量的代码生成器：MetaTensor

- 对所有的数据有统一的表示：数据描述 (datadesc)
- 对程序语法树有统一的表示：x 语言块 (block)
- 自身功能组织为统一的插口：插件系统 (plugin)
- 对软硬件环境有统一的管理：编译目标 (target)

格点 QCD 新软件框架



编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

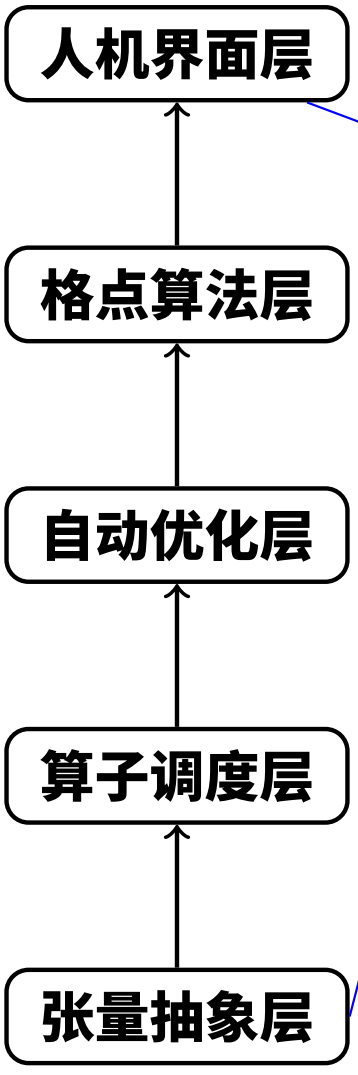
面向张量的代码生成器：MetaTensor

- 对所有的数据有统一的表示：数据描述 (datadesc)
- 对程序语法树有统一的表示：x 语言块 (block)
- 自身功能组织为统一的插口：插件系统 (plugin)
- 对软硬件环境有统一的管理：编译目标 (target)

MetaTensor 的数据描述

- 数据有“维度”和“指标”的结构，像多维数组
- 数据可以是稀疏的
- 编译期和运行时的数据是统一描述的

格点 QCD 新软件框架



编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

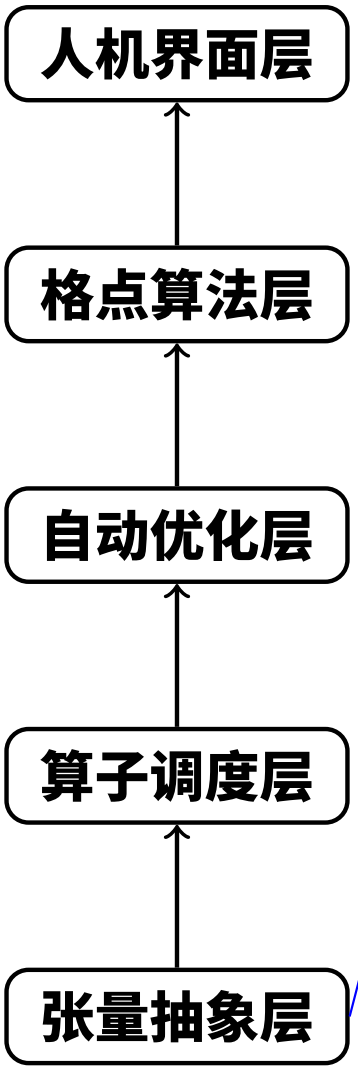
面向张量的代码生成器：MetaTensor

- 对所有的数据有统一的表示：数据描述 (datadesc)
- 对程序语法树有统一的表示：x 语言块 (block)
- 自身功能组织为统一的插口：插件系统 (plugin)
- 对软硬件环境有统一的管理：编译目标 (target)

MetaTensor 的中间表示

- MetaTensor 的中间表示 (IR) 是 x 语言的子集
- x 语言是自主设计的图形化的编程语言
- MetaTensor 用 x 语言的“块”结构来描述张量表达式

格点 QCD 新软件框架



编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

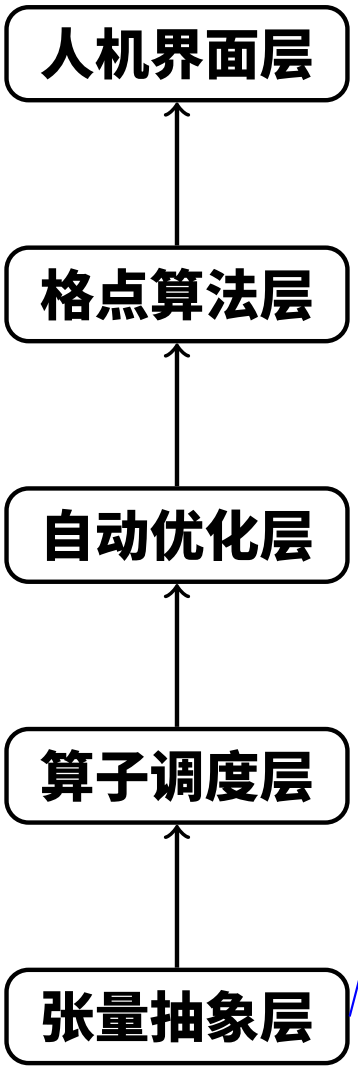
面向张量的代码生成器：MetaTensor

- 对所有的数据有统一的表示：数据描述 (datadesc)
- 对程序语法树有统一的表示：x 语言块 (block)
- 自身功能组织为统一的插口：插件系统 (plugin)
- 对软硬件环境有统一的管理：编译目标 (target)

MetaTensor 的插件系统

- 包含且仅包含四个定义良好的插口：
  - 复制 (datacopy) 插口
  - 求值 (block2data) 插口
  - 优化 (optimize) 插口
  - 编译 (compile) 插口
- 强大而灵活，足够描述所有需要的功能！

格点 QCD 新软件框架



编译期的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 表达式的分析越早越好：元编程

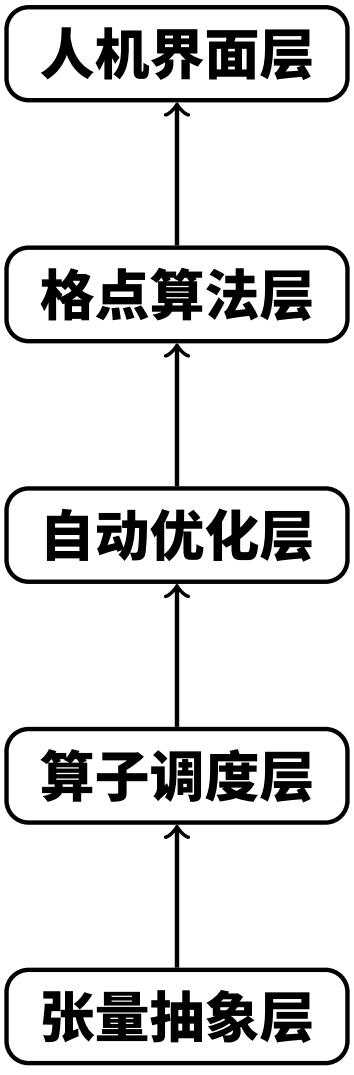
面向张量的代码生成器：MetaTensor

- 对所有的数据有统一的表示：数据描述 (datadesc)
- 对程序语法树有统一的表示：x 语言块 (block)
- 自身功能组织为统一的插口：插件系统 (plugin)
- 对软硬件环境有统一的管理：编译目标 (target)

MetaTensor 的编译目标

- 编译目标是对插件集合的选取方案
- 每个插口上安排一条插件链
- 为不同的软硬件环境设置不同的编译目标
- 目前的代码编译到 C 语言，保留其他后端的兼容性
- 插件和编译目标的维护工作是脱耦的

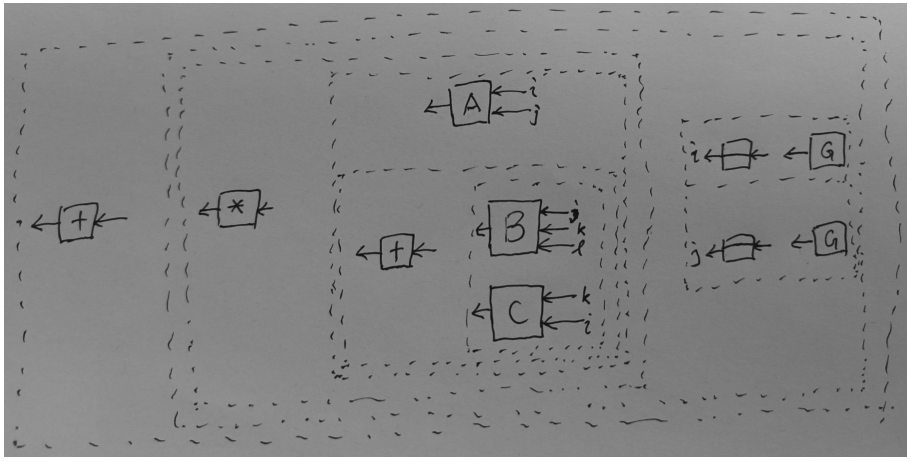
格点 QCD 新软件框架



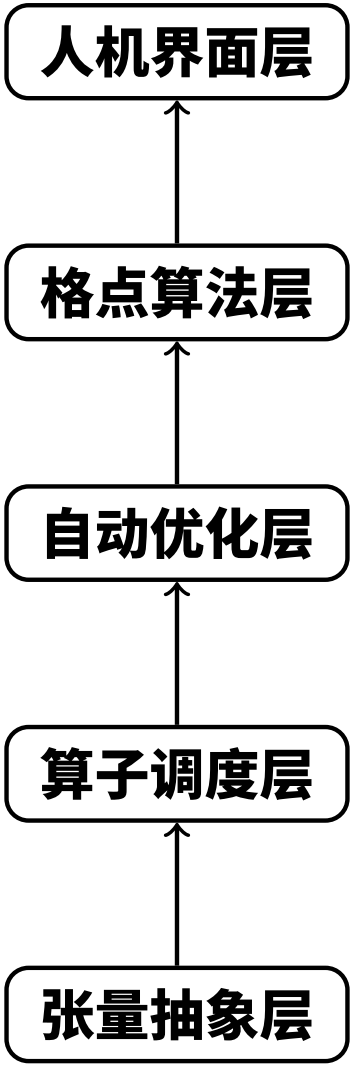
需要计算的张量表达式：

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

用 x 语言转写为：



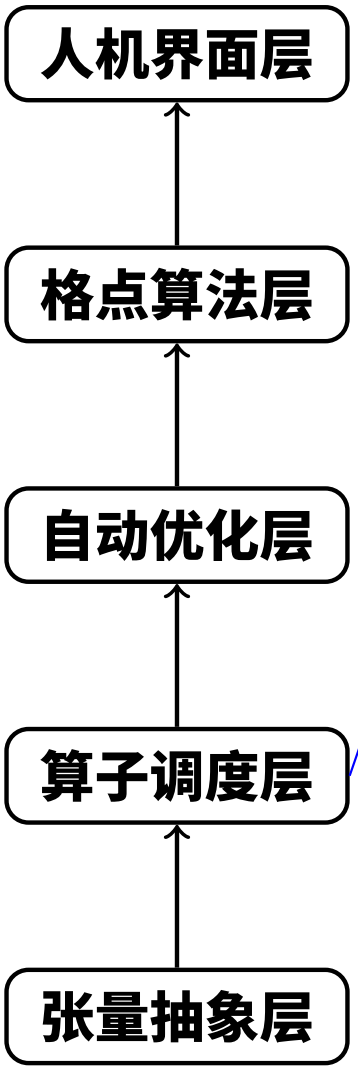
格点 QCD 新软件框架



MetaTensor 把它编译到 C 语言：

```c
#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_];
```

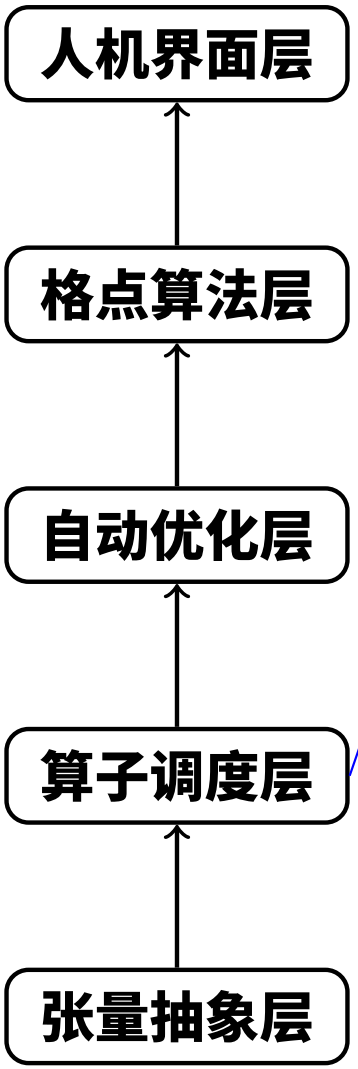
格点 QCD 新软件框架



运行时的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 格点 QCD 计算特点：大规模并行、数据带宽要求高
- 国产超算架构多样，生态环境建设落后

格点 QCD 新软件框架



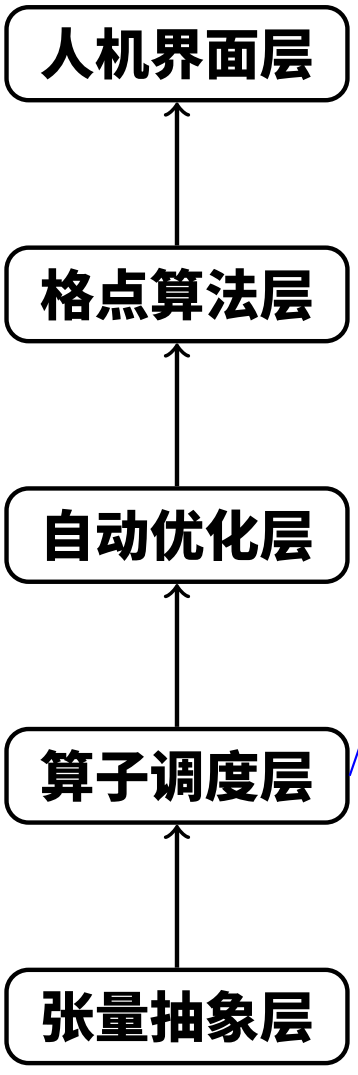
运行时的设计思考

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
- 格点 QCD 计算特点：大规模并行、数据带宽要求高
- 国产超算架构多样，生态环境建设落后

跨平台运行时调度器：DDQ

- 跨平台！跨平台！跨平台！
  - 用协程模型写任意软硬件平台的驱动程序
- 通用的算子模型和内存对象模型，包含自动内存管理
- 工作流用有环图描述，可以运行时动态生成，以此实现复杂的动态控制流
- 协程并发机制，自动实现双缓冲、后台 IO 等优化技术
- 计算任务与调度逻辑完全脱耦，非常适合作为通用的计算服务器

格点 QCD 新软件框架



运行时的设计思考

- ……

跨平台运行时调度器：DDQ

- ……驱动程序……
- ……算子……

驱动程序

- DDQ 通过驱动程序在软硬件平台上运行算子
- 只需要几百行规模的代码即可实现一个驱动程序
- 目前实现的驱动程序：
  - 任意处理器的“本地”运行
  - x86/arm 的 fork 进程
  - x86/arm 的 pthread 线程
  - 天河三 DSP 设备上的线程
  - CUDA 设备 (初步)
- 理论上容易实现的平台：
  - CUDA/HIP/SYCL 等常用异构平台
  - 申威主/从核上的线程
  - 通过网络的 RPC 调用

格点 QCD 新软件框架

运行时的设计思考

- ……

跨平台运行时调度器：DDQ

- ……驱动程序……
- ……算子……

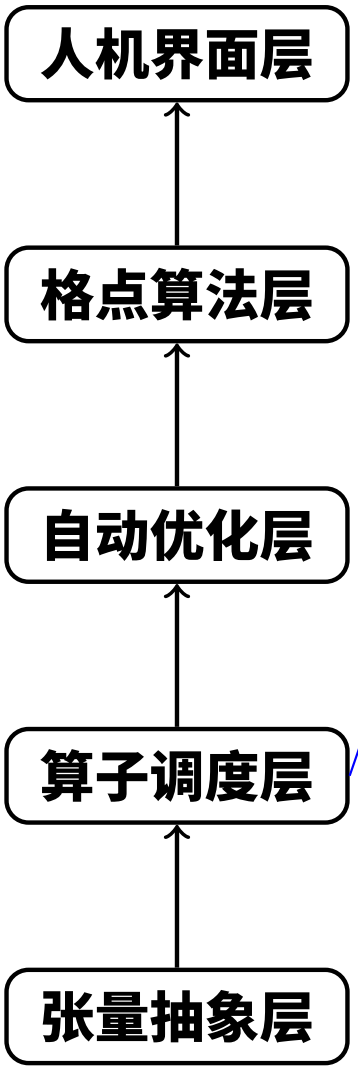
算子

- DDQ 的算子声明：

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

  - 通用性非常强，可以描述几乎所有功能
- 目前实现的算子：
  - 文件 IO 算子、命令行调用算子
  - 主内存和 CUDA 显存的 memcpy 算子
  - zmq 网络接口算子、Redis 数据库访问算子
  - 天河三 DSP 设备上的 Gromacs 热点函数算子
  - 天河三 DSP 设备上的软 cache 算子
  - 格点 QCD 的 Chroma 包装算子
  - 动态更新工作流的 bridge 算子
  - 并发分派的 iterator 算子
  - 脚本编译算子

格点 QCD 新软件框架



运行时的设计思考

- ……

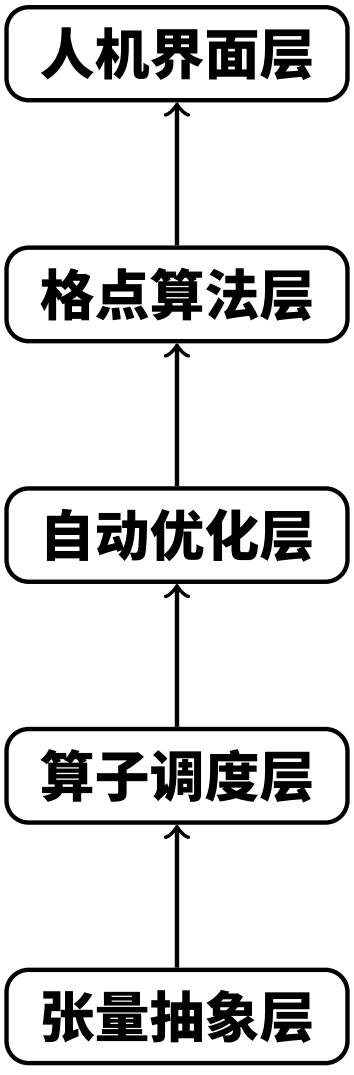
跨平台运行时调度器：DDQ

- ……驱动程序……
- ……算子……

脚本与复合算子

- DDQ 提供一个定制的脚本环境
- 脚本负责描述算子构成的有向图，实现程序控制流
- 可以用脚本把多个算子组合成为复合算子，从而层迭构建各种粒度的模块
- 脚本的语法是图灵完全的
- 所有不同功能的基本算子和复合算子，整理为算子库（oplib），以树形结构规划，适合分布式开发和维护

格点 QCD 新软件框架



举例：计算圆周率 π

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

用 DDQ 脚本转写为：

add.ddq:

```
newop.f = load_so(arithmetic/add)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double) load_type(double)]
```

mul.ddq:

```
newop.f = load_so(arithmetic/mul)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double) load_type(double)]
```

inv.ddq:

```
newop.f = load_so(arithmetic/div)
newop.processor = "direct"
[load_type(double)] < newop < [1.0 load_type(double)]
```

mul4.ddq:

```
[s] < newop < [a b c d]
[s] < load_op(mul) < [abc d]
[abc] < load_op(mul) < [ab c]
[ab] < load_op(mul) < [a b]
```

init.ddq:

```
newop.f = load_so(arithmetic/init)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double)]
```

print.ddq:

```
newop.f = load_so(arithmetic/print)
newop.processor = "direct"
[] < newop < [load_type(double)]
```

calc_pi.ddq:

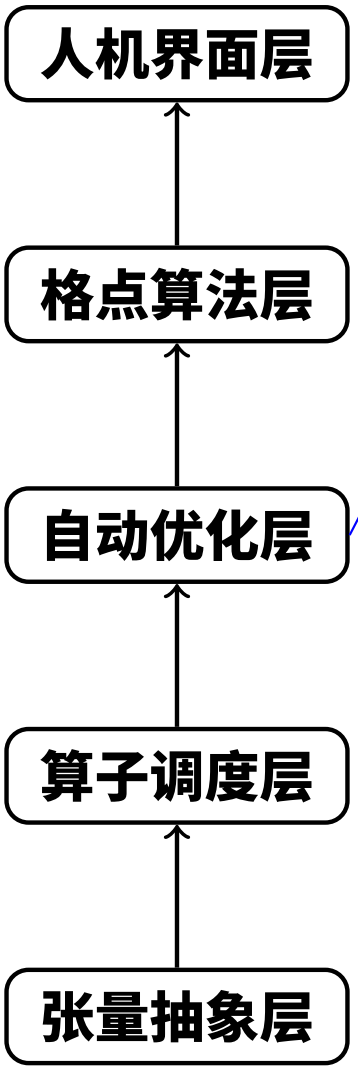
```
load_op(print) < [pi] < load_op(add) < [pi term]

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

[s] < load_op(mul) < [s -1.0]
[ia] < load_op(inv) < [a] < load_op(add) < [a 2.0]
[ib] < load_op(inv) < [b] < load_op(add) < [b 2.0]
[ic] < load_op(inv) < [c] < load_op(add) < [c 2.0]

[pi]  < load_op(init) < [3.0]
[s] < load_op(init) < [4.0]
[a] < load_op(init) < [2.0]
[b] < load_op(init) < [3.0]
[c] < load_op(init) < [4.0]
```

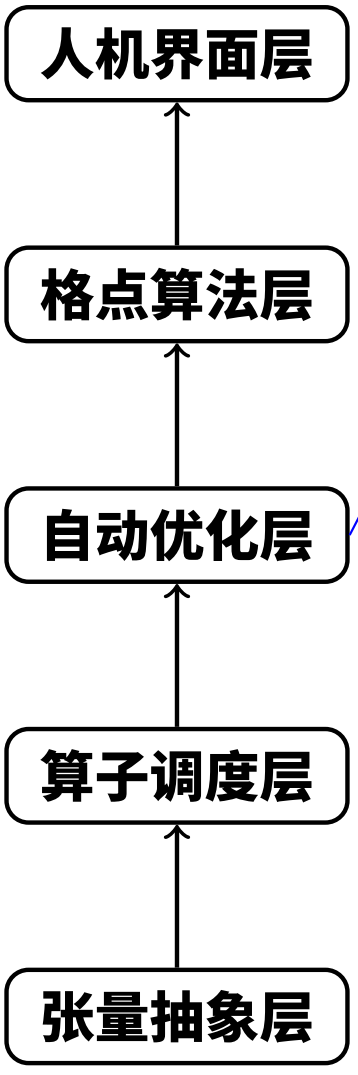
格点 QCD 新软件框架



自动优化的设计思考

- 自动优化的关键：优化空间
- 好的编程模型和软件框架，应该把计算过程的“有关部分”和“无关部分”充分分开，后者就是优化空间
- 应当为优化空间构建适当的表示：
  - 参数表、变换图、大模型参数……

格点 QCD 新软件框架



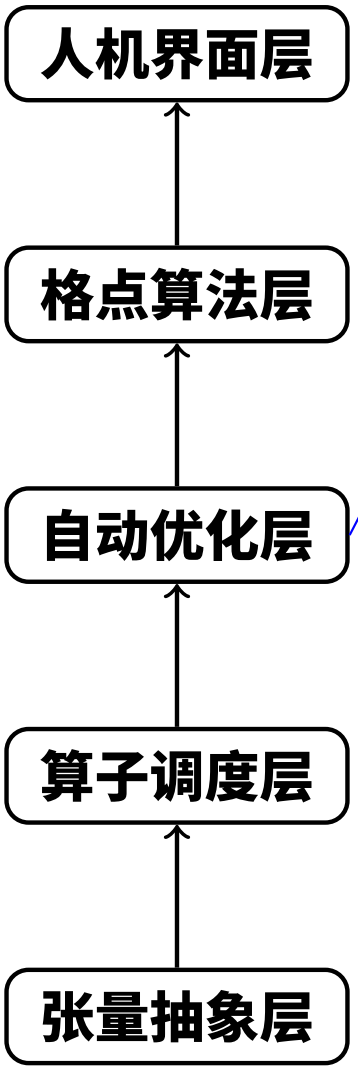
自动优化的设计思考

- 自动优化的关键：优化空间
- 好的编程模型和软件框架，应该把计算过程的“有关部分”和“无关部分”充分分开，后者就是优化空间
- 应当为优化空间构建适当的表示：
  - 参数表、变换图、大模型参数……

优化空间的图表示

- MetaTensor 的数据描述和 IR、DDQ 的算子图，都是图的顶点；对它们的等价变换，构成图的边
- 自动优化问题转变为图搜索问题
  - 深度/广度优先搜索、A* 搜索、遗传算法、……

格点 QCD 新软件框架



自动优化的设计思考

- 自动优化的关键：优化空间
- 好的编程模型和软件框架，应该把计算过程的“有关部分”和“无关部分”充分分开，后者就是优化空间
- 应当为优化空间构建适当的表示：
  - 参数表、变换图、大模型参数……

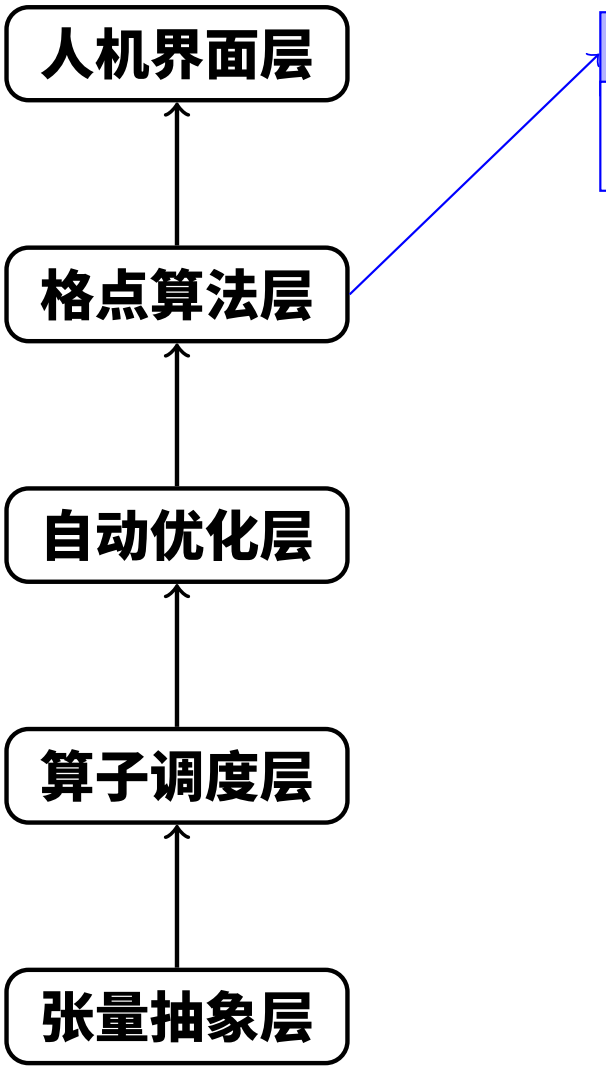
优化空间的图表示

- MetaTensor 的数据描述和 IR、DDQ 的算子图，都是图的顶点；对它们的等价变换，构成图的边
- 自动优化问题转变为图搜索问题
  - 深度/广度优先搜索、A* 搜索、遗传算法、……

针对树结构的通用优化器：Trene

- 采用遗传算法，变异是无损的
- 可用于代数表达式或程序 AST
  - 正在测试在量子计算中的应用

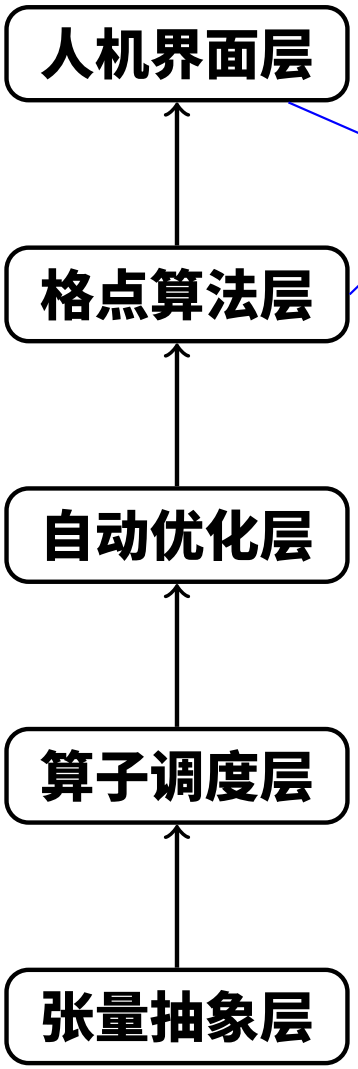
格点 QCD 新软件框架



应用软件的设计思考

- 从更抽象的层面上审视应用需求，构建可重用、可移植、高性能的代码

格点 QCD 新软件框架



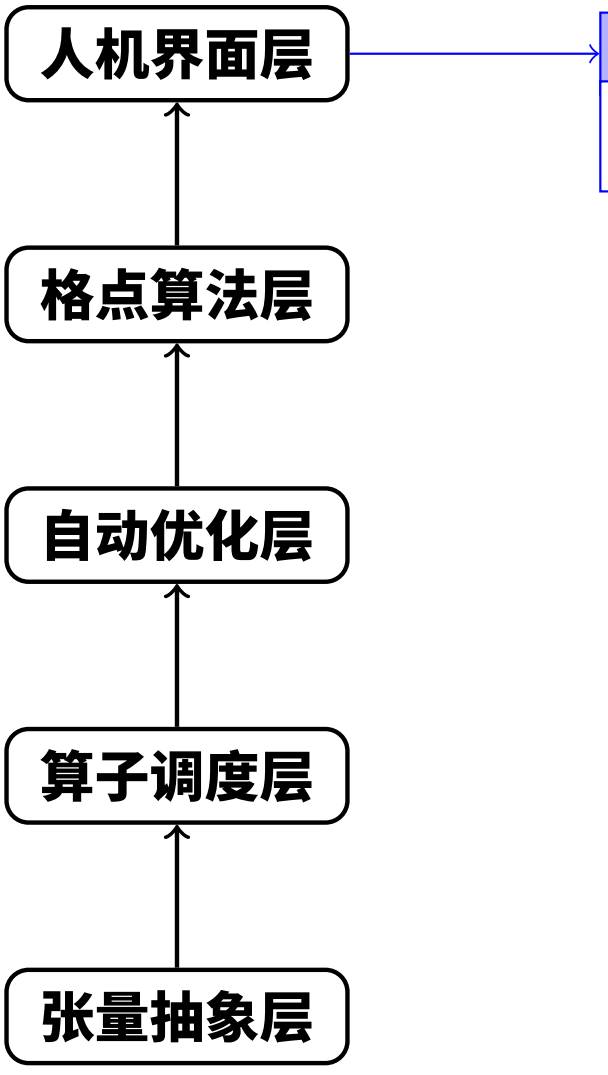
应用软件的设计思考

- 从更抽象的层面上审视应用需求，构建可重用、可移植、高性能的代码

格点 QCD 应用软件开发

- 格点 QCD 算法归结为：张量表达式计算、循环迭代
  - 所有张量计算核心都用 MetaTensor 自动产生代码，变成 DDQ 算子
  - 把算法中的循环迭代的流程用 DDQ 脚本构建为图，组合为更大的算子
  - 层迭构建不同尺度的算子，直到描述完整的计算任务
- 用自动优化工具透明地优化计算任务
- 把计算任务提交给部署在各平台上的 DDQ 运行

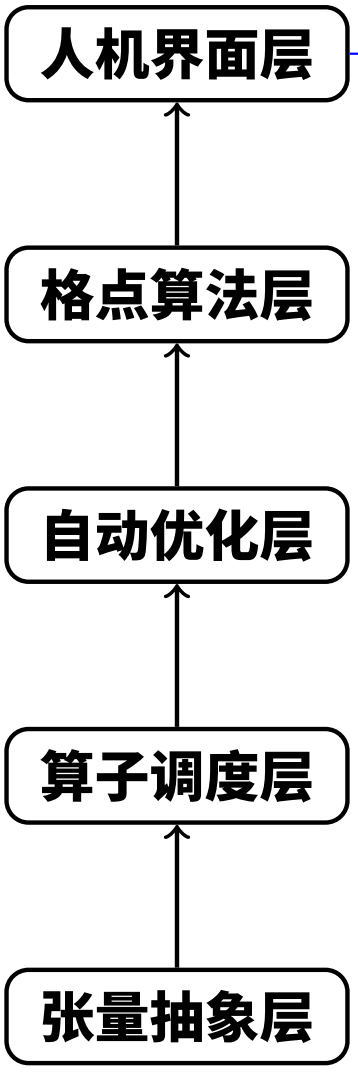
格点 QCD 新软件框架



人机界面的设计思考

- 好的编程模型，应该把计算过程的“有关部分”和“无关部分”充分分开，人机界面只操作前者

格点 QCD 新软件框架



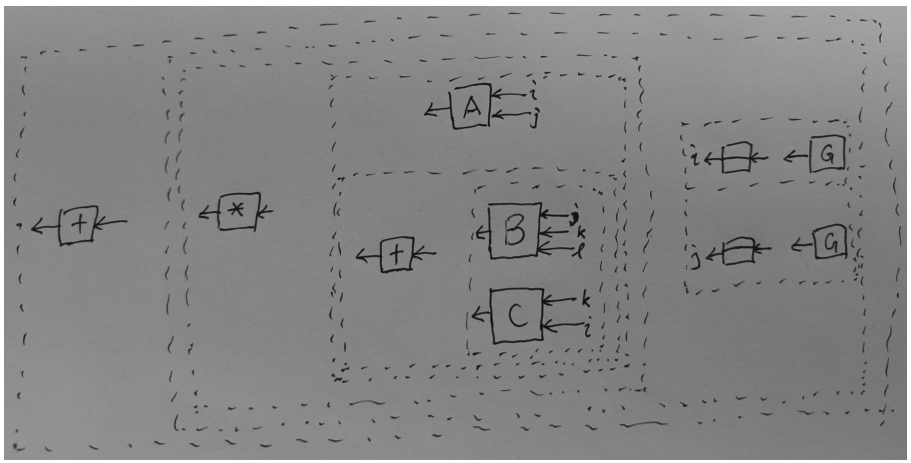
人机界面的设计思考

- 好的编程模型，应该把计算过程的“有关部分”和“无关部分”充分分开，人机界面只操作前者

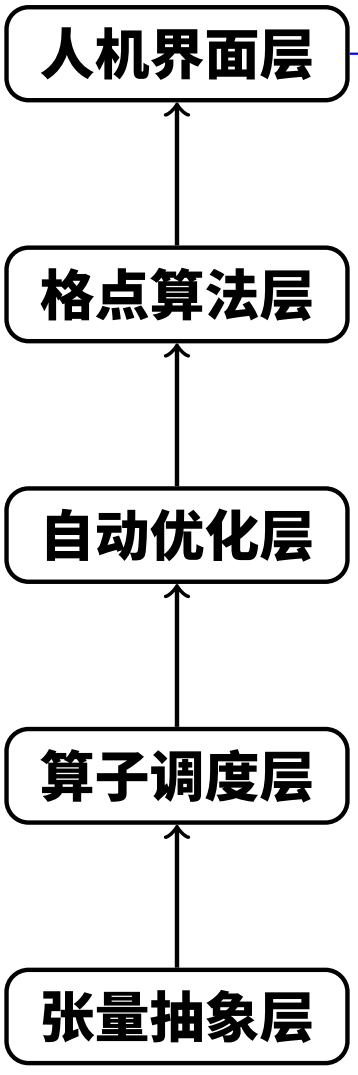
x 语言

- x 语言是一个自主设计的新编程语言
- 极简原则、纯函数式、静态强类型
- 只有一种数据/程序类型：块
- 适合图形界面操作
- 正在开发基于 HTML5 的人机界面

x 语言示例：



格点 QCD 新软件框架



人机界面的设计思考

- 好的编程模型，应该把计算过程的“有关部分”和“无关部分”充分分开，人机界面只操作前者

DDQ 脚本环境

- DDQ 脚本环境用于编辑 DDQ 计算任务
- 脚本是强类型的，可以保证 DDQ 运行时的类型安全
- 可以通过脚本，将算子组合为更大的算子，并储存在算子库 (oplib) 里
- 在 x 语言成熟后，DDQ 脚本可以从 x 语言生成

DDQ 脚本示例：

```
add.ddq:

newop.f = load_so(arithmetic/add)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double) load_type(double)]
-------------------------------------------------------------------
mul.ddq:

newop.f = load_so(arithmetic/mul)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double) load_type(double)]
-------------------------------------------------------------------
inv.ddq:

newop.f = load_so(arithmetic/div)
newop.processor = "direct"
[load_type(double)] < newop < [1.0 load_type(double)]
-------------------------------------------------------------------
mul4.ddq:

[s] < newop < [a b c d]
[s] < load_op(mul) < [abc d]
[abc] < load_op(mul) < [ab c]
[ab] < load_op(mul) < [a b]
-------------------------------------------------------------------
```

```
init.ddq:

newop.f = load_so(arithmetic/init)
newop.processor = "direct"
[load_type(double)] < newop < [load_type(double)]
-------------------------------------------------------------------
print.ddq:

newop.f = load_so(arithmetic/print)
newop.processor = "direct"
[] < newop < [load_type(double)]
-------------------------------------------------------------------
calc_pi.ddq:

load_op(print) < [pi] < load_op(add) < [pi term]

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

[s] < load_op(mul) < [s -1.0]
[ia] < load_op(inv) < [a] < load_op(add) < [a 2.0]
[ib] < load_op(inv) < [b] < load_op(add) < [b 2.0]
[ic] < load_op(inv) < [c] < load_op(add) < [c 2.0]

[pi]  < load_op(init) < [3.0]
[s] < load_op(init) < [4.0]
[a] < load_op(init) < [2.0]
[b] < load_op(init) < [3.0]
[c] < load_op(init) < [4.0]
```

总结

- 我们设计并正在研发一整套软件框架
- 编程模型适合包括格点 QCD 在内的各种科学计算
  - 包含一个通用的自动生成代码的 Metatensor，用 x 语言编写算子
  - 包含一个通用的运行时调度器 DDQ，用脚本构建工作流
  - 为未来的自动优化算法研发预留了接口和平台
  - 未来应可以作为计算集群的重要基础设施
- 目前的研发工作还在进行中，将陆续发布各模块的最新版本
- 欢迎大家参与，共同开发！

谢谢！