超算：从应用研究到基础软件研发

李宇轩

超算领域应用特征

- 传统的超级计算主要面向科学计算领域，该领域往往可以通过更高的时空分辨率提升价值，有着“明确”的优化目标
- 和互联网（微服务、云原生）等场景无法统一
- 计算有更好的并行度，可以使用GPGPU、many core等架构
- 网络互联基于近乎L2级的互联（比如InfiniBand），并大量采用DPDK、DPU offload等技术

7 dwarfs

HPC高性能计算：

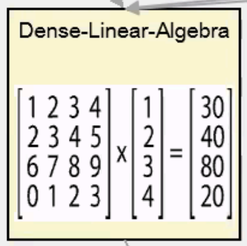
应用

EDA	制造	CFD	气象/海洋	油气	生命科学	计算化学	天文物理	粒子物理
• 电路设计 • 电路仿真	• 工业设计 • 模拟/验证	• 流体设计 • 结构设计	• 气象/气候 • 环境	• 油藏模拟 • 地质资料	• 细胞架构 • 药物研究	• 新材料	• 暗物质研究 • 星体发现	• 新粒子发现

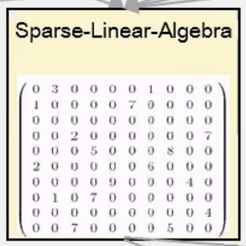
计算模式

- 2004：7种
- 2006：扩展到13种

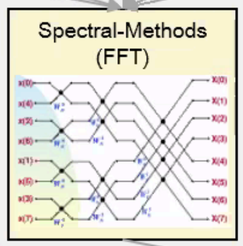
Dense-Linear-Algebra



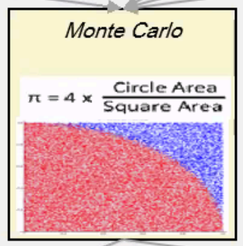
Sparse-Linear-Algebra



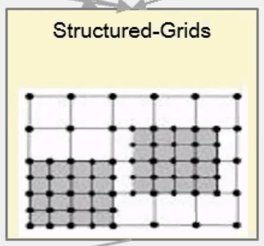
Spectral-Methods (FFT)



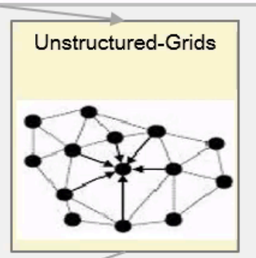
Monte Carlo



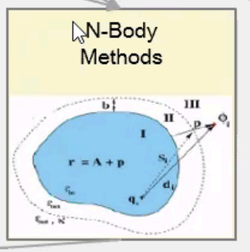
Structured-Grids



Unstructured-Grids



N-Body Methods



计算特征

Compute Bound | Memory BW Bound | Memory latency Bound | I/O Bound

一种处理器无法最佳满足所有HPC应用

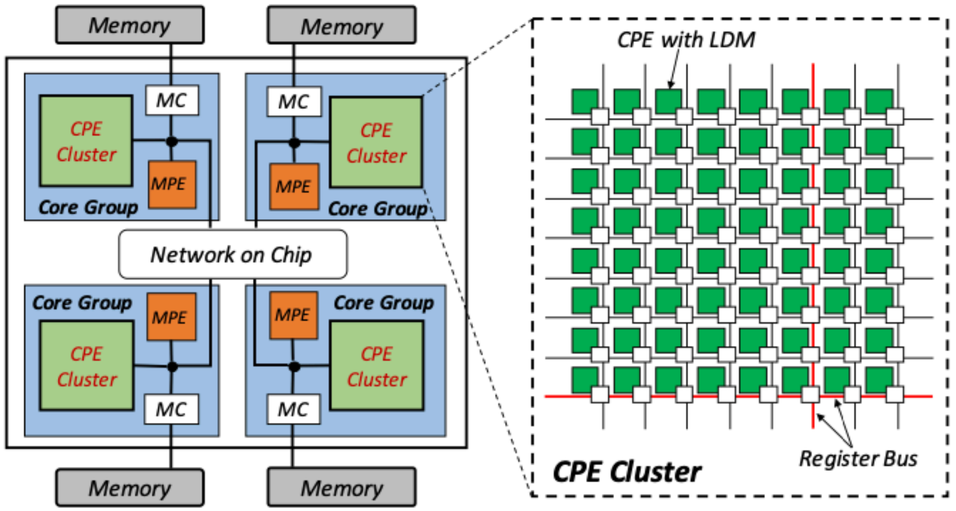
处理器

CPU | GPU | ASIC/IPcore | FPGA | ...

神威超算简介

神威 · 太湖之光

1. 异构众核架构

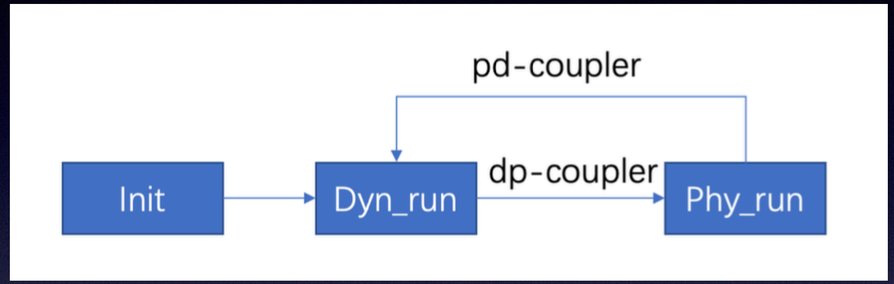


2. 纯软件控制的 64KB LDM 局存
   - 支持核间通信
3. 特有的申威架构指令集

大气模式

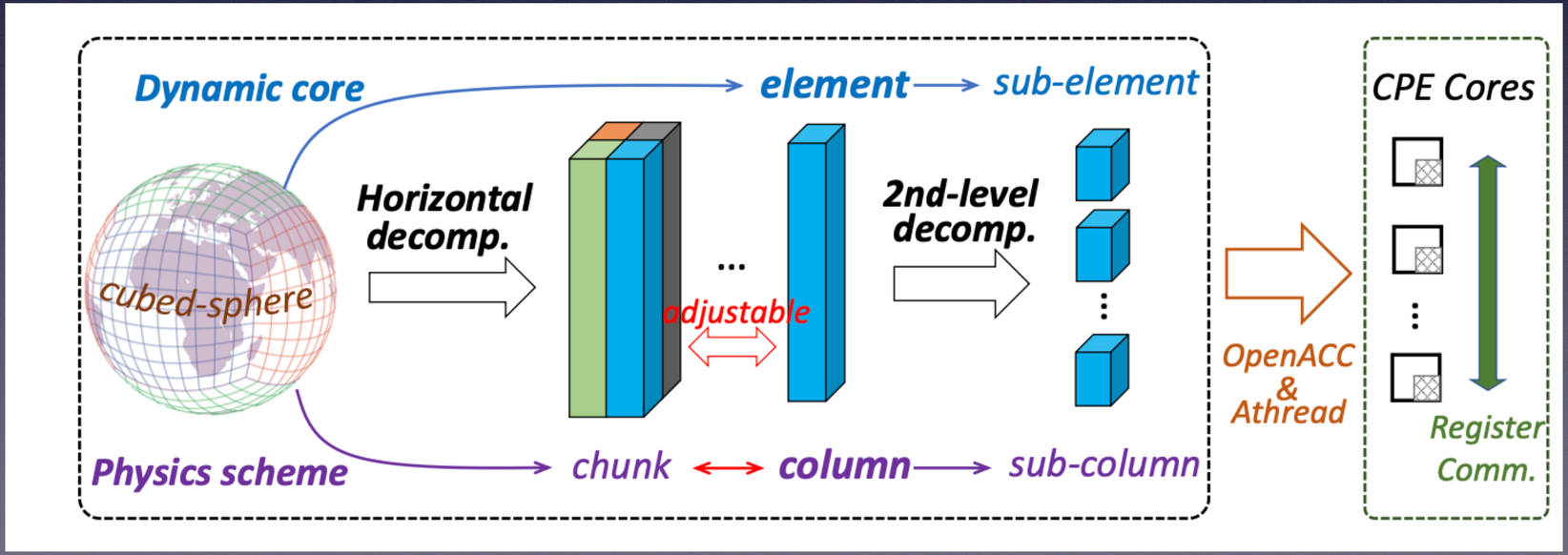
大气模式

- 应用特征：
  - 结构化网格，代码100w行，fortran
  - 动力框架和物理过程有显著差别，模块众多
- 核心矛盾：如何用可控的工作量完成10w+行代码的从核化
  - 可控，降低门槛的优化架构设计
  - 工具链体系完善，提升研发流程



通用并行方法

- 由于神威3TFlops vs 120GB/s，可以直接放弃向量化等计算侧优化
- 追求绝大多数核心的从核并行化

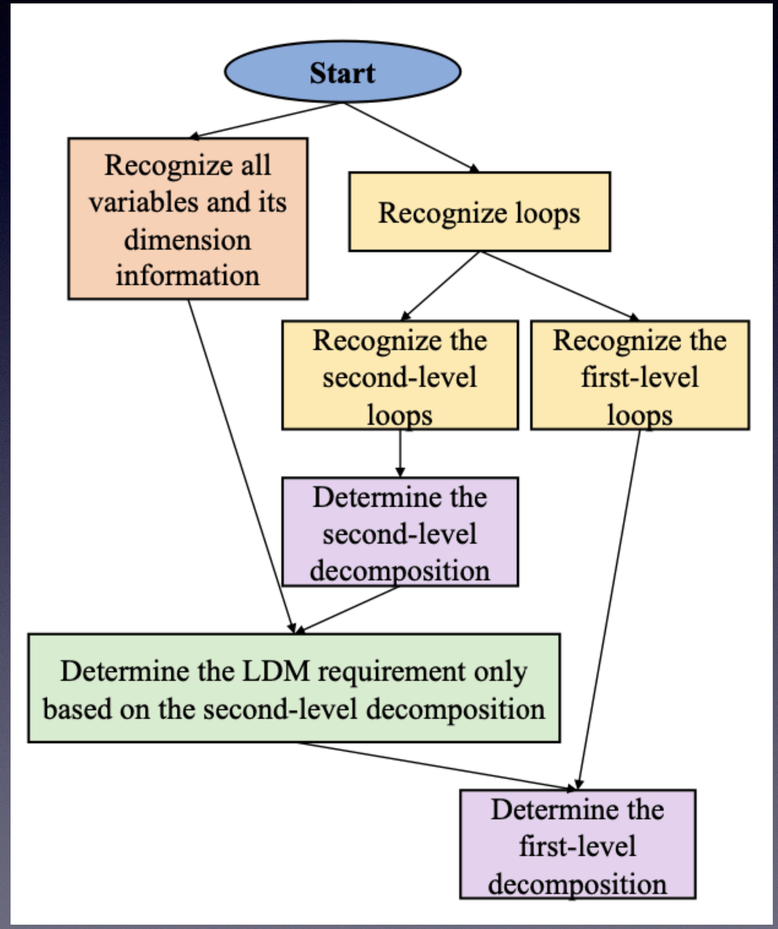


通用并行方法·物理过程

- 物理框架的众核并行方法
  - 在 chunk 维度上进行切分
    - 大部分的 kernel，包括云宏物理、深对流等等
  - 可配置二维划分，第二维为垂直层
    - 以微物理模块为主
  - 可配置二维划分，第二维为额外维度
    - 以短波辐射、长波辐射模块为代表
    - 划分波段，通过寄存器通信归约结果

通用并行方法·动力框架

- 为不同kernel的并行优化提供了统一的指导



通用并行方法·物理过程

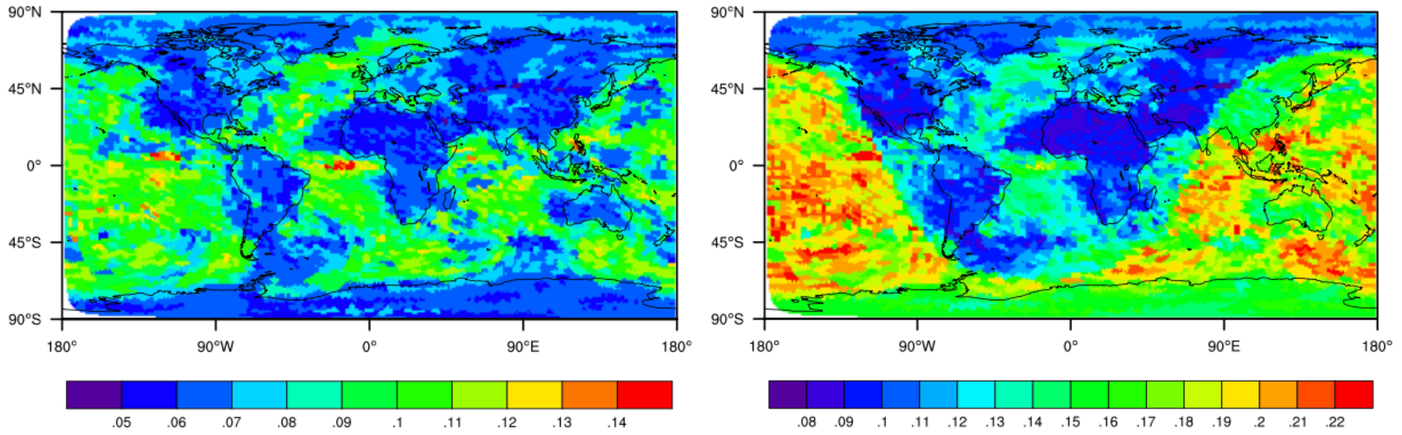
物理框架的众核并行方法

1. 在 chunk 维度上进行切分
   - 大部分的 kernel，包括云宏物理、深对流等等
2. 可配置二维划分，第二维为垂直层
   - 以微物理模块为主
3. 可配置二维划分，第二维为额外维度
   - 以短波辐射、长波辐射模块为代表
   - 划分波段，通过寄存器通信归约结果

静态负载均衡方法

静态负载均衡方法

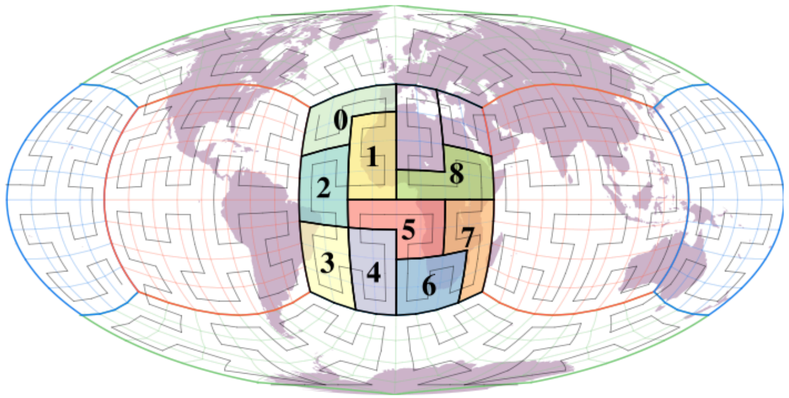
- 当进程数达到超大规模，物理过程负载不均开销超过动力框架领域通信
- 动态的负载均衡策略将引入过大的通信拥塞
- 物理过程负载特征：热寒带不均、昼夜不均、海陆不均

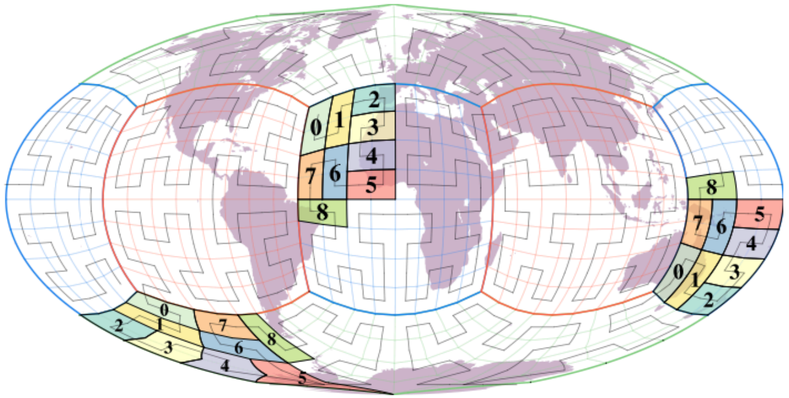


静态负载均衡方法

静态负载均衡方法

- 基于 round-robin 的分配策略
- 三大负载不均都得到了缓解
- 动力框架的领域通信开销增加可以接受





应用开发支撑技术

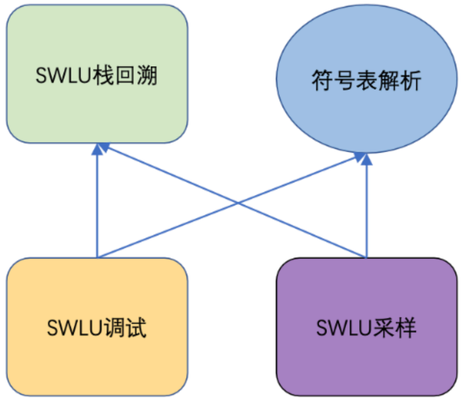
基于预扫描的编译构建提速技术

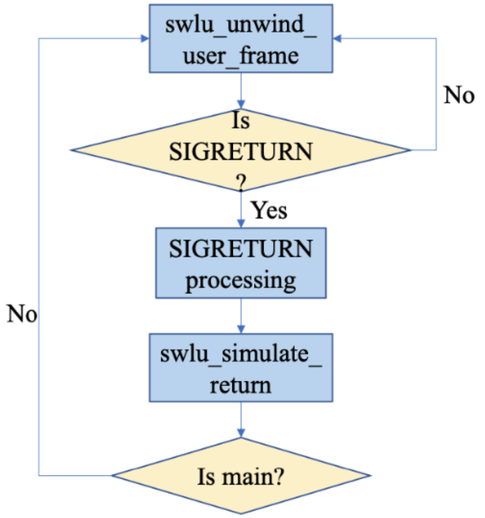
- OpenACC 编译器对于大 kernel 的支持很慢
- 编译工具链套壳，利用正则表达式识别 OpenACC 模式，对于不需要 OpenACC 的文件不再使用 OpenACC 编译器

应用开发支撑技术

基于采样和栈回溯的性能分析技术

- 栈回溯技术路线
  - 无第三方软件依赖，高效
  - 反向执行和正向模拟执行



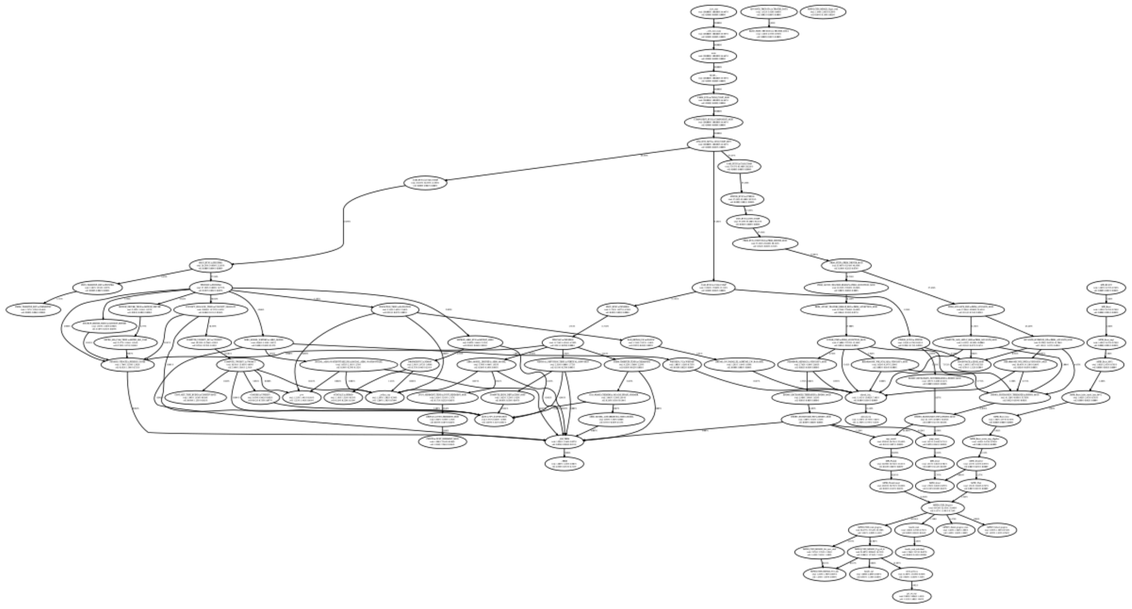


- 采样基于 linux 信号机制，并同时支持的基于信号中断的调试功能

应用开发支撑技术

基于采样和栈回溯的性能分析技术

- 通过详细的分析，优化了初始化时间
  - 定位到若干 Fortran 库函数优化不佳导致的性能问题
  - 定位到 MPI malloc 引起的 0 号进程性能问题，使用专用内存池解决
- 并支撑其他多个领域应用的开发工作



应用开发支撑技术

二进制一致验证方法

1. 该技术的价值
   - 解决了产业级科学应用跨平台精度验证的部分问题
   - 减少了跨平台移植的调试成本
2. 技术的两部分
   - 跨平台跨编译器的二进制一致性
   - 神威主从核的二进制一致性

光量子计算

光量子计算

- 用作带阈值的高斯玻色采样的验证
- 核心公式：

$$Tor(A) = \sum_{Z \in P_N} (-1)^{N-|Z|} \frac{1}{\sqrt{|det(I - A_Z)|}}, \tag{3}$$

where P_N is the power set of $1, 2, \ldots, N$, and thus, there are 2^N determinant terms in the summation [12]. The computational complexity of the Torontonian function is $O(2^N)$, which is the same as the permanent function in standard boson sampling with N photons.

- 核心问题：提供足够的精度和极致的性能
  - 通过精度分析，需要128-256位有效数字

并行框架

- get_next_mask 使用BitManip指令集（二进制取反）
- 基本上是尴尬并行的

Algorithm 3. Calculate $Torontonian(A)$

Require: Matrix A
Ensure: A is Hermitian positive definite

1: **function** $TorontonianA$
2: $result \leftarrow 0$
3: **for** $i_{|Z|} = 1 \rightarrow N$ **do**
4: get $mask_Z$ and $mask_ZEnd$ of the process or the thread
5: **while** $mask_Z \neq mask_ZEnd$ **do**
6: $A_Z \leftarrow \mathrm{GEN\_A_Z}(mask_Z)$
7: $det \leftarrow \mathrm{GET\_DETERMINANT}(I - A_Z)$
8: $result \leftarrow result + (-1)^{N-i_{|Z|}} \frac{1}{\sqrt{|det|}}$
9: $mask_Z \leftarrow \mathrm{GET\_NEXT\_MASK}(mask_Z)$
10: **end while**
11: **end for**
12: **return** $result$
13: **end Function**

定点精度库

Algorithm 1. 256-Bit Signed Multiplication

Require: 256-bit signed number a and b

function mul256a, b
- $a_1 \leftarrow \text{VSHFF}(0, a, 0\text{x}0\text{e}) \triangleright \text{SHUFFLE}(0, \text{a}, \{2, 3, 0, 0\})$
- $b_1 \leftarrow \text{VSHFF}(0, b, 0\text{x}0\text{e}) \triangleright \text{SHUFFLE}(0, \text{b}, \{2, 3, 0, 0\})$
- $a_s \leftarrow \text{VSHFF}(0, a, 0\text{xff}) \triangleright \text{SHUFFLE}(\text{a}, \text{a}, \{3, 3, 3, 3\})$
- $b_s \leftarrow \text{VSHFF}(0, b, 0\text{xff}) \triangleright \text{SHUFFLE}(\text{b}, \text{b}, \{3, 3, 3, 3\})$
- $a_x \leftarrow \text{VSELLT}(b_s, a, 0)$
- $b_x \leftarrow \text{VSELLT}(a_s, b, 0)$
- $c \leftarrow \text{UMULQA}(a, b)$
- $c_{10} \leftarrow \text{UMULQA}(a_1, b)$
- $c_{10} \leftarrow \text{UMULQA}(a, b_1)$
- $c_{11} \leftarrow \text{UMULQA}(a_1, b_1)$
- $c \leftarrow \text{SRLOW}(c, SF)$
- $c_{10} \leftarrow \text{SRLOW}(c_{10}, SF - 128)$
- $c_{01} \leftarrow \text{SRLOW}(c_{01}, SF - 128)$
- $c_{11} \leftarrow \text{USUBO\_CARRY}(c_{11}, a_x)$
- $c_{11} \leftarrow \text{USUBO\_CARRY}(c_{11}, b_x)$
- $c_{11} \leftarrow \text{SLLOW}(c_{11}, 256 - SF)$
- $c \leftarrow \text{UADDO\_CARRY}(c, c_{10})$
- $c \leftarrow \text{UADDO\_CARRY}(c, c_{01})$
- $c \leftarrow \text{UADDO\_CARRY}(c, c_{11})$
- **return**c

end Function

Algorithm 2. 128-Bit Signed Multiplication

Require:128-bits signed number a and b

function mul128a, b
- $c \leftarrow \text{UMULQA}(a, b)$
- $a_s \leftarrow \text{VSHFF}(a, 0, 0\text{x}55) \triangleright \text{SHUFFLE}(0, \text{a}, \{1, 1, 1, 1\})$
- $b_s \leftarrow \text{VSHFF}(b, 0, 0\text{x}55) \triangleright \text{SHUFFLE}(0, \text{b}, \{1, 1, 1, 1\})$
- $a_x \leftarrow \text{VSELLT}(b_s, a, 0)$
- $b_x \leftarrow \text{VSELLT}(a_s, b, 0)$
- $c_1 \leftarrow \text{UADDO\_CARRY}(a_x, b_x)$
- $c \leftarrow \text{SRLOW}(c, SF)$
- $c_1 \leftarrow \text{SLLOW}(c_1, 128 - SF)$
- $c \leftarrow \text{USUBO\_CARRY}(c, c_1)$
- **return**c

end Function

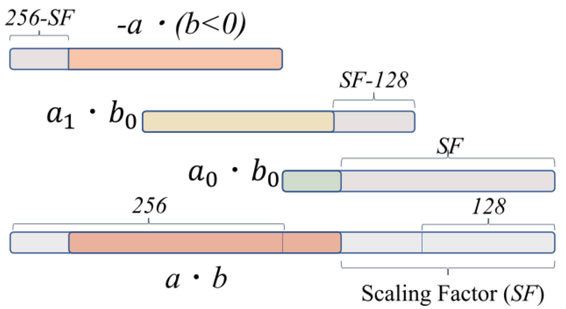


Fig. 4. Scaling factor implemented by shifting.

指令调度优化

- 四维状态压缩动态规划
  - 乘法作为特殊状态
  - 利用指令对称性
- 同时解决指令调度和寄存器分配两个编译问题
  - 有constrain programming的通用方案

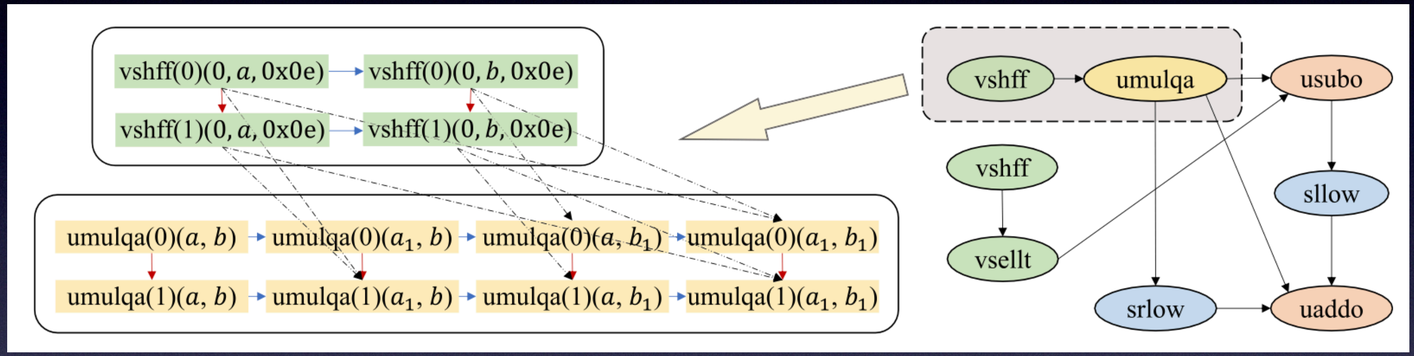
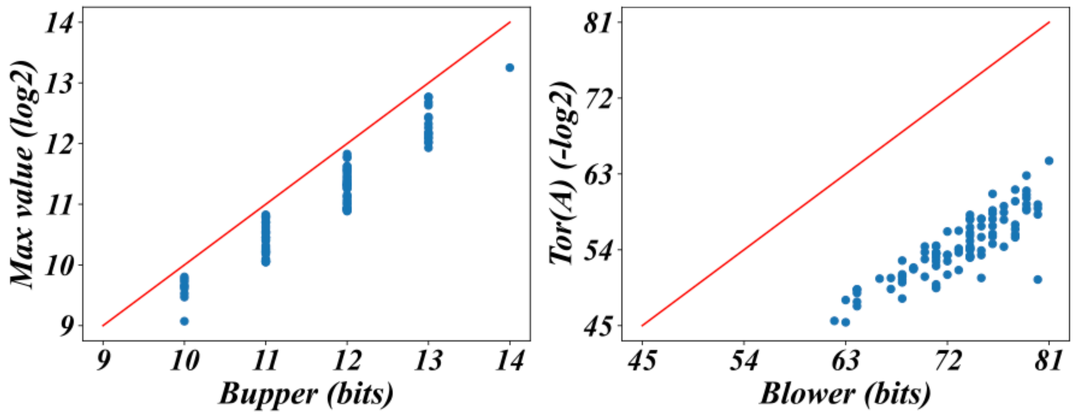


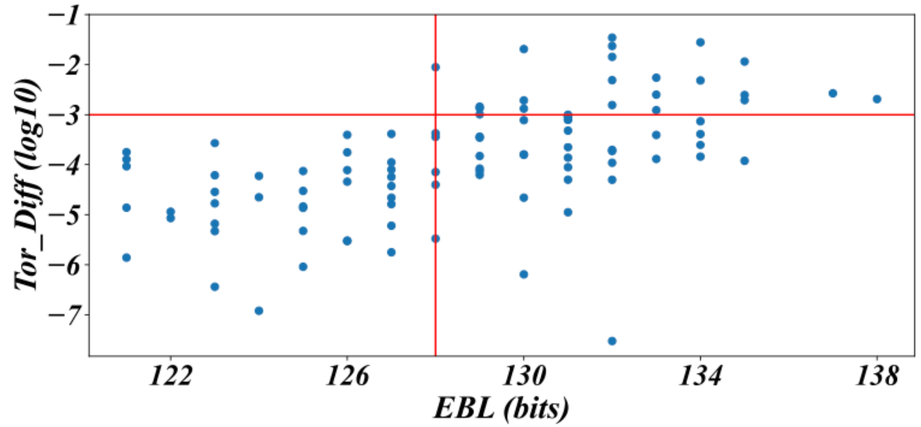
TABLE 2
Results of the Shortest-Path-Based Methods on 128-bit and 256-bit Kernels

Precision	Instructions	Naive-Clocks	Opt-Clocks	Opt-RegNums
128 bits	32	68	34	9
256 bits	62	130	64	14

性能结果



(a) Evaluation of B_{upper} (b) Evaluation of B_{lower}



(c) Evaluation of EBL

Fig. 15. Experimental results of 100 random real-world matrices with $n \in [39, 43]$.

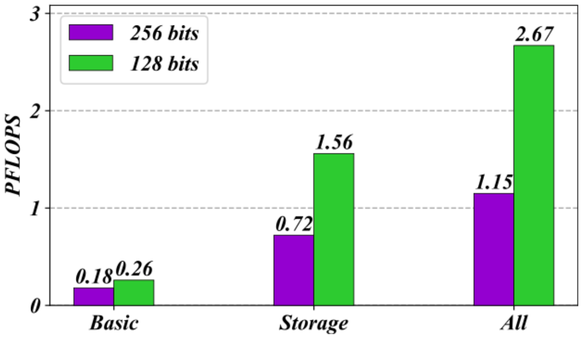


Fig. 13. PFLOPS of time cost of each version with different optimizations when $N = 50$.

性能结果

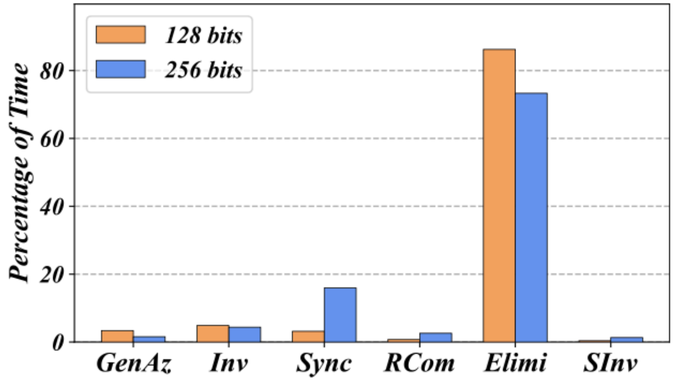
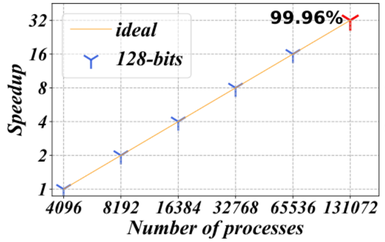
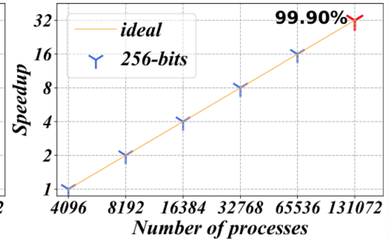


Fig. 14. Percentage of time cost of each part.

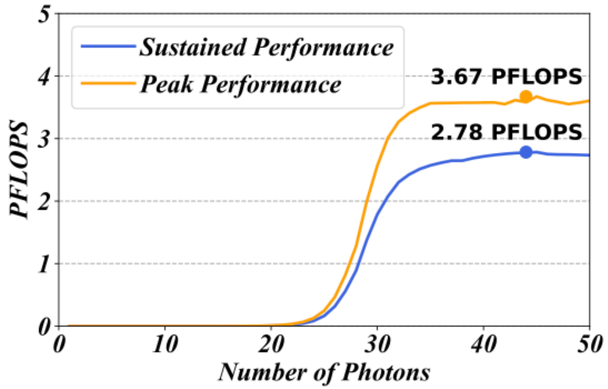


(a)

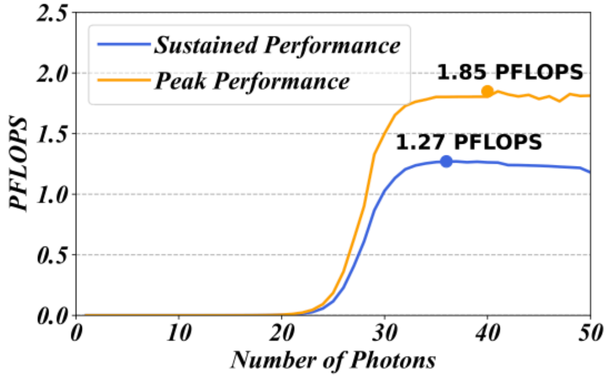


(b)

Fig. 12. Strong scalability of (a) 128 bits; (b) 256 bits.



(a)



(b)

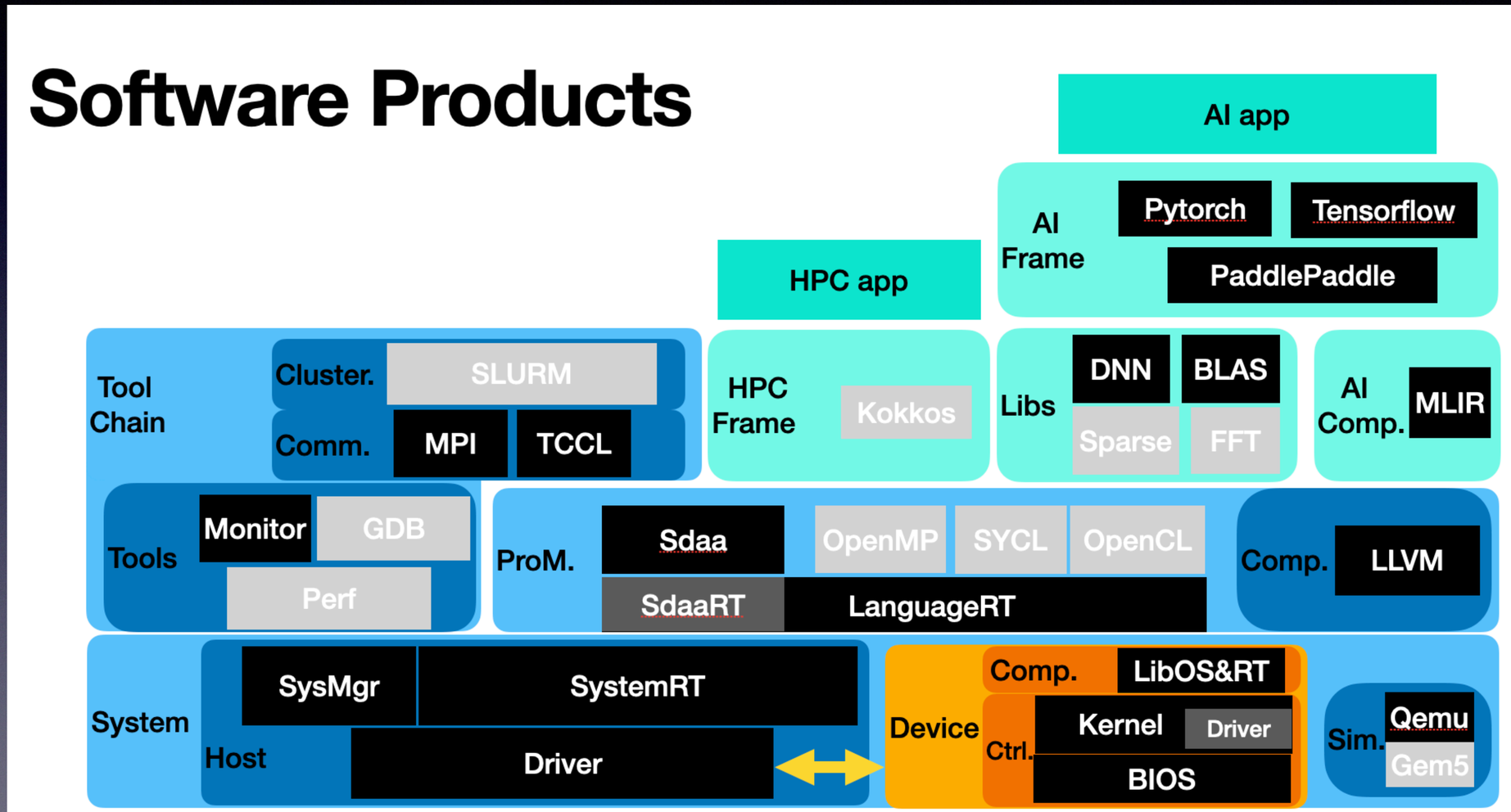
Fig. 11. Peak and Sustained Performance of (a) 128-bit precision; (b) 256-bit precision.

基础软件

从应用到基础软件

- 方向转变原因：
  - 想深入到计算机技术本身
  - 想深入了解企业的运作
- 跨方向优势：
  - 作为基础软件的资深用户，知道研发流程以及痛点，可以用于做方案取舍
  - 相当于具有产品经理角色的产品经验

异构计算基础设施


Software Products
AI app
AI Frame
Pytorch
Tensorflow
PaddlePaddle
HPC app
Tool Chain
Cluster.
SLURM
Comm.
MPI
TCCL
HPC Frame
Kokkos
Libs
DNN
BLAS
Sparse
FFT
AI Comp.
MLIR
Tools
Monitor
GDB
Perf
ProM.
Sdaa
OpenMP
SYCL
OpenCL
SdaaRT
LanguageRT
Comp.
LLVM
System
Host
SysMgr
SystemRT
Driver
Device
Comp.
LibOS&RT
Ctrl.
Kernel
Driver
BIOS
Sim.
Qemu
Gem5

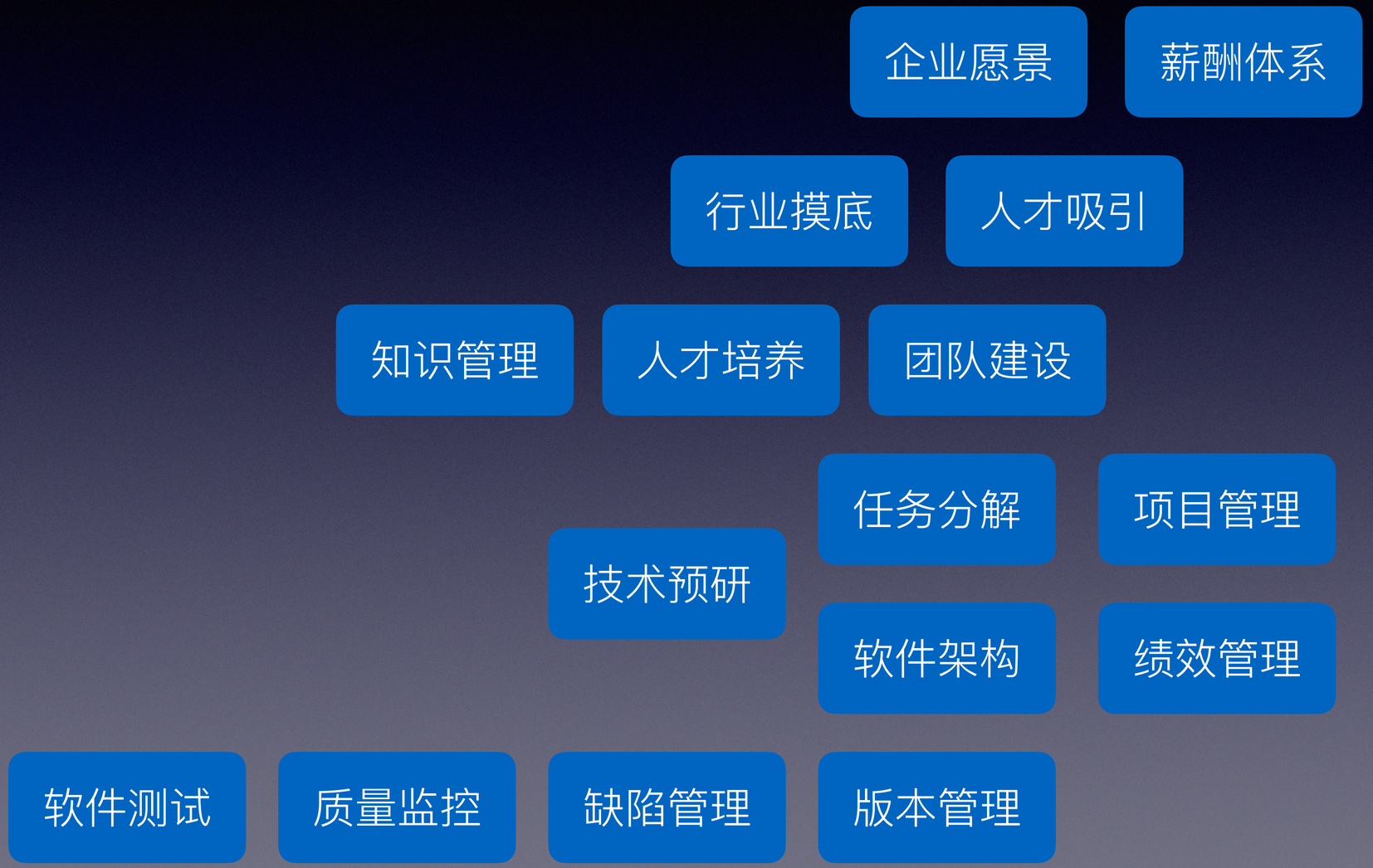
异构计算vs传统服务器

- 由于user-defined cache的存在（shared memory/SPM/LDM），软件的递归层数难以放开，函数调用不宜太复杂，并且受制于host端生态，形成了以module为单位的链接体系
- 传统的调度控制任务天生适合于host端执行，出现了命令队列调度、虚拟、物理内存管理向host端移动
- 由于硬件本身快速迭代，为了尽可能隔离硬件层的变化，形成了PTX中间表示，WMMA接口等技术

关于软件研发

- 相比科研阶段工作的优化目标变了，工作优先级更加重要
- 软件架构设计变得异常重要，深刻理解了软件工程的重要性

错综复杂的研发体系


企业愿景
薪酬体系
行业摸底
人才吸引
知识管理
人才培养
团队建设
任务分解
项目管理
技术预研
软件架构
绩效管理
软件测试
质量监控
缺陷管理
版本管理

科研感悟

谈谈科研价值

- 大气、量子等太湖之光下的工作
  - 价值：更多是工程价值，且被用于解决实际的科学科研问题
  - 不足：解决方案本身不够计算机学科前沿，没有形成有效的积累
- 回归科研/科技价值
  - 发现一个新的有价值的问题，并且将其做到极致
  - 如何找到“真·问题”，对上下游有常识性的认知

回顾科研

- 如何找到“真·问题”，对上下游要有常识性的认知
  - 不熟悉上下游和同行甚至工业界现状，很难判断一个工作是否真的有长期价值
- 用**常识**快速判断方向价值
  - 每当你发现一个新的问题和思路，先想想这个问题的背景以及普遍性，再大致判断这个问题已经被解决的可能性

技术/行业壁垒

- 真的壁垒不在于认知，而在于经验
  - 认知思维模式是能跨领域复制的
- 不要高估了解一个领域核心逻辑的难度，不要低估“掌握”一个领域的难度
- 好的模式应该是深耕一个方向，并且了解该方向完整的产业链条（乃至于更大的范围）
  - 这样才能在方向开创者的视角，在正确的位置解决了真问题

谢谢大家