GPU 在高能物理中的应用

蔡浩

武汉大学

hcai@whu.edu.cn

GPU 在高能物理实验中有以下几个主要应用：

模拟和重建
高能物理实验通常需要进行大量的模拟和重建工作，以研究粒子的行为和相互作用。GPU 的并行计算能力可以加速模拟和重建算法，缩短计算时间。

机器学习
机器学习在高能物理实验中有广泛的应用，如粒子识别、事件分类、信号处理等。GPU 的并行计算能力可以加速机器学习算法的训练和推断，提高模型的性能和效率。

实时处理
高能物理实验中有些应用需要实时处理数据，如在线监测、触发系统等。GPU 的高性能计算能力和并行处理能力可以满足实时处理的需求。

数据处理和分析
高能物理实验产生的数据量巨大，需要进行复杂的数据处理和分析。GPU 的并行计算能力可以加速数据处理过程，提高数据分析的效率。

分波分析（Partial Wave Analysis）

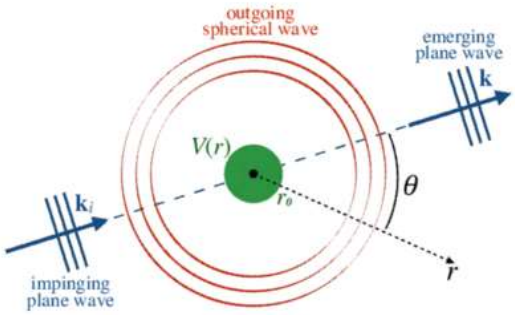
- The stationary Schrödinger equation

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(r)\right]\Psi(\mathbf{r}) = E\Psi(\mathbf{r})$$

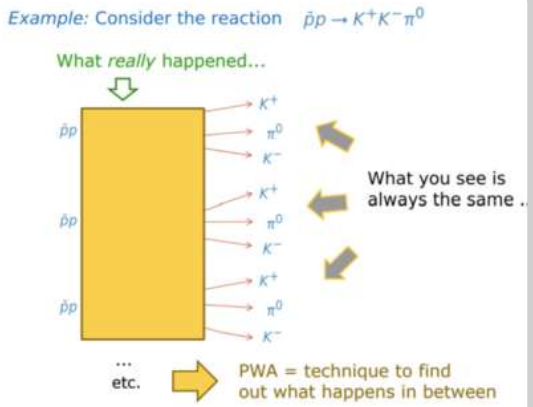
$$\Psi(\mathbf{r}) \to \Psi^{(+)}(\mathbf{r}) = \exp(ikz) + f(\theta,k)\frac{\exp(ikr)}{r}$$

Differential cross section

$$\frac{d\sigma}{d\Omega} = |f(\theta,k)|^2$$



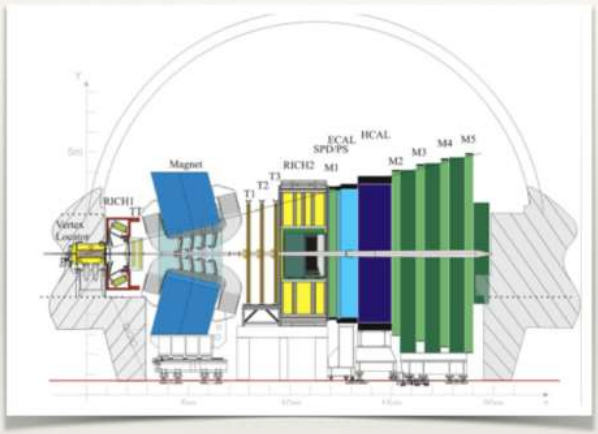
- 分波分析的特点
  - 分波分析强子谱学是研究强子物理和量子色动力学的重要工具。
  - 分波分析利用事例全部物理信息，直接拟合振幅，可以处理共振态的干涉、叠加，可以精确测量强子共振态的物理性质，是研究强子谱学的重要工具。
  - 分波分析使用的数据数据量大，计算复杂，十分费时。

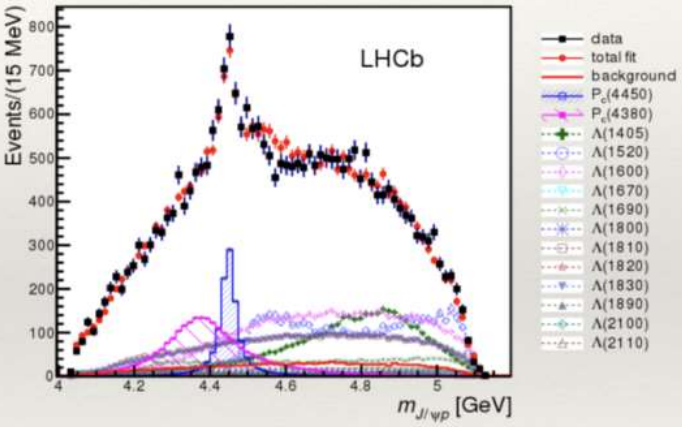


分波分析的典型例子：

大型强子对撞机底夸克实验

Large Hadron Collider beauty experiment (LHCb)

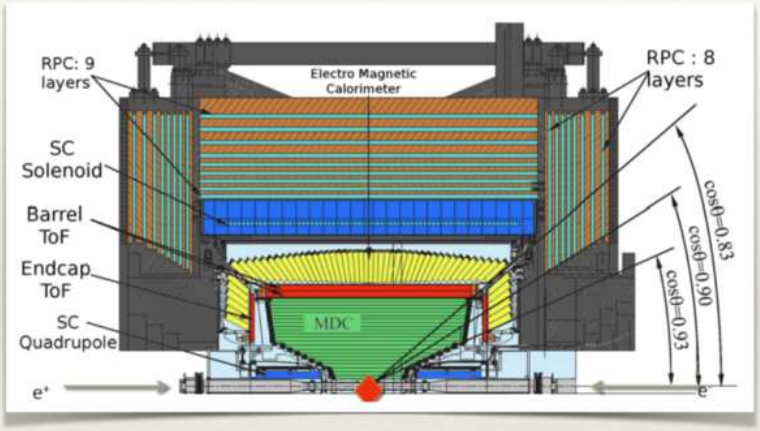


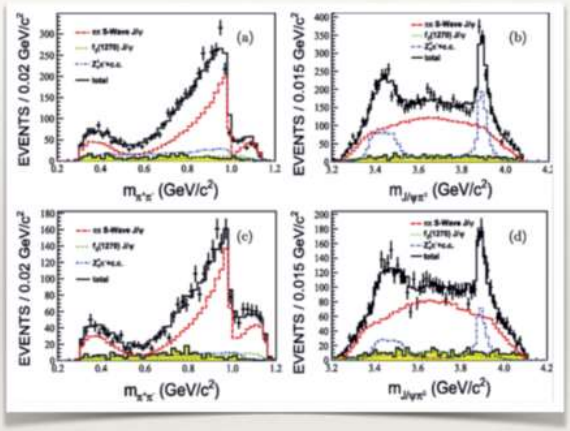


LHCb发现五夸克态入选2015年英国物理学会公布的
年度国际物理学领域的十项重大突破（Breakthrough of the Year）

北京谱仪III实验

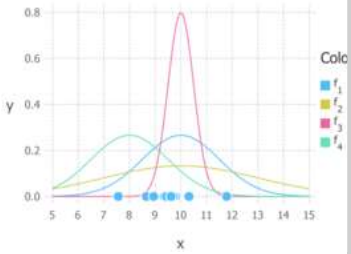
Beijing Spectrometer (BESIII) Experiment





“BESIII发现Zc(3900)”入选2013年度“中国科学十大进展”

最大似然估计（Maximum Likelihood Estimation）



- 最大似然估计的原理

  给定一个概率分布 D，已知其概率密度函数（连续分布）或概率质量函数（离散分佈）为 f_D，以及一个分佈参数 θ，我们可以从这个分布中抽出一个具有 n 个值的采样 $X_1, X_2, \ldots, X_n$，利用 f_D 计算出其似然函数：

  $$\mathrm{L}(\theta \mid x_1, \ldots, x_n) = f_\theta(x_1, \ldots, x_n).$$

  若 D 是离散分布，f_θ 即是在参数为 θ 时观测到这一采样的概率；若其是连续分布，f_θ 则为 $X_1, X_2, \ldots, X_n$ 联合分布的概率密度函数在观测值处的取值。一旦我们获得 $X_1, X_2, \ldots, X_n$，我们就能求得一个关于 θ 的估计。最大似然估计会寻找关于 θ 的最可能的值（即，在所有可能的 θ 取值中，寻找一个值使这个采样的“可能性”最大化）。从数学上来说，我们可以在 θ 的所有可能取值中寻找一个值使得似然函数取到最大值。这个使可能性最大的 $\hat{\theta}$ 值即称为 θ 的最大似然估计。由定义，最大似然估计是样本的函数。

最大似然拟合

最常见的连续概率分布是正态分布，其概率密度函数如下：

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

现在有 n 个正态随机变量的采样点，要求的是一个这样的正态分布，这些采样点分布到这个正态分布可能性最大（也就是概率密度积最大，每个点更靠近中心点），其 n 个 正态随机变量的采样的对应密度函数（假设其独立并服从同一分布）为：

$$f(x_1, \ldots, x_n \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}}$$

也可以写为：

$$f(x_1, \ldots, x_n \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2 + n(\bar{x}-\mu)^2}{2\sigma^2}\right),$$

这个分布有两个参数： μ, σ^2 . 有人可能会担心两个参数与上边的讨论的例子不同, 上边的例子都只是在一个参数上对可能性进行最大化。实际上, 在两个参数上的求最大 值的方法也差不多：只需要分别把可能性 $\mathrm{L}(\mu, \sigma)=f\left(x_1, \ldots, x_n \mid \mu, \sigma^2\right)$ 在两个参数上最大化即可。当然这比一个参数麻烦一些，但是一点也不复杂。使用上边例 子同样的符号, 我们有 $\theta=\left(\mu, \sigma^2\right)$.

最大化一个似然函数同最大化它的自然对数是等价的。因为自然对数 $\log$ 是一个连续且在似然函数的值域内严格递增的上凹函数。 [注意 : 可能性函数（似然函数）的自然 对数跟信息熵以及 Fisher 信息联系紧密。]求对数通常能够一定程度上简化运算，比如在这个例子中可以看到：

$$
\begin{aligned}
0 & =\frac{\partial}{\partial \mu} \log \left(\left(\frac{1}{2 \pi \sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^n\left(x_i-\bar{x}\right)^2+n(\bar{x}-\mu)^2}{2 \sigma^2}}\right) \\
& =\frac{\partial}{\partial \mu}\left(\log \left(\frac{1}{2 \pi \sigma^2}\right)^{\frac{n}{2}}-\frac{\sum_{i=1}^n\left(x_i-\bar{x}\right)^2+n(\bar{x}-\mu)^2}{2 \sigma^2}\right) \\
& =0-\frac{-2 n(\bar{x}-\mu)}{2 \sigma^2}
\end{aligned}
$$

这个方程的解是 $\widehat{\mu}=\bar{x}=\sum_{i=1}^n x_i / n$. 这的确是这个函数的最大值, 因为它是 μ 里头惟一的一阶导数等于零的点并且二阶导数严格小于零。

同理, 我们对 σ 求导, 并使其为零。

$$
\begin{aligned}
0 & =\frac{\partial}{\partial \sigma} \log \left(\left(\frac{1}{2 \pi \sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^n\left(x_i-\bar{x}\right)^2+n(\bar{x}-\mu)^2}{2 \sigma^2}}\right) \\
& =\frac{\partial}{\partial \sigma}\left(\frac{n}{2} \log \left(\frac{1}{2 \pi \sigma^2}\right)-\frac{\sum_{i=1}^n\left(x_i-\bar{x}\right)^2+n(\bar{x}-\mu)^2}{2 \sigma^2}\right) \\
& =-\frac{n}{\sigma}+\frac{\sum_{i=1}^n\left(x_i-\bar{x}\right)^2+n(\bar{x}-\mu)^2}{\sigma^3}
\end{aligned}
$$

这个方程的解是 $\widehat{\sigma}^2=\sum_{i=1}^n\left(x_i-\widehat{\mu}\right)^2 / n$.

因此，其关于 $\theta=\left(\mu, \sigma^2\right)$ 的最大似然估计为：

$$
\hat{\theta}=(\widehat{\mu}, \widehat{\sigma}^2)=\left(\bar{x}, \sum_{i=1}^n\left(x_i-\bar{x}\right)^2 / n\right)
$$

最优化方法（Optimization Method）

- 基于梯度计算的常用优化方法
  - 梯度下降法

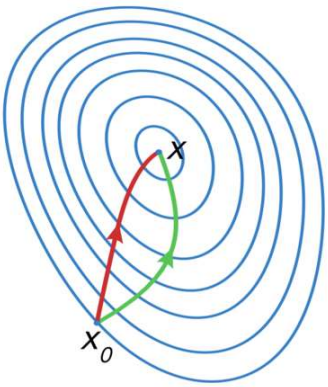
$$
x_{n+1}=x_n-\gamma_n \nabla f\left(x_n\right)
$$

  - 牛顿法

$$
x_{n+1}=x_n-\left[\mathbf{H} f\left(x_n\right)\right]^{-1} \nabla f\left(x_n\right)
$$

  - 拟牛顿方法

$$
x_{n+1}=x_n-B^{-1} \nabla f\left(x_n\right)
$$

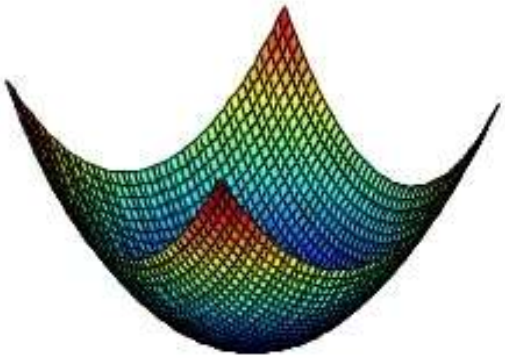


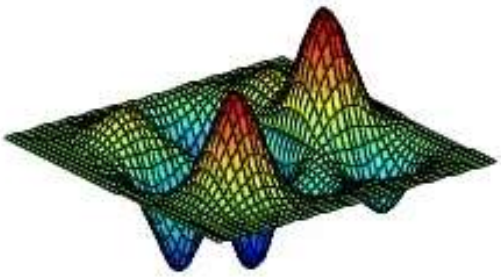
最速下降法 (绿色) 与牛顿法 (红色) 在求最小值问题上的比较 (带有步长)。可见牛顿法根据曲率选择了一条“快捷方式”。

🔥 高能物理使用的优化方法与人工智能优化方法的异同?

凸优化 vs. 非凸优化

Convex vs. Nonconvex Optimization





- Unique optimum: global/local.
- Multiple local optima
- In high dimensions possibly exponential local optima

How to deal with non-convexity?

🔥 如何解决非凸优化问题?

GPGPU (General-purpose computing on graphics processing units)

GPGPU（通用图形处理单元）是指利用图形处理器（GPU）进行通用计算任务的技术。GPU 最初是为了处理图形渲染而设计的，但是随着计算需求的增加，GPU 也被用来加速其他计算任务，例如科学计算、机器学习、密码学等。

GPGPU 的优势在于 GPU 具有大量的并行处理单元和高速的内存带宽，能够在短时间内完成大量的计算任务。同时，GPU 还具有低功耗和低成本的优势，能够在相同的成本下提供更高的计算性能。

GPGPU 的应用范围非常广泛，包括计算流体力学、分子动力学模拟、天体物理学、人工智能、深度学习、图像处理等领域。随着 GPU 技术的不断发展，GPGPU 的应用也将越来越广泛。

GPGPU 计算框架

CUDA：由 NVIDIA 开发的 GPGPU 计算框架，支持 C、C++ 和 Fortran 编程语言。

OpenCL：由 Khronos Group 开发的开放式跨平台 GPGPU 框架，支持多种编程语言，包括 C、C++、Python 和 Java 等。

HIP：由 AMD 开发的 GPGPU 框架，支持 C++ 编程语言，可与 CUDA 代码兼容。

SYCL：由 Khronos Group 开发的基于 C++ 的 GPGPU 框架，支持异构计算，可在 CPU 和 GPU 之间自动分配任务。

ROCm：由 AMD 开发的开源 GPGPU 计算框架，支持多种编程语言，包括 C、C++、Python 和 Julia 等。

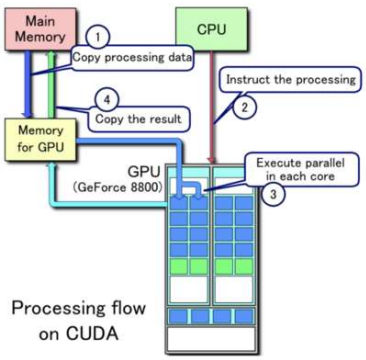
TensorFlow：由 Google 开发的人工智能框架，支持 GPGPU 计算，可在多种硬件平台上运行。

PyTorch：由 Facebook 开发的深度学习框架，支持 GPGPU 计算，可在多种硬件平台上运行。

🔥 面对目前 GPU 软硬件的限制，我们能做些什么?

CUDA（Compute Unified Device Architecture）

- **CUDA 处理流程示例**
  1. 将数据从主内存复制到 GPU 内存
  2. CPU 启动 GPU 计算内核
  3. GPU 的 CUDA 内核并行执行内核
  4. 将生成的数据从 GPU 内存复制到主内存

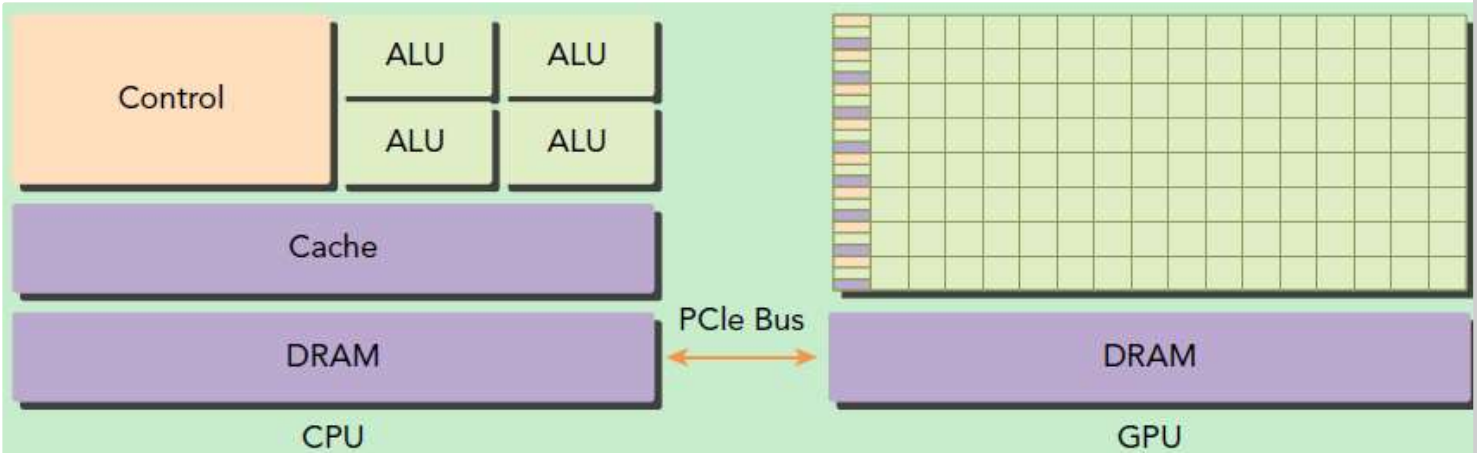


GPU 硬件

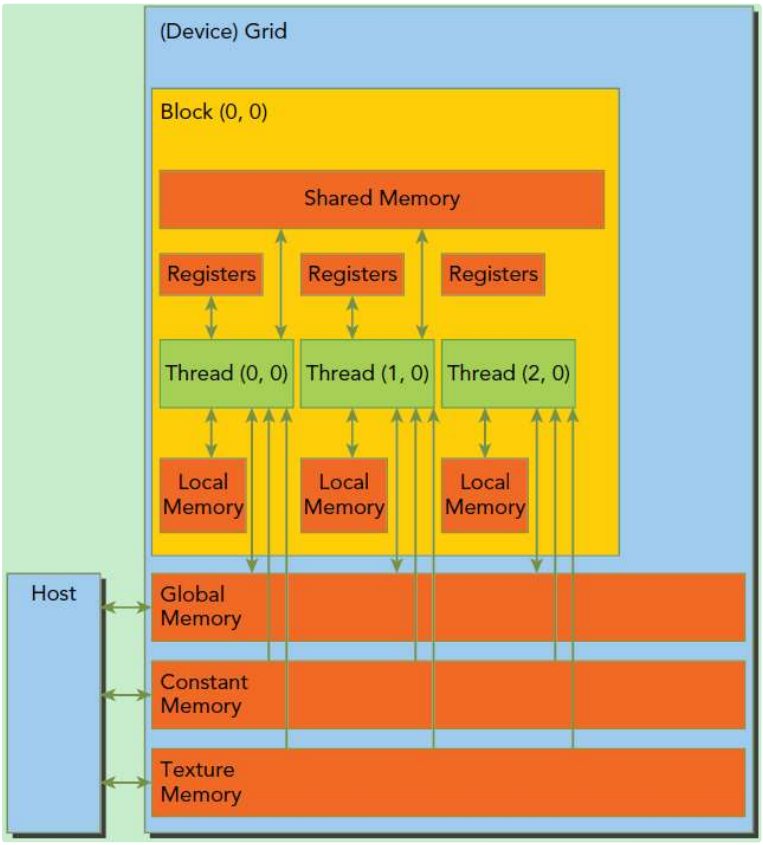
	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.5 TFLOPS
Single-Precision Performance	14 TFLOPS	15 TFLOPS
Tensor Performance	112 TFLOPS	120 TFLOPS
GPU Memory	16 GB HBM2	
Memory Bandwidth	900 GB/sec	
ECC	Yes	
Interconnect Bandwidth*	32 GB/sec	300 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

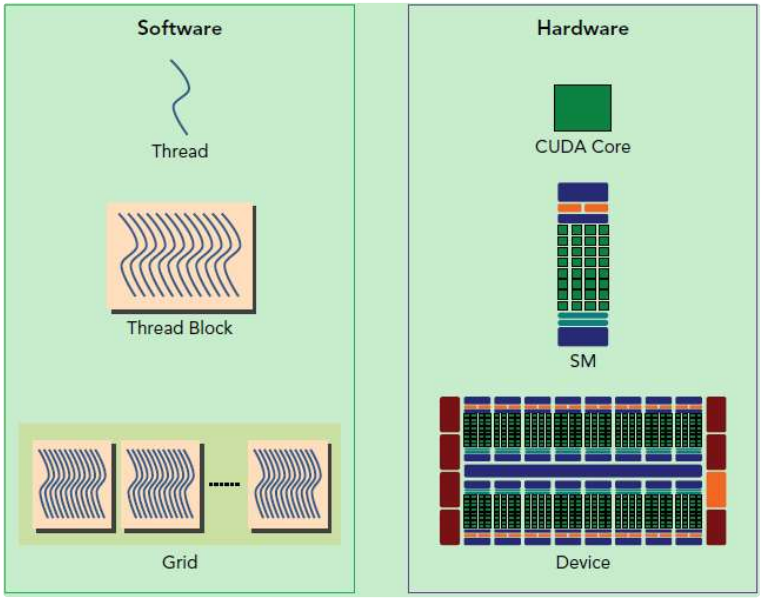
GPU 编程

CUDA 编程入门极简教程



GPU 并不是一个独立运行的计算平台，而需要与 CPU 协同工作，可以看成是 CPU 的协处理器，因此当我们在说 GPU 并行计算时，其实是指的基于 CPU+GPU 的异构计算架构。在异构计算架构中，GPU 与 CPU 通过 PCIe 总线连接在一起来协同工作，CPU 所在位置称为为主机端（host），而 GPU 所在位置称为设备端（device）。



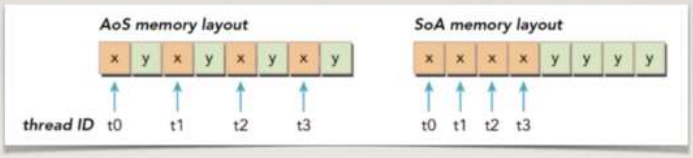


分波分析程序的第一次尝试

满带宽分波分析程序包 FALLS（Full bandwidth amplitude analysis software）

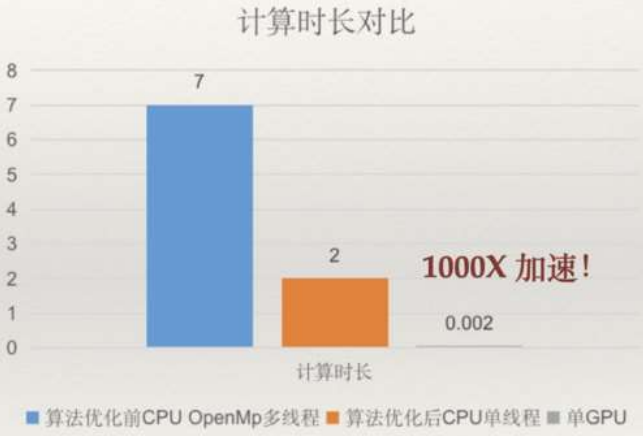
优化示例：数据对齐

- 当一个内存事务的首个访问地址是缓存粒度（32或128字节）的偶数倍的时候称为对齐内存访问，非对齐的内存访问会造成带宽浪费。
- 当一个线程束（warp）内的线程访问的内存都在一个内存块里的时候，就会出现合并访问
  - 一次内存操作完成所有线程（thread）的读取请求。
- 在涉及到复数计算时，如果简单地将数据以复数形式（连续两个double变量）存储，会浪费一半内存带宽。

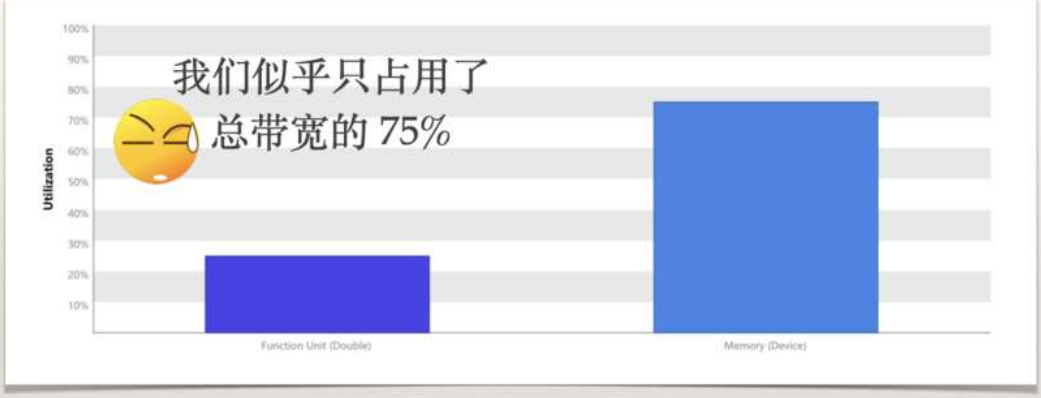


计算速度测试

- 例如处理1.6万事例，并用40万蒙特卡洛事例对参数空间进行积分，参数空间维度为190，使用单个Tesla V100，迭代步数50万步，程序总时长为50分钟。
- FALLS 可以处理更大的数据量和更大的参数空间。



FALLS 真的满带宽吗?



- 利用 nvprof 软件可以得到 NVIDIA GPU 完整运行报告。
- Tesla V100-SXM2-16GB 的理论内存带宽是 900GB/s

Tesla V100的“最佳”带宽

Table 3.1: Geometry, properties and latency of the memory hierarchy on the Volta, Pascal, Maxwell and Kepler architectures. All data in this table are measured on PCI-E cards.

		Volta V100 GV100	Pascal P100 GP100	Pascal P4 GP104	Maxwell M60 GM204	Kepler K80 GK210
Global memory	Memory bus	HBM2	HBM2	GDDR5	GDDR5	GDDR5
	Size	16,152 MiB	16,276 MiB	8,115 MiB	8,155 MiB	12,237 MiB
	Max clock rate (f_m)	877 MHz	715 MHz	3,003 MHz	2,505 MHz	2,505 MHz
	Theoretical bandwidth	900 GiB/s	732 GiB/s	192 GiB/s	160 GiB/s	240 GiB/s
	Measured bandwidth	750 GiB/s	510 GiB/s	162 GiB/s	127 GiB/s	191 GiB/s
	Measured/Theoretical Ratio	83.3%	69.6%	84.4%	79.3%	77.5%

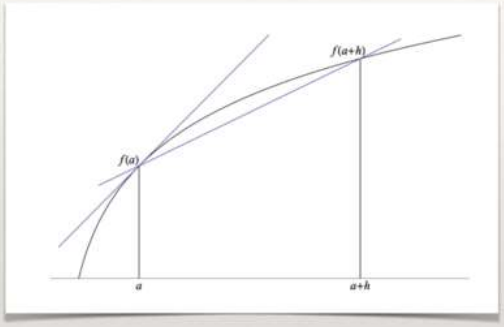
Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking, Zhe Jia, Marco Maggioni, Benjamin Staiger, Daniele P. Scarpazza, arXiv:1804.06826

FALLS 在 Tesla V100 上的确无愧于“满带宽”！

“天下武功，唯快不破？”

分波分析程序的第二次尝试

差分求导



用差分法计算导数

$$f(x) = \sin(x) \quad f'(x) = \cos(x)$$

$$f'(0.5) \approx 0.8775825619$$

$$f'(a) \approx \frac{f(a+h) - f(a)}{h}$$

h	$\big(f(a+h) - f(a)\big)/h$	$E(f;a,h)$
10^{-1}	0.8521693479	2.5×10^{-2}
10^{-2}	0.8751708279	2.4×10^{-3}
10^{-3}	0.8773427029	2.4×10^{-4}
10^{-4}	0.8775585892	2.4×10^{-5}
10^{-5}	0.8775801647	2.4×10^{-6}
10^{-6}	0.8775823222	2.4×10^{-7}
10^{-7}	0. 8775825372	2.5×10^{-8}
10^{-8}	0.8775825622	-2.9×10^{-10}
10^{-9}	0.8775825622	-2.9×10^{-10}
10^{-11}	0.8775813409	1.2×10^{-6}
10^{-14}	0.8770761895	5.1×10^{-4}
10^{-15}	0.8881784197	-1.1×10^{-2}
10^{-16}	1.110223025	-2.3×10^{-1}
10^{-17}	0.000000000	8.8×10^{-1}

- 数值差分计算导数的精度无法满足高维优化的需要。

拟牛顿法

Quasi Newton Method

- $\Delta x_k = -\alpha B_k^{-1} \nabla f(x_k)$
- $x_{k+1} = x_k + \Delta x_k$
- 计算新一个迭代点下的梯度$\nabla f(x_{k+1})$
- 令$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
- 利用y_k，直接近似Hessian矩阵的逆矩阵B_{k+1}^{-1}. 近似的方法如下表:

$$x_{k+1} = x_k - B^{-1} \nabla f\left(x_k\right)$$

Method	$B_{k+1} =$	$H_{k+1} = B_{k+1}^{-1} =$
DFP法	$\left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right) B_k \left(I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k}\right) + \frac{y_k y_k^T}{y_k^T \Delta x_k}$	$H_k + \frac{\Delta x_k \Delta x_k^T}{y_k^T \Delta x_k} - \frac{H_k y_k y_k^T H_k^T}{y_k^T H_k y_k}$
BFGS法	$B_k + \frac{y_k y_k^T}{y_k^T \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^T}{\Delta x_k^T B_k \Delta x_k}$	$\left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right)^T H_k \left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^T}{y_k^T \Delta x_k}$
Broyden法	$B_k + \frac{y_k - B_k \Delta x_k}{\Delta x_k^T \Delta x_k} \Delta x_k^T$	$H_k + \frac{(\Delta x_k - H_k y_k) \Delta x_k^T H_k}{\Delta x_k^T H_k y_k}$
Broyden族	$(1-\varphi_k) B_{k+1}^{BFGS} + \varphi_k B_{k+1}^{DFP}, \quad \varphi \in [0,1]$	
SR1法	$B_k + \frac{(y_k - B_k \Delta x_k)(y_k - B_k \Delta x_k)^T}{(y_k - B_k \Delta x_k)^T \Delta x_k}$	$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^T}{(\Delta x_k - H_k y_k)^T y_k}$

- Δx_k 的大小通过Wolfe条件确定
- y_k 对Hessian矩阵及其逆矩阵的计算起决定性作用
- 拟牛顿法需要准确的导数方向进行移动
- BFGS方法需要准确的导数保证 B_{k+1} 和 H_{k+1} 矩阵的正定性

数值差分对BFGS方法正定性的影响

Broyden–Fletcher–Goldfarb–Shanno algorithm

- 差分计算一阶导数的计算精度低于10个有效数字，而梯度下降过程中，两临近点的一阶导数差别往往在10个有效数字之后，这导致 y_k 的精度无法得到有效保证

h	(f'(a+h)-f'(a))/h	Error
10^(- 0)	-0.8068453682952281	6.82942e-01
10^(- 1)	-0.522469845165574	8.97831e-02
10^(- 2)	-0.4838129896711507	9.15148e-03
10^(- 3)	-0.4798939023942239	9.76927e-04
10^(- 4)	-0.4796163466380676	3.97993e-04
10^(- 5)	-0.48294701571194304	7.34520e-03
10^(- 6)	-0.49960036108132044	4.20812e-02
10^(- 7)	-0.5551115123125783	1.57868e-01
10^(- 8)	0.0	-1.00000e+00
10^(- 9)	-55.51115123125783	1.14787e+02
10^(-10)	0.0	-1.00000e+00
10^(-11)	0.0	-1.00000e+00
10^(-12)	-55511.15123125783	1.15786e+05
10^(-13)	0.0	-1.00000e+00
10^(-14)	0.0	-1.00000e+00
10^(-15)	-55511151.231257826	1.15787e+08
10^(-16)	-555111512.3125783	1.15787e+09

As H_k is spd, the form $(x, y) \mapsto \langle x, y \rangle_k := x^t H_k y$ defines a scalar product. By Cauchy-Schwarz we have

$$\langle x, y \rangle_k^2 \leq \langle x, x \rangle_k \langle y, y \rangle_k$$

for any $x, y \in \mathbb{R}^n$ and with strict inequality if x and y aren't linear dependent. Now let $x \in \mathbb{R}^n \setminus \{0\}$, then

$$\begin{aligned}
\langle x, x \rangle_{k+1} &= x^t H_{k+1} x \\
&= x^t H_k x + x^t \frac{\eta \eta^t}{\delta^t \eta} x - x^t \frac{H_k \delta \delta^t H_k^t}{\delta^t H_k \delta} x \\
&= \langle x, x \rangle_k + \frac{1}{\delta^t \eta} x^t \eta \eta^t x - \frac{1}{\langle \delta, \delta \rangle_k} \langle x, \delta \rangle_k \langle \delta, x \rangle_k \\
&= \langle x, x \rangle_k + \frac{1}{\delta^t \eta} (x^t \eta)^2 - \frac{1}{\langle \delta, \delta \rangle_k} \langle x, \delta \rangle_k^2
\end{aligned}$$

If now x and δ aren't linear dependent, then we can estimate further using Cauchy-Schwarz:

$$\begin{aligned}
\langle x, x \rangle_{k+1} &> \langle x, x \rangle_k + 0 - \frac{1}{\langle \delta, \delta \rangle_k} \langle x, x \rangle_k \langle \delta, \delta \rangle_k \\
&= 0.
\end{aligned}$$

Otherwise, if, say $x = \lambda \delta$, then $x^t \eta = \lambda \delta^t \eta \neq 0$, and, hence

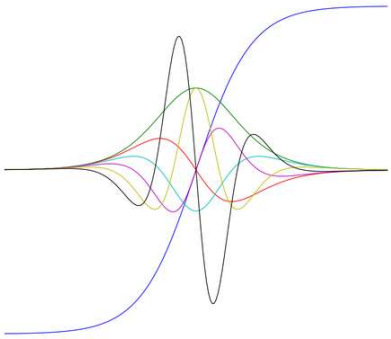
$$\begin{aligned}
\langle x, x \rangle_{k+1} &= \langle x, x \rangle_k + \frac{1}{\delta^t \eta} (x^t \eta)^2 - \frac{1}{\langle \delta, \delta \rangle_k} \langle x, x \rangle_k \langle \delta, \delta \rangle_k \\
&= \frac{1}{\delta^t \eta} (x^t \eta)^2 \\
&> 0.
\end{aligned}$$

So $\langle x, x \rangle_{k+1} > 0$ for $x \neq 0$, and H_{k+1} is positive definite.

Autograd

GitHub - HIPS/autograd: Efficiently computes derivatives of numpy code.

```
>>> from autograd import elementwise_grad as egrad  # for functions that vectorize over inputs
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-7, 7, 200)
>>> plt.plot(x, tanh(x),
...          x, egrad(tanh)(x),                                 # first  derivative
...          x, egrad(egrad(tanh))(x),                          # second derivative
...          x, egrad(egrad(egrad(tanh)))(x),                   # third  derivative
...          x, egrad(egrad(egrad(egrad(tanh))))(x),            # fourth derivative
...          x, egrad(egrad(egrad(egrad(egrad(tanh)))))(x),     # fifth  derivative
...          x, egrad(egrad(egrad(egrad(egrad(egrad(tanh))))))(x))  # sixth  derivative
>>> plt.show()
```

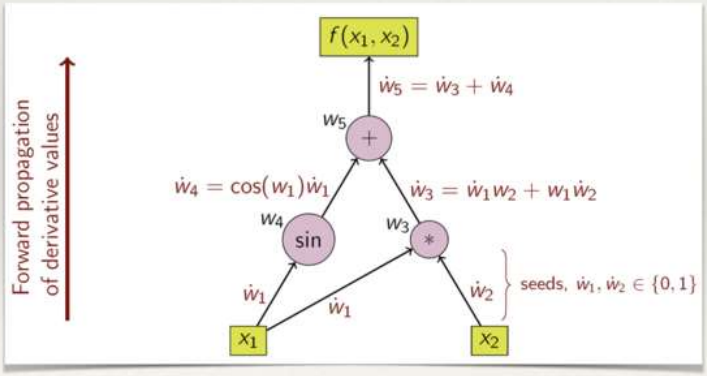


Note: Autograd is still being maintained but is no longer actively developed. The main developers (Dougal Maclaurin, David Duvenaud, Matt Johnson, and Jamie Townsend) are now working on JAX, with Dougal and Matt working on it full-time. JAX combines a new version of Autograd with extra features such as jit compilation.

向前自动求导

Forward Accumulation

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_{n-1}}\frac{\partial w_{n-1}}{\partial x} = \frac{\partial y}{\partial w_{n-1}}\left(\frac{\partial w_{n-1}}{\partial w_{n-2}}\frac{\partial w_{n-2}}{\partial x}\right) = \frac{\partial y}{\partial w_{n-1}}\left(\frac{\partial w_{n-1}}{\partial w_{n-2}}\left(\frac{\partial w_{n-2}}{\partial w_{n-3}}\frac{\partial w_{n-3}}{\partial x}\right)\right) = \cdots$$



$$\begin{aligned} z &= f(x_1, x_2) \\ &= x_1 x_2 + \sin x_1 \\ &= w_1 w_2 + \sin w_1 \\ &= w_3 + w_4 \\ &= w_5 \end{aligned}$$

$$\dot{w} = \frac{\partial w}{\partial x}$$

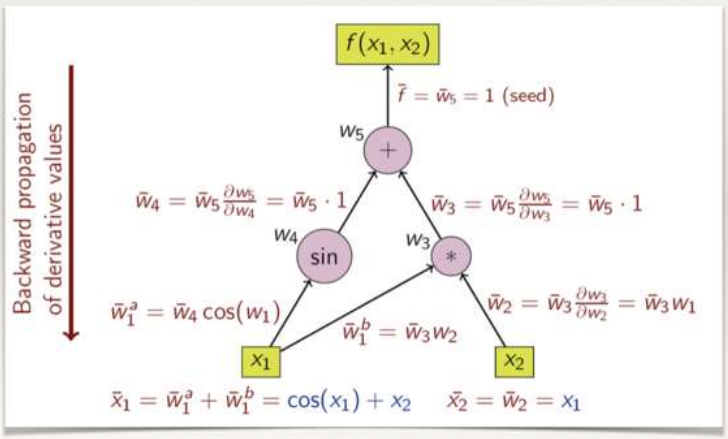
$$\dot{w}_1 = \frac{\partial x_1}{\partial x_1} = 1$$
$$\dot{w}_2 = \frac{\partial x_2}{\partial x_1} = 0$$

Operations to compute value	Operations to compute derivative
$w_1 = x_1$	$\dot{w}_1 = 1$ (seed)
$w_2 = x_2$	$\dot{w}_2 = 0$ (seed)
$w_3 = w_1 \cdot w_2$	$\dot{w}_3 = w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2$
$w_4 = \sin w_1$	$\dot{w}_4 = \cos w_1 \cdot \dot{w}_1$
$w_5 = w_3 + w_4$	$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$

向后自动求导

Reverse Accumulation

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$



$$\begin{aligned} z &= f(x_1, x_2) \\ &= x_1 x_2 + \sin x_1 \\ &= w_1 w_2 + \sin w_1 \\ &= w_3 + w_4 \\ &= w_5 \end{aligned}$$

$$\bar{w} = \frac{\partial y}{\partial w}$$

Operations to compute derivative
$\bar{w}_5 = 1$ (seed)
$\bar{w}_4 = \bar{w}_5$
$\bar{w}_3 = \bar{w}_5$
$\bar{w}_2 = \bar{w}_3 \cdot w_1$
$\bar{w}_1 = \bar{w}_3 \cdot w_2 + \bar{w}_4 \cdot \cos w_1$

- 在只有一个输出值的情况下，向后自动求导遍历一遍计算图，可以得到所有的导数。
- 向前自动求导的计算顺序与函数本身的计算顺序是一致的。
- 深度学习多使用向后自动求导，我们的分波分析程序应该使用哪一个？

自动微分与深度学习

Auto Gradient and Deep Learning

- 目前主流的深度学习框架都已实现自动求导（TensorFlow, PyTorch…）
- 为了适合神经网络构造的复杂性和多样性，深度学习框架使用的自动求导功能强大，针对深度学习有专门优化。
- 针对特定计算领域的自动微分优化算法。

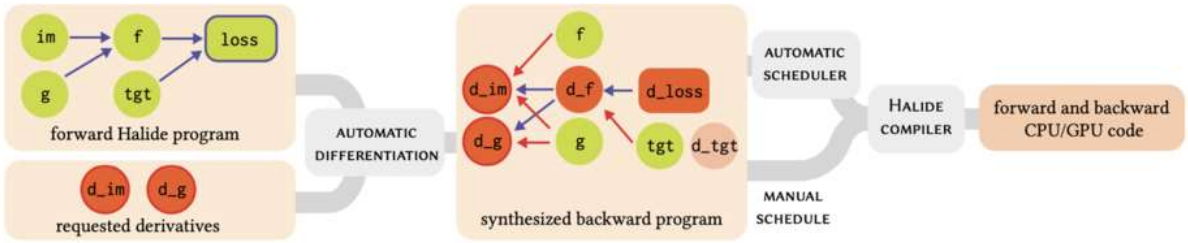


Fig. 3. Overview of our compiler. The user writes a forward Halide program as they would normally. Then, they specify the set of outputs and gradients the system should produce. Our automatic differentiation generates new Halide functions that implement the requested gradients. The user can either manually schedule the pipeline or use our automatic scheduler. Finally, the Halide compiler generates machine code for the scheduled forward and backward algorithms.

Differentiable Programming for Image Processing and Deep Learning in Halide, Tzu-Mao Li, etc., ACM Trans. Graph., Vol. 37, No. 4, Article 139. Publication date: August 2018.

自动微分的实现

Technique	Advantage(s)	Drawback(s)
Hand-coded analytical derivative	Exact and often fastest method.	Time consuming to code, error prone, and not applicable to problems with implicit solutions. Not automated.
Finite differentiation	Easy to code.	Subject to floating point precision errors and slow, especially in high dimensions, as the method requires at least D evaluations, where D is the number of partial derivatives required.
Symbolic differentiation	Exact, generates symbolic expressions.	Memory intensive and slow. Cannot handle statements such as unbounded loops.
Automatic differentation	Exact, speed is comparable to hand-coding derivatives, highly applicable.	Needs to be carefully implemented, alhough this is already done in several packages.

arXiv:1811.05031v2 [cs.MS] 14 Jan 2019

autodiff.org

Automatic Differentiation (AD) is a set of techniques based on the mechanical application of the chain rule to obtain derivatives of a function given as a computer program. AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions or elementary functions such as exp(). By applying the chain rule of derivative calculus repeatedly to these operations, derivatives of arbitrary order can be computed automatically, and accurate to working precision.

代码生成与FALLS的自动微分

变长参数

变参数函数是能以不同数目参数调用的函数。

只有新式（原型）函数声明可以为变长参数的。它必须通过出现在参数列表最后的 `...` 形式的参数指定，且至少跟随一个具名参数。

```
//新式声明
int printx(const char* fmt, ...); // 此方法声明的函数
printx("hello world"); // 可能会以一个
printx("a=%d b=%d", a, b); // 或更多参数调用

// int printy(..., const char* fmt); // 错误:  ... 必须在最后
// int printz(...); // 错误:  ... 必须跟随至少一个具名参数
```

在函数调用中，每个属于变长参数列表一部分的参数会经历名为默认参数提升的隐式转换。

在函数体内使用变长参数时，这些参数的值必须用 <stdarg.h> 库工具访问：

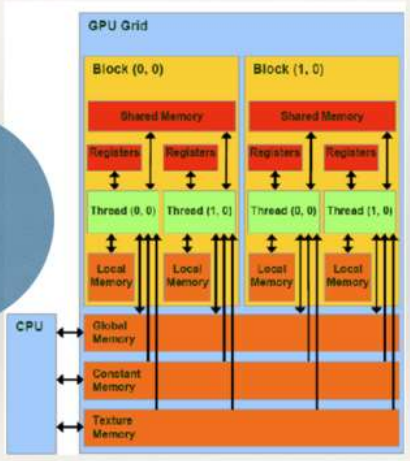
定义于头文件 <stdarg.h>

va_start	令函数得以访问可变参数 (宏函数)
va_arg	访问下一个函数可变参数 (宏函数)
va_copy (C99)	创造函数可变参数的副本 (宏函数)
va_end	结束函数可变参数的行程 (宏函数)
va_list	保有 va_start 、 va_arg 、 va_end 及 va_copy 所需信息 (typedef)

- 依靠C++变长参数和类型推断功能实现自动微分运算的组装
- 不同的运算可以写成同样的形式，只是传入的模板不同
- 可以对同一运算设定不同的类型，也就是不同的运算策略和缓存值，进而提高运算效率

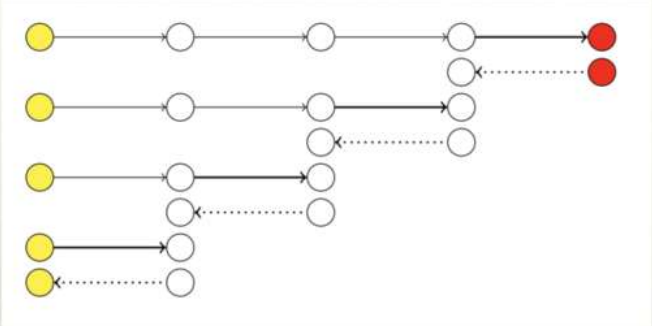
自动微分的优化

Method	Benefits	Limitations	Packages
Source transformation	Can be optimized by hand or with a compiler.	Can only differentiate functions which are fully defined at compile time.	`AdiFor`, `TAC++` `Tapenade`
Operator Overloading	Can handle most programing statements.	Memory handling requires some care.	All the packages listed below.
Retaping	Fast when we differentiate through the same expression graph multiple times.	Applicable only if the expression graph is conserved from point to point. When the graph changes a new tape can be created.	`Adol-C`, `CppAd`
Checkpointing	Reduces peak memory usage.	Longer runtime. An optimal placement is binary partition, but this scheme does not always apply.	`Adol-C`, `CppAD` (`Tapenade` for source transformation).
Region-based memory	Improves speed and memory.	Makes the implementation less transparent. Some care is required for nested AD.	`Adept`, `Sacado`, `Stan Math`
Expression templates	Removes certain redundant operations, whereby the code becomes faster and more memory efficient, and allows user to focus on code clarity.		`Adept`, `CoDiPack` `Sacado` (with some work).

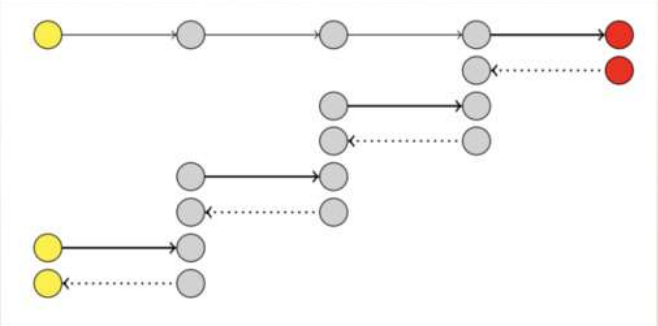


arXiv:1811.05031v2 [cs.MS] 14 Jan 2019

优化措施: checkpoints



recompute-all approach



store-all approach

Fortunately, recompute-all and store-all constitute two ends of a spectrum: in between methods can be used by selecting a subset of the checkpoints where the values are stored. Such checkpoints constitute snapshots, and give us more control on peak memory usage.

arXiv:1811.05031v2 [cs.MS] 14 Jan 2019

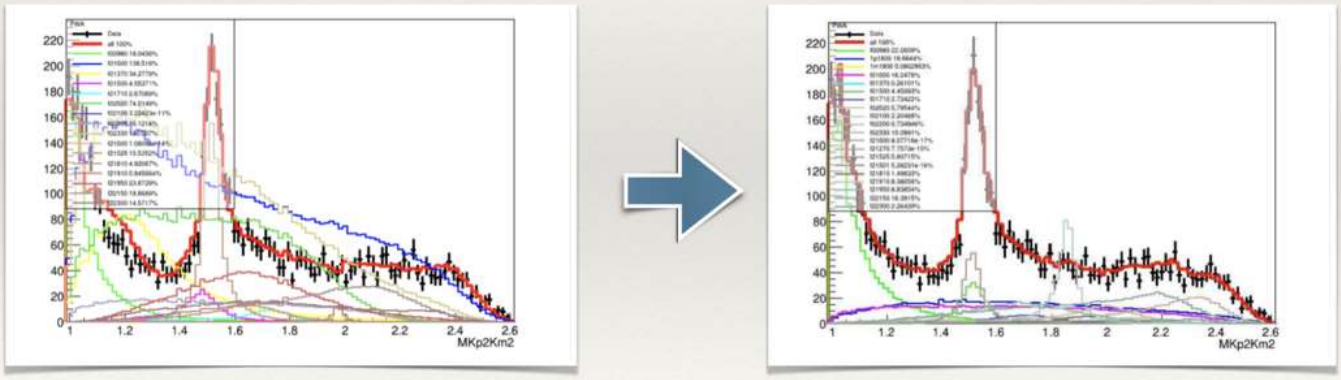
优化措施： GPU寄存器使用优化

- 一个GPU设备上的所有SM中活跃的线程数目通常数以万计。
- GPU的每个SM（流多处理器）有上千个寄存器GPU
- 利用多线程隐藏了内存获取与指令执行带来的延迟。GPU不使用寄存器重命名机制，而是致力于为每一个线程都分配一个真实的寄存器（毕竟每个SM有上千个）。因此，当需要上下文切换时，所需要的操作就是将指向当前寄存器组的选择器或指针更新，以指向下一个执行的线程束的寄存器组，开销几乎为零。
- 编译时会计算出每个内核线程需要的寄存器数目。所有的线程块都具有相同的大小，并拥有已知的数目的线程，每个线程块需要的寄存器数目也就是已知的和固定的。如果一个内核函数中的每个线程需要的寄存器过多，在每个SM中GPU能调度的线程块的数量就会受到限制，因此总的执行的线程数量也会受到限制。此时，开启的线程数量过少会造成硬件无法被充分利用，性能下降。

LASSO

Least Absolute Shrinkage and Selection Operator

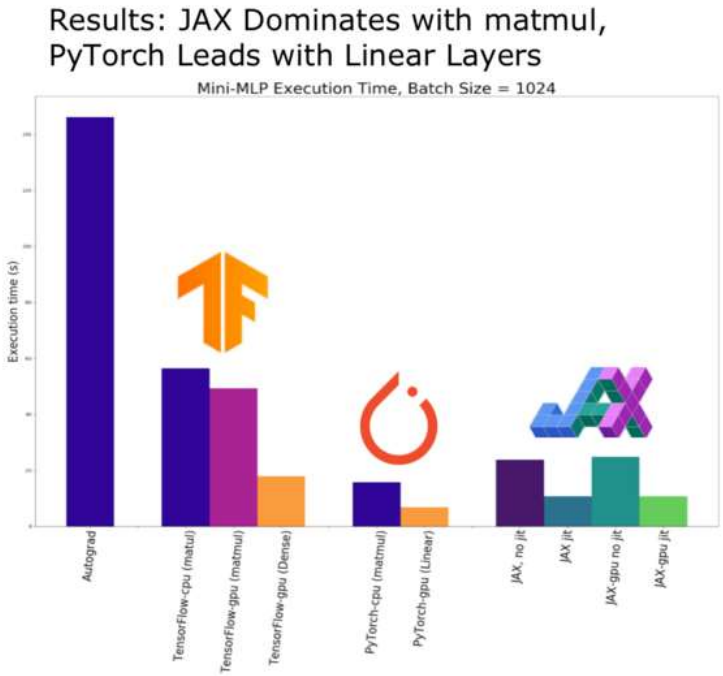
- Lasso是一种同时进行特征选择和正则化（数学）的回归分析方法，旨在增强统计模型的预测准确性和可解释性。
  Regression shrinkage and selection via the Lasso, Robert Tibshirani, J. R. Statist. Soc. B(1996) 58, No. 1, pp.267-288
- 通过引入LASSO惩罚项，我们可以压缩无关参数的大小，选择出我们需要的中间态，抑制过拟合现象。



分波分析程序的第三次尝试

自动微分

- 分波分析使用最大似然法拟合数据，而在拟合过程中通常使用梯度下降类型的方法进行优化；
- 梯度的计算精度对拟合性能有决定性影响；
- 基于GPU的自动微分算法在精度和性能两方面相对于数值微分和解析微分都有明显优势：
  - TensorFlow
  - PyTorch
  - JAX



GitHub - google/jax: Composable transformations of Python+NumPy programs: differentiate, vectorize, JIT to GPU/TPU, and more

Automatic differentiation with `grad`

JAX has roughly the same API as Autograd. The most popular function is `grad` for reverse-mode gradients:

```
from jax import grad
import jax.numpy as jnp

def tanh(x):  # Define a function
  y = jnp.exp(-2.0 * x)
  return (1.0 - y) / (1.0 + y)

grad_tanh = grad(tanh)  # Obtain its gradient function
print(grad_tanh(1.0))   # Evaluate it at x = 1.0
# prints 0.4199743
```

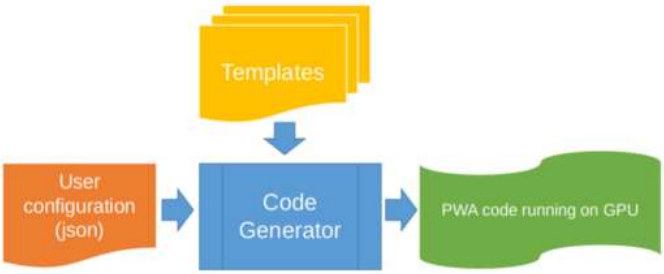
You can differentiate to any order with `grad`.

```
print(grad(grad(grad(tanh)))(1.0))
# prints 0.62162673
```

🔥 目前最简单高效调用 GPU 的方式!

代码生成

- JAX作为底层的自动微分工具，受框架限制更少；
- 代码生成技术的优点：
  - 性能优势；
  - 节约显存；
  - 多GPU支持；
  - 适应各种分波分析需要脚本。




```
Every 0.1s: nvidia-smi

Mon Jun 28 10:10:13 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.45.01    Driver Version: 455.45.01    CUDA Version: 11.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  GeForce RTX 3090    Off  | 00000000:1A:00.0 Off |                  N/A |
| 67%   68C    P2   304W / 350W |  23171MiB / 24268MiB |     98%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  GeForce RTX 3090    Off  | 00000000:68:00.0 Off |                  N/A |
| 68%   69C    P2   298W / 350W |  23169MiB / 24265MiB |     98%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A   2744169      C   python3                         23169MiB |
|    1   N/A  N/A   2744169      C   python3                         23167MiB |
+-----------------------------------------------------------------------------+
```

多GPU下可以让显卡达到满载的水平。

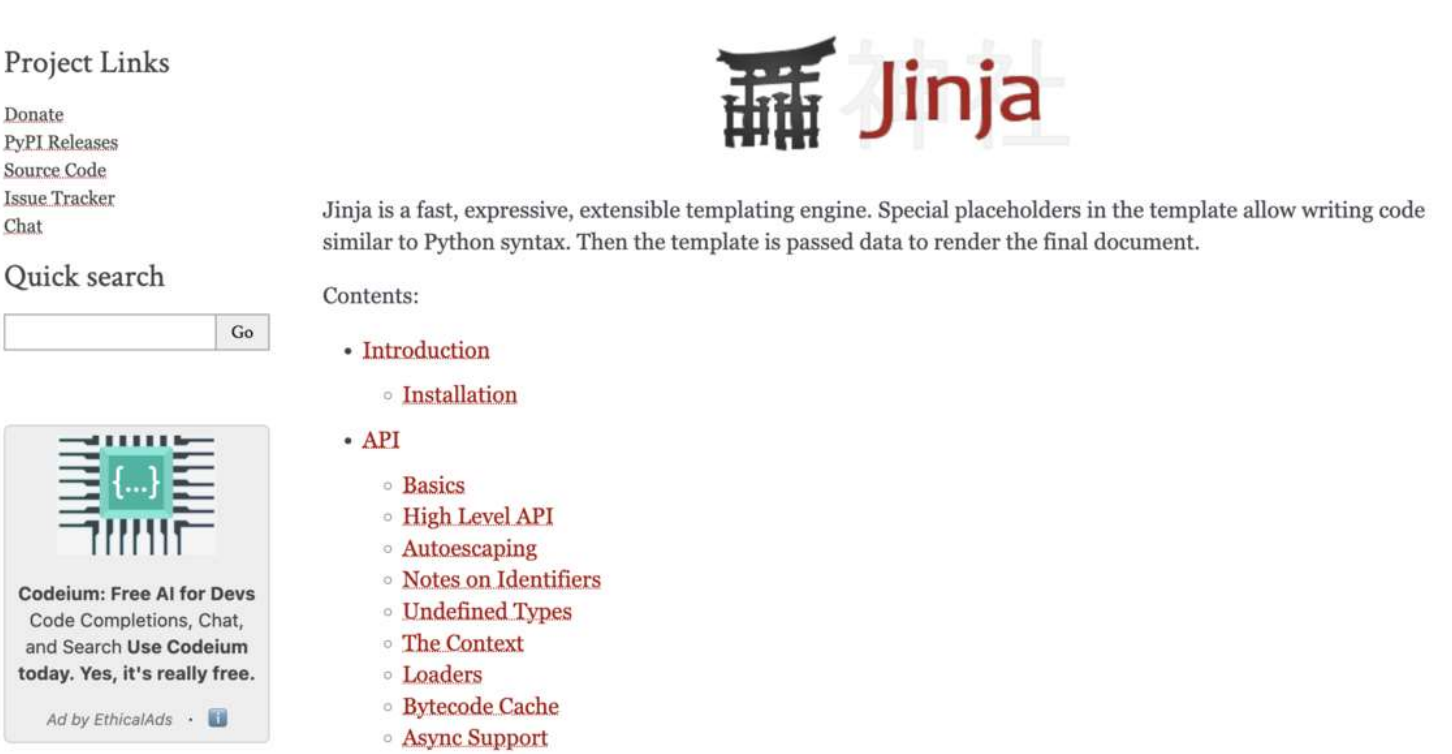
多GPU支持

- With the chain rule, we can construct a multi-GPU computing model:

$$\frac{\partial F}{\partial x_n} = \sum_m \frac{\partial G}{\partial y_m} \frac{\partial y_m}{\partial x_n}$$

$$\frac{\partial^2 F}{\partial x_i \partial x_j} = \sum_{mn} \frac{\partial^2 G}{\partial y_m \partial y_n} \frac{\partial y_n}{\partial x_j} \frac{\partial y_m}{\partial x_i} + \sum_m \frac{\partial G}{\partial y_m} \frac{\partial^2 y_m}{\partial x_i \partial x_j}$$

- The calculations related to y_m could be distributed on different computing equipments through `multithreading` or `multiprocessing`.
- They could run on multiple cards simultaneously to increase the computational efficiency.
- Or they could run on a single card one by one to break through the memory limit.



Project Links

Donate
PyPI Releases
Source Code
Issue Tracker
Chat

Quick search

Go

Jinja

Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document.

Contents:

Jinja — Jinja Documentation (3.1.x) (palletsprojects.com)

Jinja 允许使用方便的点表示法来访问 Python 词典中的数据。

`dicts.py`

```
#!/usr/bin/env python3

from jinja2 import Template

person = { 'name': 'Person', 'age': 34 }

tm = Template("My name is {{ per.name }} and I am {{ per.age }}")
# tm = Template("My name is {{ per['name'] }} and I am {{ per['age'] }}")
msg = tm.render(per=person)

print(msg)
```

我们有一本人词典。 我们使用点运算符访问字典键。

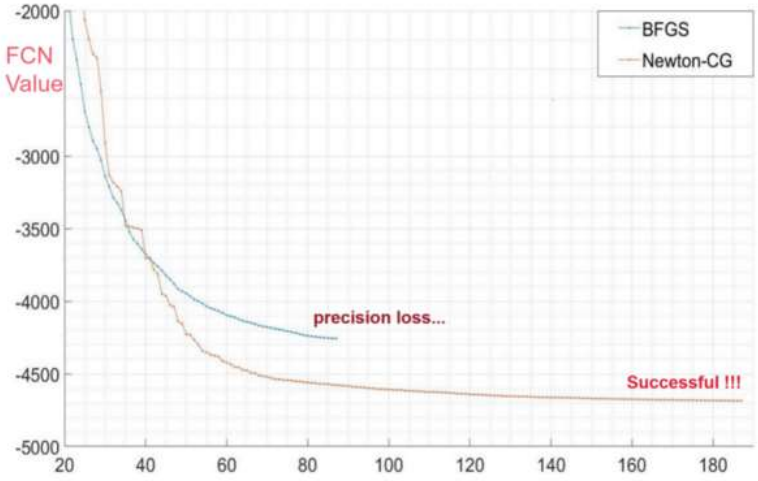
```
tm = Template("My name is {{ per.name }} and I am {{ per.age }}")
# tm = Template("My name is {{ per['name'] }} and I am {{ per['age'] }}")
```

活动方式和注释方式均有效。 点符号更方便。

优化方法与统计误差

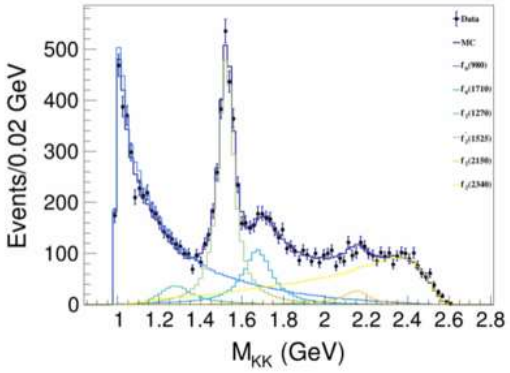
scipy.optimize.minimize

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)



精确的二阶导数矩阵（Hessian matrix）保证了极点的性质，同时提供了可靠的统计误差！

分波分析示例



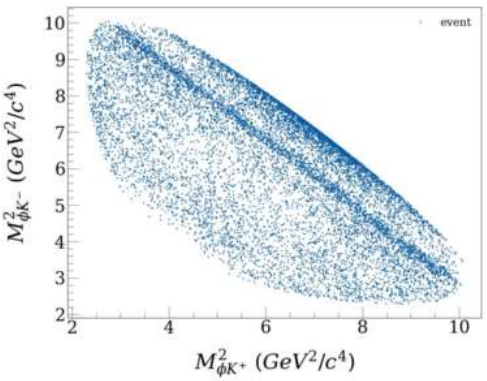


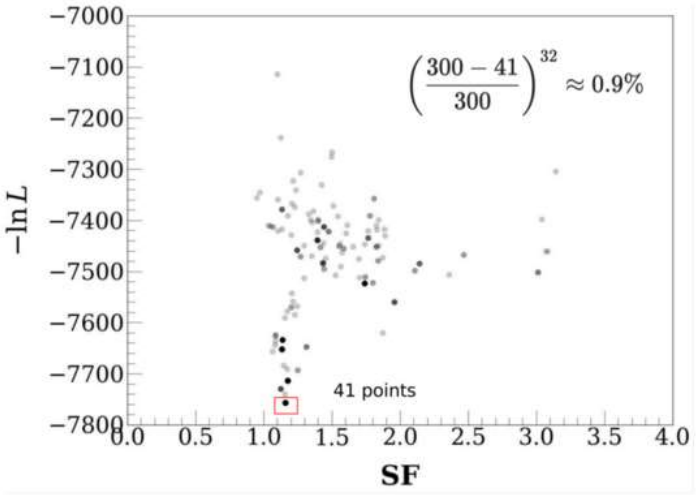
Table 1: The partial wave analysis model R_0 used to generate the data.

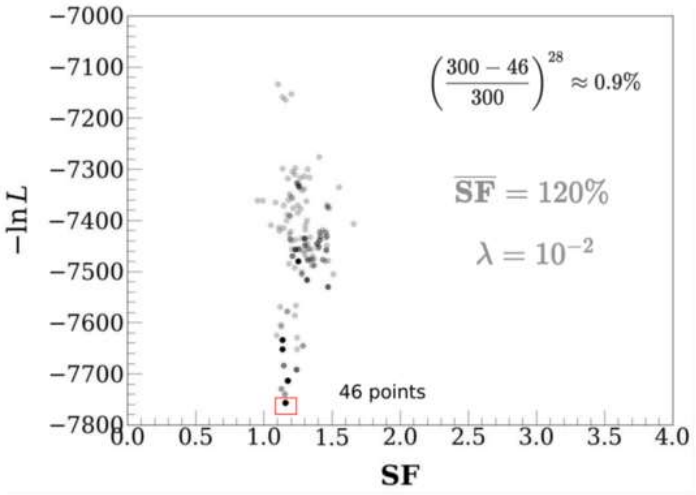
R_0	Name	F_i (%)	Mass(GeV)	width
1	$f_0(980)$	39.5	0.979	0.107
2	$f_2(2340)$	37.1	2.548	0.324
3	$f_2^{'}(1525)$	24.7	1.5223	0.0858
4	$f_0(1710)$	8.30	1.6762	0.1627
5	$f_2(1270)$	3.16	1.290	0.196
6	$f_2(2150)$	2.22	2.162	0.159
	SF	115.0		

$$\mathbf{F}_k = \sum_{i=1}^{N} \frac{|c_k M_k(\zeta_i)|^2}{|\sum_k c_k e^{i\phi_k} M_k(\zeta_i)|^2}$$

$$\mathbf{SF} = \sum_k \mathbf{F}_k$$

组分约束





$$-\ln L \quad \Longrightarrow \quad \tilde{L} = -\ln L + \lambda N(\mathbf{SF} - \overline{\mathbf{SF}})^2$$

组分约束对误差的影响

Table 2: Fitting results of the partial wave analysis model R_0 using $-\ln L$.

R_0	Name	F_i (%)	Mass(GeV)	Width(GeV)
1	$f_0(980)$	39.2 ± 1.5	1.015 ± 0.043	0.102 ± 0.030
2	$f_2(2340)$	37.5 ± 1.6	2.571 ± 0.015	0.281 ± 0.017
3	$f_2^{'}(1525)$	23.5 ± 1.0	1.5233 ± 0.0015	0.0841 ± 0.0031
4	$f_0(1710)$	8.66 ± 0.93	1.6714 ± 0.0047	0.159 ± 0.010
5	$f_2(1270)$	2.68 ± 0.57	1.288 ± 0.013	0.181 ± 0.027
6	$f_2(2150)$	2.52 ± 0.63	2.152 ± 0.012	0.170 ± 0.026
	SF	114.0		

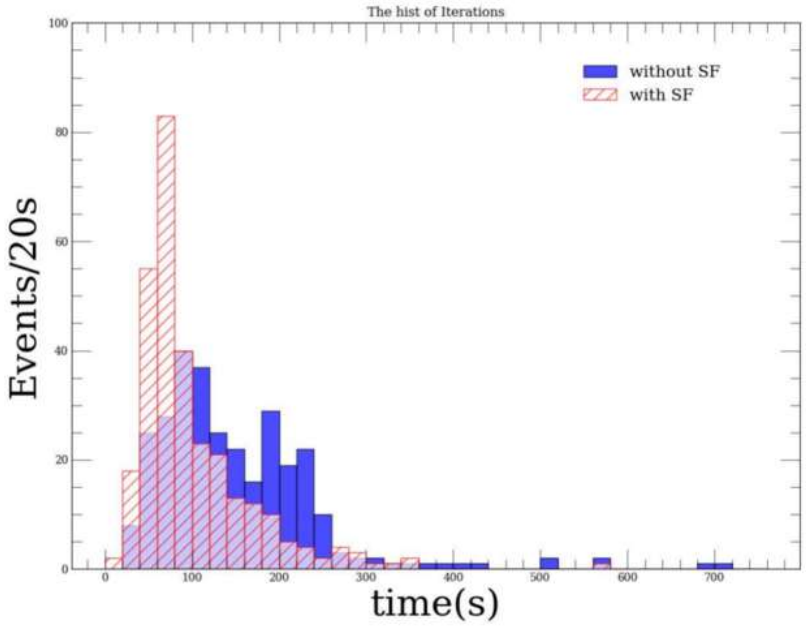
Table 3: Fitting results of the partial wave analysis model R_0 using $\tilde{L}$.

R_0	Name	F_i (%)	Mass(GeV)	Width(GeV)
1	$f_0(980)$	39.3 ± 1.6	1.017 ± 0.039	0.101 ± 0.035
2	$f_2(2340)$	37.5 ± 1.8	2.571 ± 0.016	0.282 ± 0.018
3	$f_2^{'}(1525)$	23.6 ± 1.0	1.5233 ± 0.0015	0.0842 ± 0.0031
4	$f_0(1710)$	8.72 ± 0.96	1.6712 ± 0.0047	0.159 ± 0.010
5	$f_2(1270)$	2.72 ± 0.58	1.288 ± 0.014	0.182 ± 0.026
6	$f_2(2150)$	2.52 ± 0.62	2.152 ± 0.012	0.170 ± 0.027
	SF	114.3		

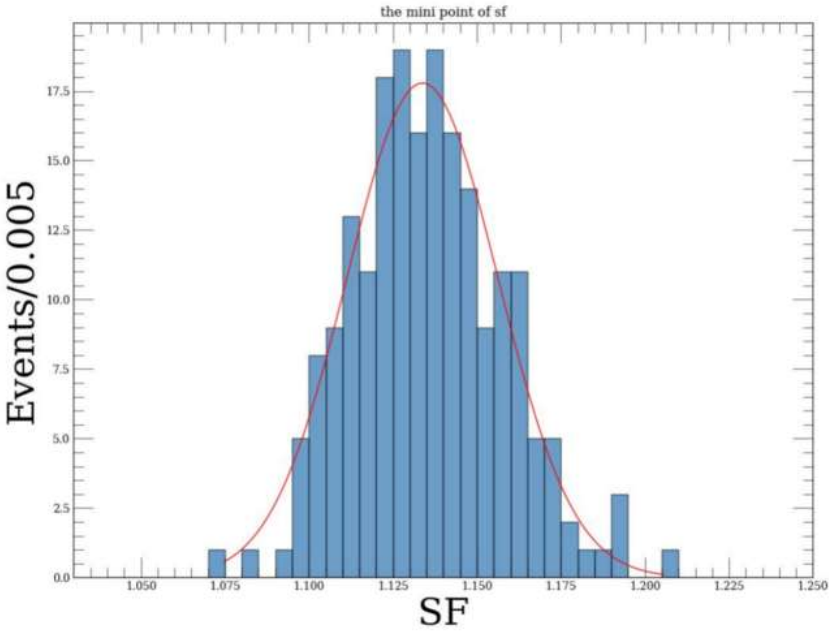
$\overline{\mathbf{SF}} = 120\%$ $\lambda = 10^{-2}$

组分约束对误差基本没有影响！

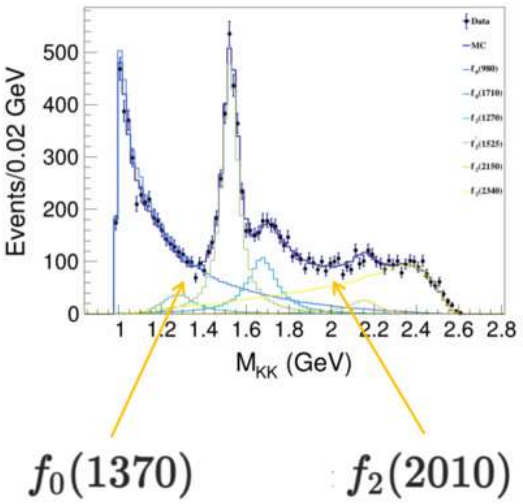
组分约束的性能

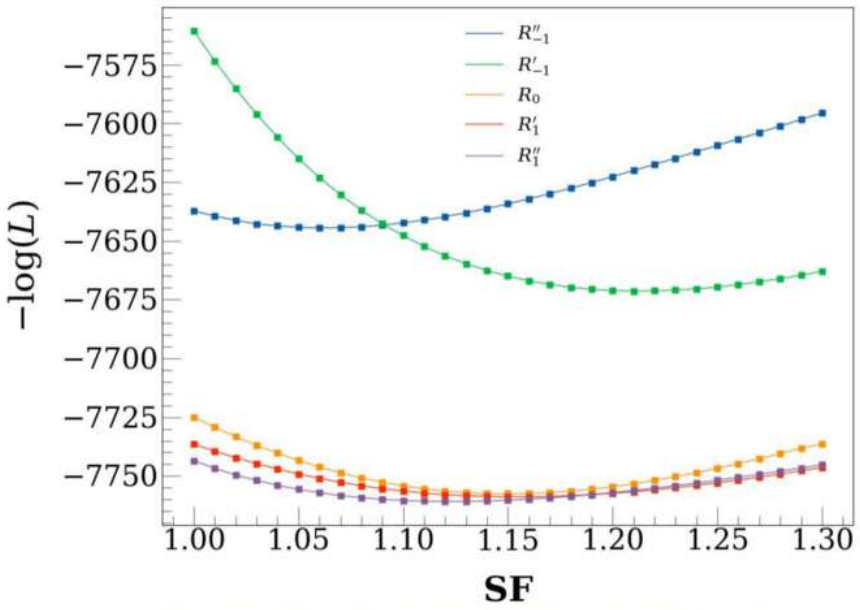


组分的涨落



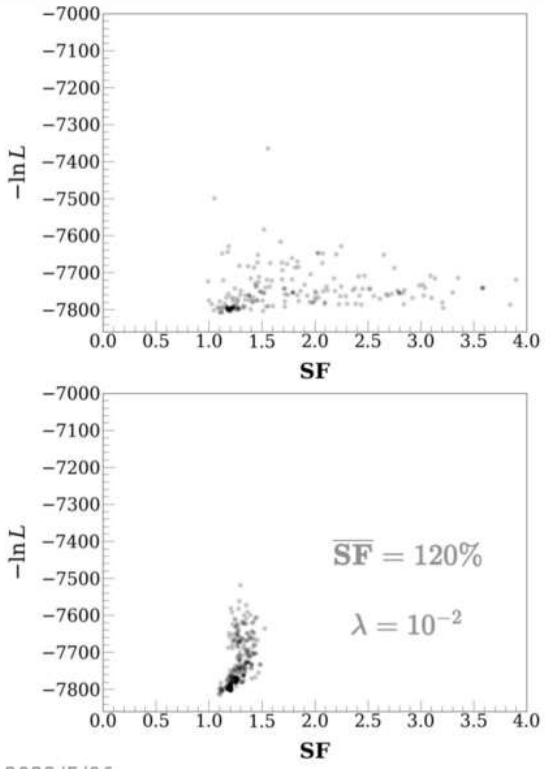
组分扫描





组分扫描的结果可以作为共振态选择的参考指标。

组分约束对多余共振态的抑制

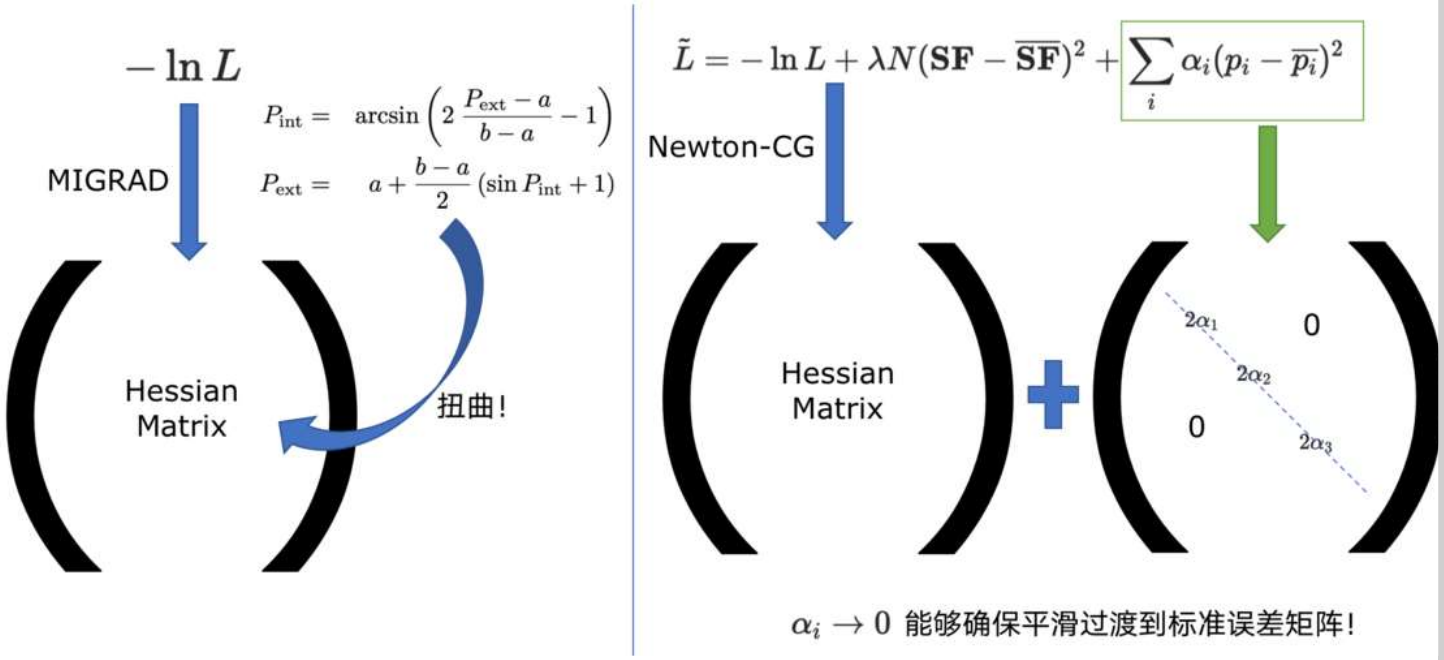


$$\lambda = 10^2\ ,\ \overline{SF} = 115.0\%$$

Table 4: Results of fitting using the model R_{+4} .

	Name	F_i (%)	Mass(GeV)	Width(GeV)
0	$f_2(2340)$	40.0	2.577	0.296
1	$f_0(980)$	34.3	1.007	0.094
2	$f_2^{'}(1525)$	24.8	1.524	0.088
3	$f_0(1710)$	9.88	1.666	0.162
4	$f_2(1270)$	2.92	1.279	0.184
5	$f_2(2150)$	2.23	2.140	0.134
6	$f_2(2010)$	0.37	2.011	0.202
7	$f_0(1500)$	0.20	1.507	0.112
8	$f_0(1370)$	0.15	1.347	0.200
9	$f_2(1640)$	0.08	1.639	0.099
	SF	115.0		

参数约束与误差



$\alpha_i \to 0$ 能够确保平滑过渡到标准误差矩阵!

AI 在分波分析中的使用

AI高精度蒙卡模拟研究现状

- 传统蒙卡模拟已被广范应用在高能物理事例产生当中，但在特定的物理过程及物理分析之中，蒙卡模拟由于其速度及计算资源限制，制约了相应的物理研究。部分高能物理学家考虑使用深度学习等技术来进行数据驱动式的快模拟
- 深度学习已在图像生成（如人脸，AI作画）以及文本生成（如AI自动对话，文章生成）领域已达到可实用的级别，计算机领域学者将图像，文本视为一种特定的高维分布问题，常规深度学习产生使用VAE和GAN进行数据驱动式的模型训练，在训练完成后则可高效完成所需目标。同时深度学习前沿考虑使用最优传输理论进行更为稳定的训练
- 高能物理学家已使用VAE，GAN在事例产生中作出一定成果，基本实现和目标高维分布大体趋势相同，但实际产生高维分布并不能达到同分布的实用要求，因此在做相关物理分析当中无法有效的代替蒙卡模拟


nature communications

Explore content ~ About the journal ~ Publish with us ~

nature › nature communications › articles › article

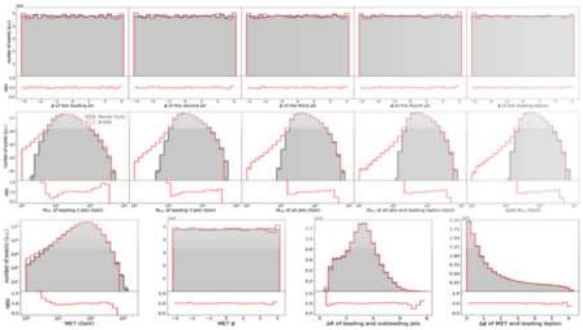
Article | Open Access | Published: 20 May 2021

Event generation and statistical sampling for physics with deep generative models and a density information buffer

Sydney Otten, Sascha Caron, Wieske de Swart, Melissa van Beekveld, Luc Hendriks, Caspar van Leeuwen, Damian Podareanu, Roberto Ruiz de Austri & Rob Verheyen

Nature Communications 12, Article number: 2985 (2021) | Cite this article

2003 Accesses | 12 Citations | Metrics



课题组早期工作