Security Level:

异构计算趋势和软件技术挑战

袁鹏
中软分布式与并行软件LAB
2016.11.10

www.huawei.com

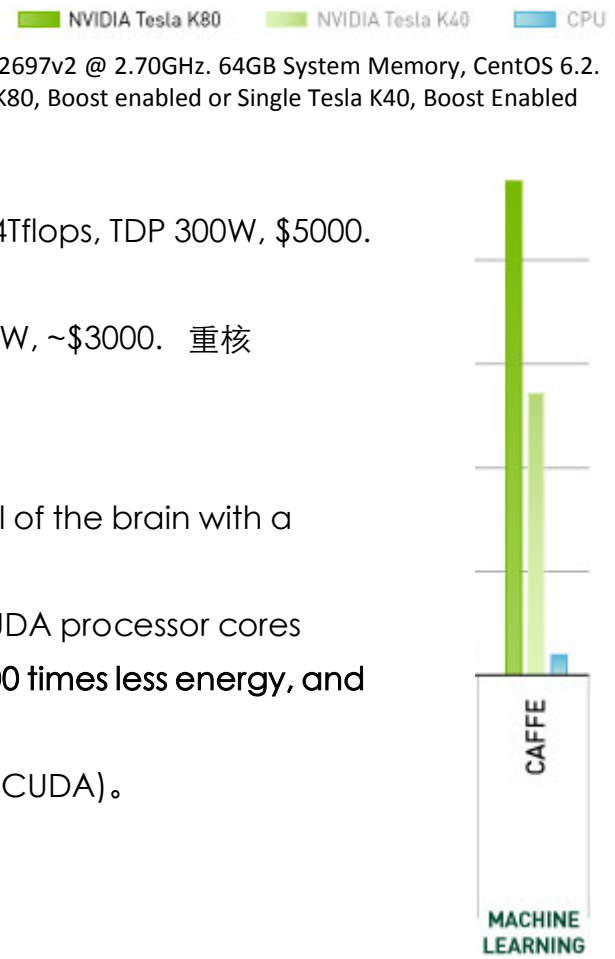
HUAWEI

目录

什么是异构计算

- Heterogeneous computing
  - refers to systems that use more than one kind of processor.
- 异构计算已经无处不在，从超算到桌面到云到终端。
  - **天河-2**：16,000个计算节点，每个节点 2*Xeon (IveBridge)+3*Phi。 Total 3,120,000 cores. Linpack : 33.86 petaFLOPS ， Power 17.6 megawatts。编程框架：OpenMC/OpeMP。
  - **Mac Pro**: Intel Xeon E5 (6/8/2 cores) + Dual AMD FirePro D500 GPU (1526 stream processors, 2.2 teraflops, 3-way 4k video)。编程框架：GCD、OpenCL、Metal。 $3,999
  - **Amazon Linux GPU Instances** g2.8xlarge: 4 GPU (each with 1,536 CUDA cores and 4 GB of video memory and the ability to 4* 1080p@30fps), 32 vCPU 。编程框架：CUDA，OpenCL。
  - **Qualcomm Snapdragon 820**： octa-core CPU+ Adreno 530 GPU+ Hexagon 680 DSP，编程框架：MARE，OpenCL。
- 异构计算不但是硬件平台，必须包含编程框架。

为什么要用异构计算

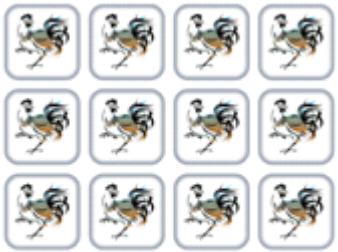


- 性能、性价比、功耗、面积。
  - **Nvidia Telsa K80**: 2* Kepler GK210, 4922 cores, DF 2.91 Tflops, SF 8.74Tflops, TDP 300W, $5000. 轻核
  - **Intel E5-2697V2 (Haswell)**：12 cores/24 threads, 600Gflops, TDP 130 W, ~$3000. 重核
- 你已经默默拥有了异构计算平台，不要浪费。
- 熟悉的例子
  - Google Brain：1,000 servers (16,000 CPU cores) simulating a model of the brain with a billion synapses 。
  - Nvidia：three GPU-accelerated servers: 12 GPUs in total, 18,432 CUDA processor cores (Google Brain has around 16,000 cores). The Nvidia solution uses **100 times less energy, and a 100 times less cost.**
  - Caffe是一个广泛应用的深度学习框架，其代码支持x86 (C++)和Nvidia (CUDA)。
- 但仍然心存困惑：哪些场景用，软件怎么用（语言、编程模型）。
  - Scale-out vs. Scale-up，足够的数据级并行，计算密集?

选谁？CPU GPU各有所长：Latency vs. Throughpu

Specifications	Sandy Bridge-EP	Kepler (Tesla K20)
Processing Elements	8 cores, 2 issue, 8 way SIMD @3.1 GHz	14 SMs, 6 issue, 32 way SIMD @730 MHz
Resident Strands/ Threads (max)	8 cores, 2 threads, 8 way SIMD: 96 strands	14 SMs, 64 SIMD vectors, 32 way SIMD: 28672 threads
SP GFLOP/s	396	3924
Memory Bandwidth	51 GB/s	250 GB/s
Register File	128 kB (?)	3.5 MB
Local Store/L1 Cache	256 kB	896 kB
L2 Cache	2 MB	1.5 MB
L3 Cache	20 MB	-

Latency

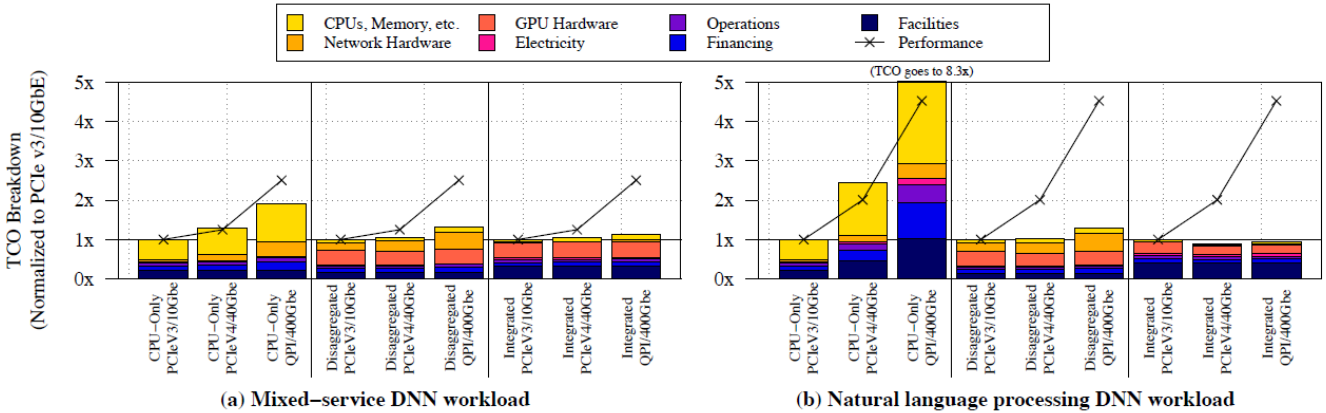


Throughput

还有：
FPGA
TPU
DSP
等等

异构计算是个系统工程，当前处于定制阶段
DjiNN and Tonic ：给出数据中心设计

- 动机： Apple Siri这样的应用需要什么样的数据中心?
  - 如何搭建一个好的GPU数据中心？CPU和GPU在同一台服务器，还是GPU池？软件技术。
- DjiNN and Tonic: DNN as a Service and Its Implications for Future Warehouse Scale Computers,
  - DjiNN, an open infrastructure for DNN as a service in WSCs,
  - Tonic Suite, a suite of 7 endto-end applications that span image, speech, and language processing.
- GPU-enabled WSCs improve total cost of ownership over CPU-only designs by 4-20, depending on the composition of the workload.
  - Batching技术, Nvidia MPS，分布式 -> GPU计算效率 （软件的绑定；趋势是垂直一体化方案？）



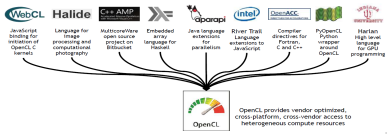
目录

异构计算：编程框架的挑战

- 要懂算法、懂硬件、有并行思维。
- 异构形态多样性给编程带来挑战。很麻烦，受束缚。
- FPGA：
  - 用Verilog、VHDL开发。Altera、Xilinx都提供C/OpenCL->FPGA的编译工具。
- GPU：
  - Intel：OpenCL、Media SDK（视频处理）。
  - AMD：OpenCL。（Apple：Metal Programming Model）
  - Nvidia：CUDA，OpenACC。OpenCL支持弱。
- Intel QAT Card： QAT API
- 理解为什么OpenCL之外还有很多编程框架?

异构并行计算框架是充分发挥异构硬件性能和屏蔽硬件差异的关键，但目前业界不存在一个统一标准。

标准组织



OpenCL

- OpenCL是业界主流公司推动的异构并行计算编程标准。OpenCL属于性能层，业界很多产能层框架对接OpenCL。
- Intel，ARM，高通的异构硬件均支持OpenCL软件设计。
- Intel Xeon+FPGA异构芯片支持OpenCL。

私有框架

- Nvidia CUDA、Apple Metal是针对各自异构硬件设计的计算框架，是封闭系统，但都有广泛的开发者支持。
- CUDA和OpenCL在设计理念上非常详尽，但Nvidia GPU性能业界最强，有能力通过CUDA绑定用户。比如深度学习上Nvidia GPU效果最好，开发者只能选择CUDA。

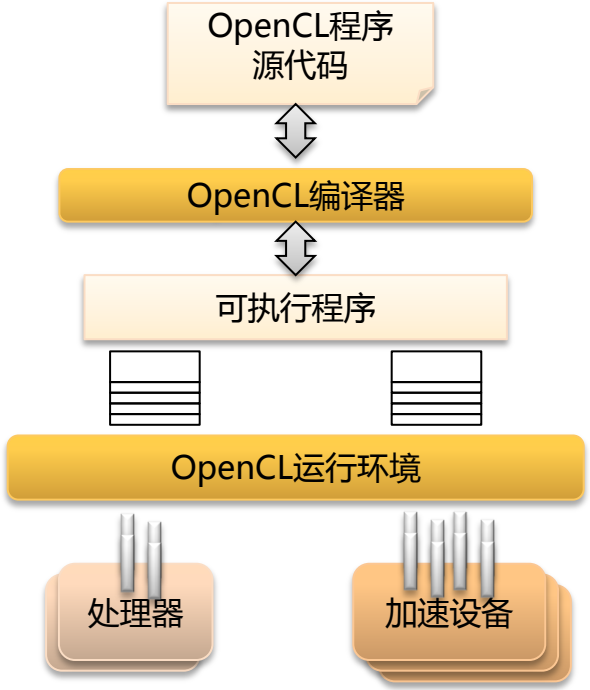
融合异构硬件场景

- 异构硬件的一个趋势是SOC上集成多种异构硬件。比如高通820芯片集成了ARM64+GPU+DSP等多种形态。
- 但传统的计算框架只针对一种硬件设计，无法支持多形态。
- 高通设计了Symphony异构并行计算框架来发挥多形态异构硬件性能，同时适应未来芯片的演进。

异构并行计算框架的思路：

- 应用的并行架构设计是关键。
- 从软件的并行抽象入手，分层设计计算框架，即解耦硬件，又能充分发挥具体硬件的性能极致。
- 参考业界，同时为我司软件的特征优化。

OpenCL：异构编程的主流，得到各大厂家的大力支持，可屏蔽异构硬件与OS差异，简化异构核编程复杂度

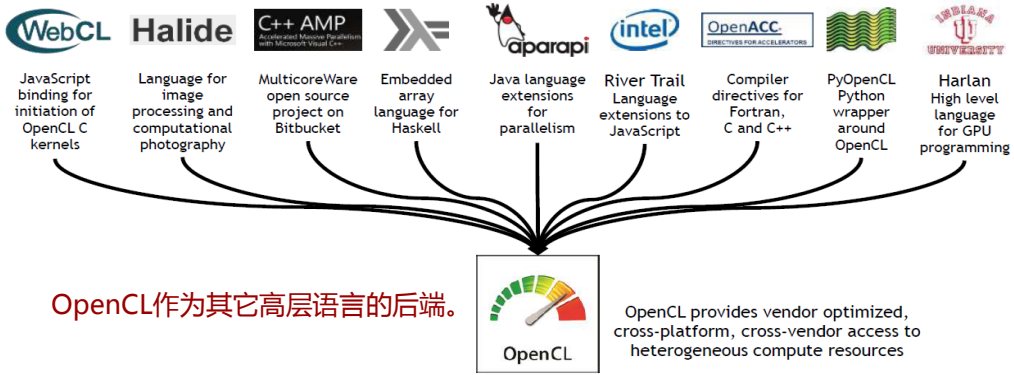


OpenCL由苹果提出，得到业界大多数厂商的支持：Nvidia、Apple、AMD、ARM、INTEL、TI等等，支撑GPGPU、DSP、FPGA等通用硬件加速器。

优势：OpenCL是针对GPGPU大规模数据并行的特性所开发的**底层**编程框架，屏蔽各厂商GPU之间的差异，与芯片设计协同演进；C语言的扩展。

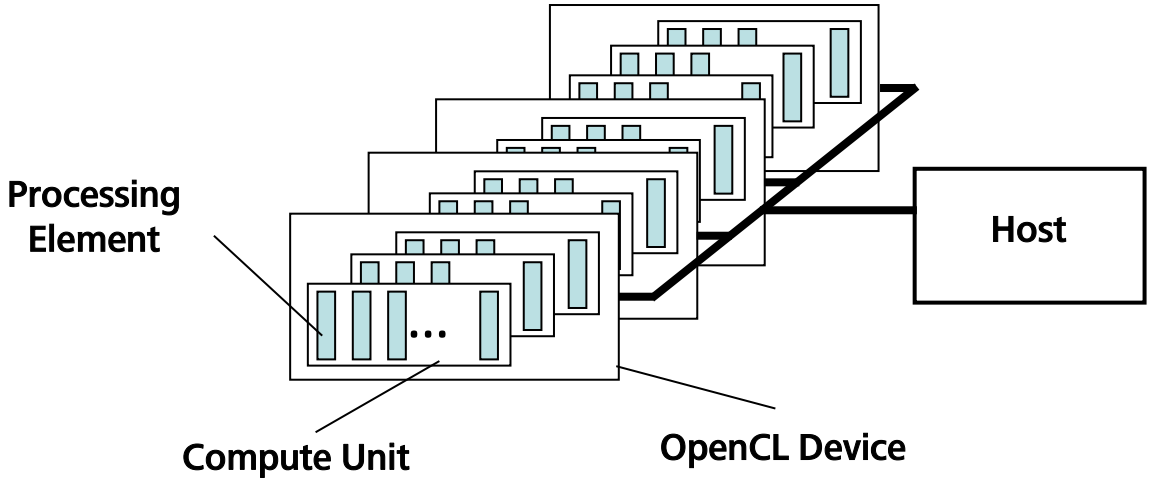
不足：性能的可移植性不足。**被挑战：CUDA、Metal （能被挑战的原因就是CUDA和metal的硬件生态强势，信心足。我们公司怎么做？）**

目前一个趋势：OpenCL考虑SIMD。未来SIMD的性能仍不可忽视。



其它类似规范：CUDA，HSA？
一些概念也慢慢在OpenCL中体现，比如共享存储，加速器动态任务分发。

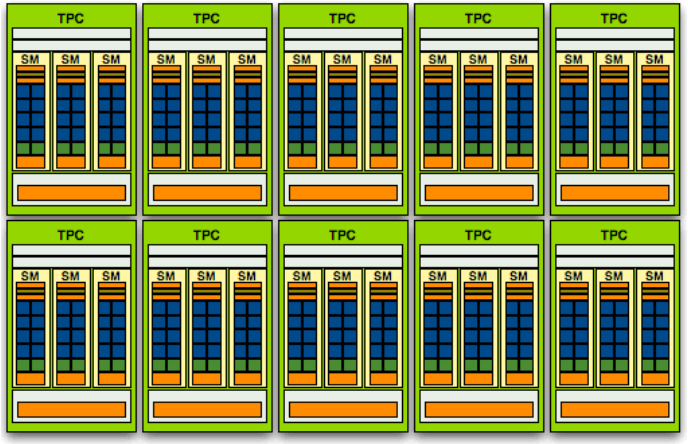
OpenCL编程模型：具备代表性，是理解异构的基础。

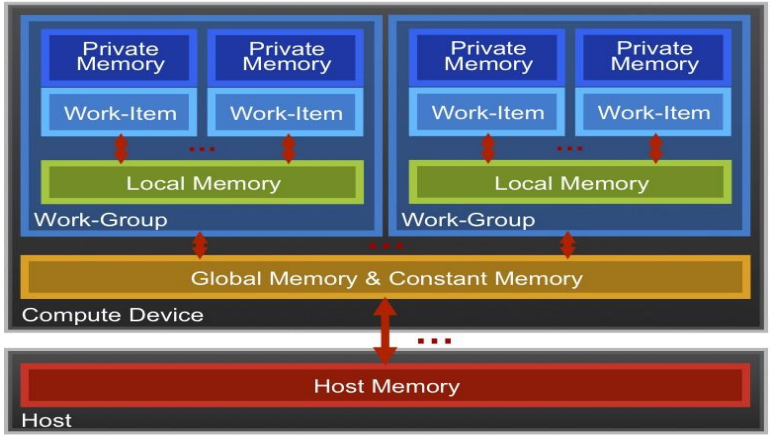


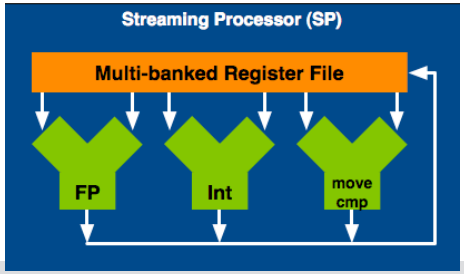
这是概念，由OpenCL的设计者去映射到硬件物理实体。

- Host – Device模型：每个Device由多个CU（Compute Units）组成，每个CU可以划分为多个PE（Processing Elements）。
- Memory divided into host memory and device memory。
- 从这个模型，可以理解OpenCL的API设计原则。

OpenCL编程模型与具体硬件映射实例



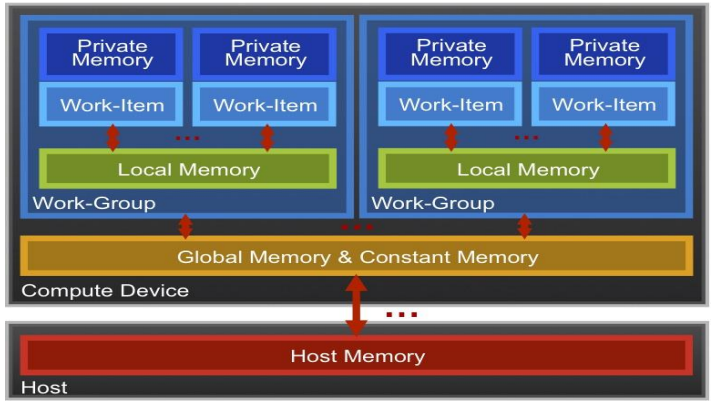


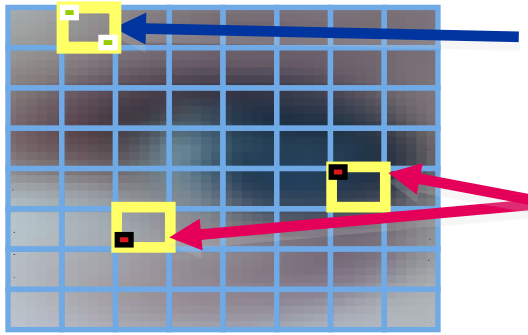


GT200 GPU:
TPC->SM->SP

OpenCL:
Work Group, Work Item,
Memory(global, shared, local)

OpenCL编程模型下的软件设计：绑定硬件架构





Synchronization between work-items possible only within work-groups:
barriers and memory fences

Cannot synchronize between work-groups within a kernel

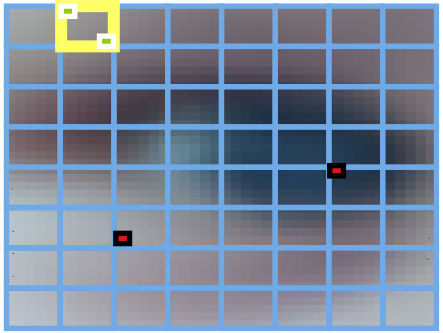
写高性能的OpenCL程序，需要对硬件架构有充分的理解。

CUDA模型和OpenCL基本一致。

OpenCL是数据级并行模型，编程核心是如何划分数据。

```
clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, 1024,
256,0,NULL,NULL);

__kernel void kadd(__global float* input, __global float* output)
{
   int i = get_global_id(0);
   output[i] = input[i] * 100;
}
```

数据级并行难点: CPU GPU对比思考，本质没变，但更复杂



CPU：

```
// Serial
for( i=0; i<N; i++)    add();

// parallel
parallel_for (100, 0, N, add, thread_num=10,  STATIC|DYNAMIC|GUIDED);
```

//OpenCL

```
clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, 1024,
256,0,NULL,NULL);

__kernel void kadd(__global float* input, __global float* output)
{
   int i = get_global_id(0);
   output[i] = input[i] * 100;
}
```

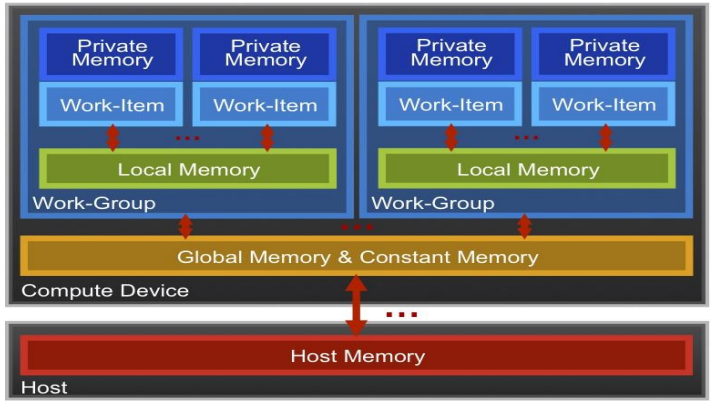
- 块大小划分：并行粒度；
- 调度策略：静态（开销小）、动态（均衡好）、混合；
- 数据局部性：

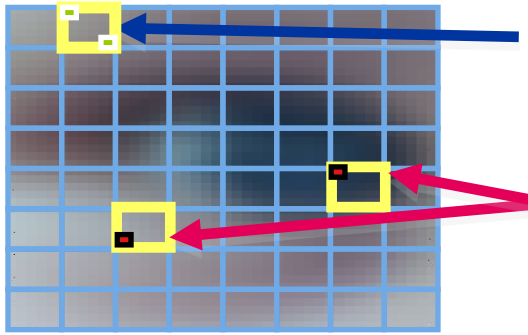
资源隔离
1，线程分组
2，任务分组
3，降低集中式锁竞争

线程协同
1，barrier同步
2，减少线程切换开销

数据亲和性
1，team与NUMA节点绑定
2，任务跟随数据迁移

OpenCL编程模型下的软件设计：绑定硬件架构





Synchronization between work-items possible only within work-groups:
barriers and memory fences

Cannot synchronize between work-groups within a kernel

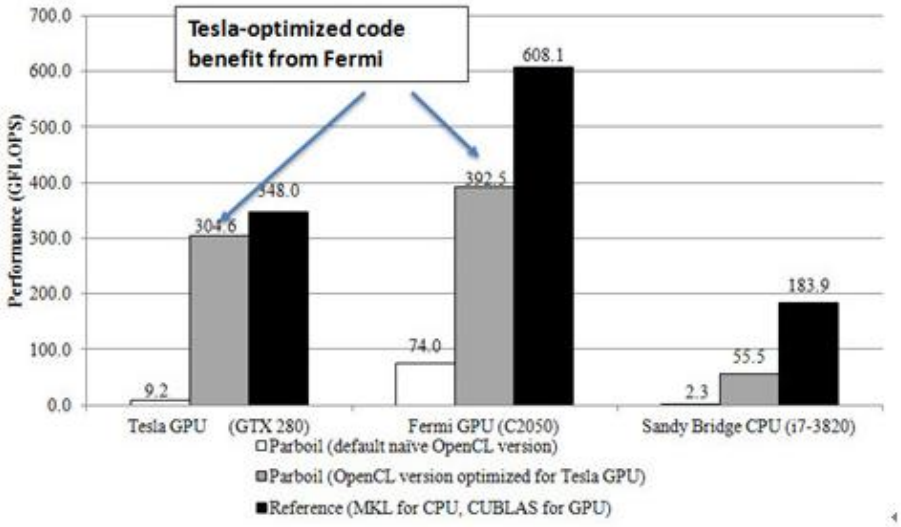
写高性能的OpenCL程序，需要对硬件架构有充分的理解。

CUDA模型和OpenCL基本一致。

OpenCL是数据级并行模型，编程核心是如何划分数据。

```
clEnqueueNDRangeKernel(cmd_queue, kernel, 1, NULL, 1024,
256,0,NULL,NULL);

__kernel void kadd(__global float* input, __global float* output)
{
   int i = get_global_id(0);
   output[i] = input[i] * 100;
}
```

OpenCL的性能一致性问题



OpenCL性能不可迁移。

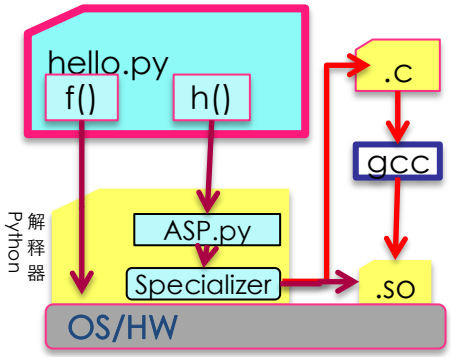
OpenCL编程模型中隐含了很多的硬件特征，比如并行粒度、内存模型、存储层次、资源粒度、**内存带宽和延迟**、ISA差异等，这导致不同的加速器上的OpenCL代码都要重新思考上述特征。

高层模型产生：SEJITS、TANGRAM、Halide、OpenACC（Nvidia）。仍在发展期。

理解为什么会有高层的编程模型，当前学术界多。

Berkeley SETJITS

- 本质：基于并行模式的源到源编译器（Pattern-Specific Compiler）
  - 打破并行开发中编程语言在生产力和性能上的代沟。
  - 采用Python和Ruby做生产力语言，通过编译器生成CUDA/OpenCL/OpenMP。
  - 标记代码块:用DSL编写，这里称之为DSEL（Domain-Specific Embedded Languages），即把DSL嵌入到已有的生产力型语言中。可针对不同的计算模式（比如FFT、stencil、树、图等）设计DSL




```
class LaplacianKernel < Kernel
 def kernel(in_grid, out_grid)
  in_grid.each_interior do |point|
    in_grid.neighbors(point,1).each
      do |x|
     out_grid[point] += 0.2*x.val
```

```
VALUE kern_par(int argc, VALUE* argv, VALUE
self) {
unpack_arrays into in_grid and out_grid;

#pragma omp parallel for default(shared)
private (t_6,t_7,t_8)
for (t_8=1; t_8<256-1; t_8++) {
 for (t_7=1; t_7<256-1; t_7++) {
  for (t_6=1; t_6<256-1; t_6++) {
    int center = INDEX(t_6,t_7,t_8);
    out_grid[center] = (out_grid[center]
      +(0.2*in_grid[INDEX(t_6-1,t_7,t_8)]));
    ...
    out_grid[center] = (out_grid[center]
      +(0.2*in_grid[INDEX(t_6,t_7,t_8+1)]));
;}}}
```

OpenACC异构编程标准

- Nvidia收购PGI的编译器，目前OpenACC->CUDA。
  - An important upcoming feature of the PGI OpenACC compiler is that it not only will speed up OpenACC code on GPUs, but also now on multi-core x86 CPUs

```
#pragma acc parallel num_group(16) num_workers(8)
{
  t=100;
  #pragma acc loop gang
  for (int i=0; i<n; ++i)
    #pragma acc loop worker
    for (int j=0; j<n; ++j)
      B[i][j] = A[i][j] + t;
}
```

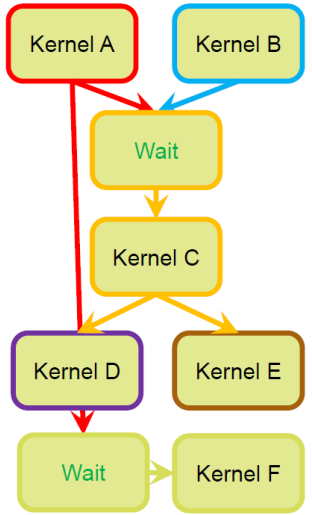
解释：

1）parallel+loop实例，每个gang都执行t=100。

2）loop通过gang和worker两个层次并行。

3）细粒度的控制并行度（num_*）。

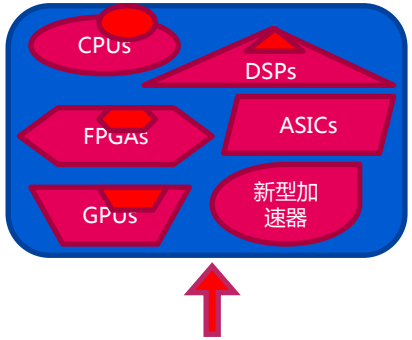
```
!$acc parallel loop async(1)
<Kernel A>
!$acc parallel loop async(2)
<Kernel B>

!$acc wait( 1, 2 ) async( 3 )

!$acc parallel loop async(3)
!! wait( 1, 2 )
<Kernel C>

!$acc parallel loop async(4) &
!$acc     wait(3)
<Kernel D>

!$acc parallel loop async(5) &
!$acc     wait(3)
<Kernel E>

!$acc wait( 1 )

<Kernel F>
```



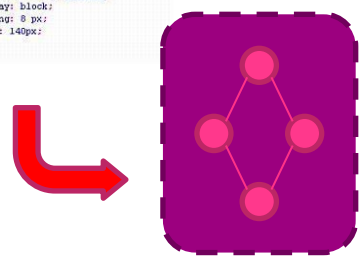
趋势：以算法组件为中心的编程框架，面向系统级并行

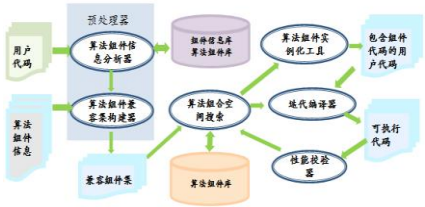
当下或未来的硬件计算平台

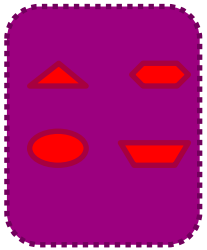
普通程序员



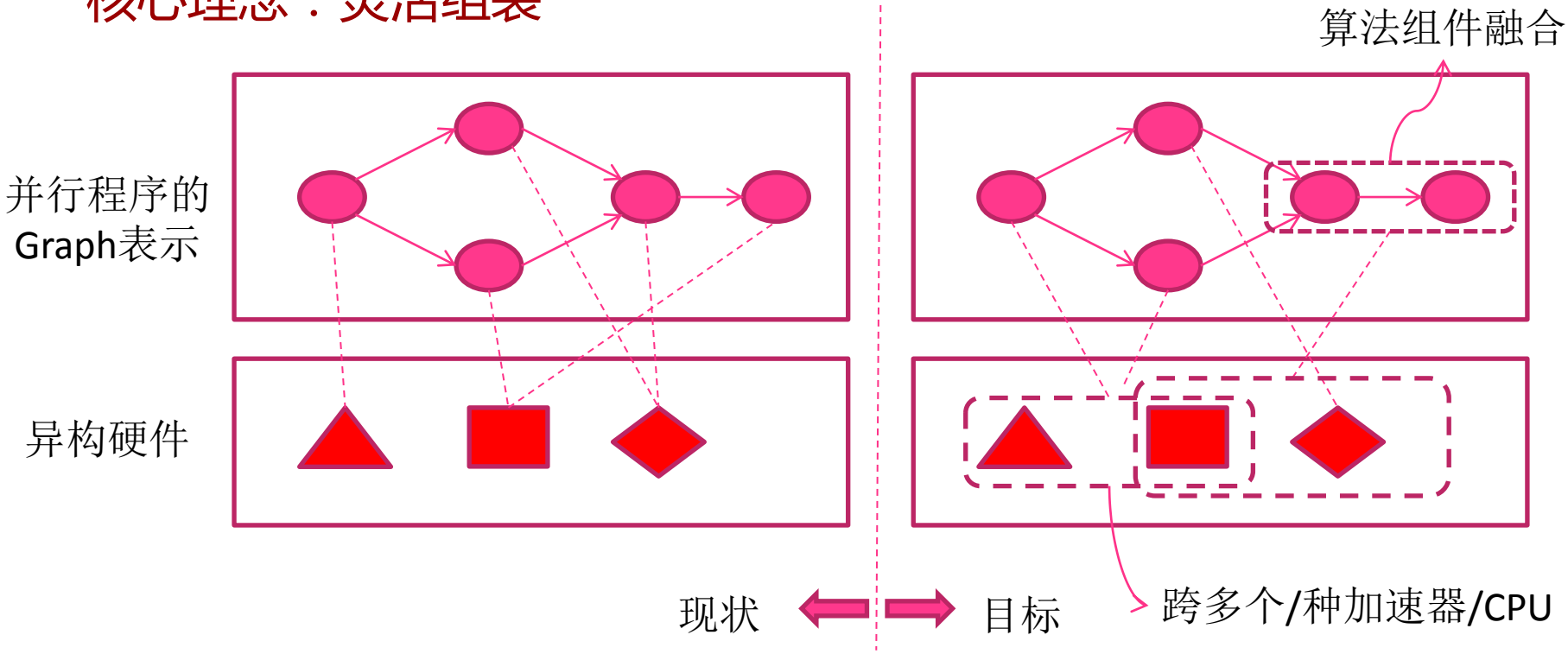
专家程序员





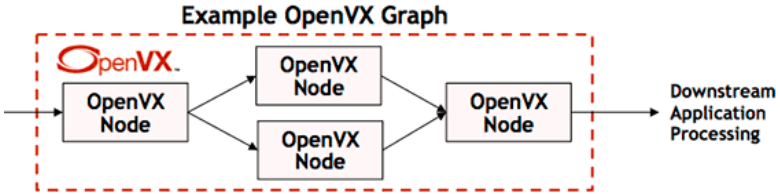


核心理念：灵活组装



业界案例：近几年异构编程框架主流趋势

- **OpenVX – Vision Acceleration (2014)**

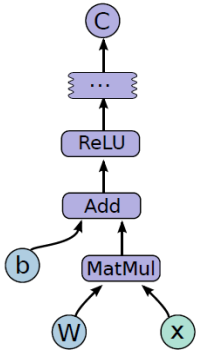


面向领域，用户可以自己增加算子库（CPU、GPU、DSP）。

Absolute Difference	Integral Image
Image Pyramid	Optical Flow Pyramid (LK)
Remap	Scale Image
Histogram (generate, equalize)	Thresholding
Accumulate (also squared, weighted)	Image Arithmetics (+, -, *)
Filters (box, custom, Gaussian, median, Sobel)	Corners (Fast, Harris)
Bitwise And, Xor, Or, Not	Edges (Canny)
Channel combine, extract	Phase, Magnitude
Convert (bit depth, color, table lookup)	Dilate, Erode
Stats (mean, std. dev., min/max location)	Warp (affine, perspective)

Table 1: OpenVX Base Kernels Version 1.0 (Provisional).

- **TensorFlow – Machine Learning Acceleration**

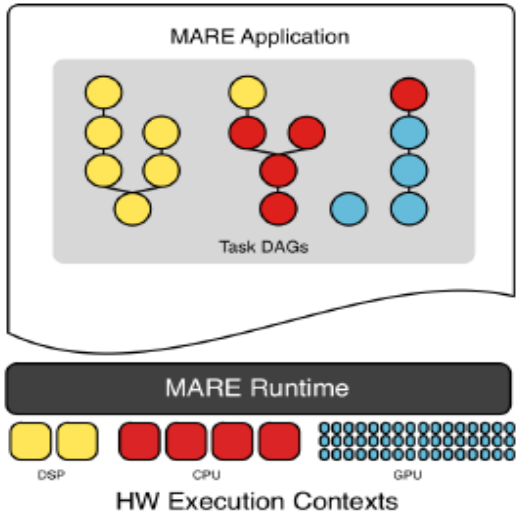


核心代码中1/3为算子库（CUDA）。

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Table 1: Example TensorFlow operation types

Qualcomm Symphony: Multicore Asynchronous Runtime Environment



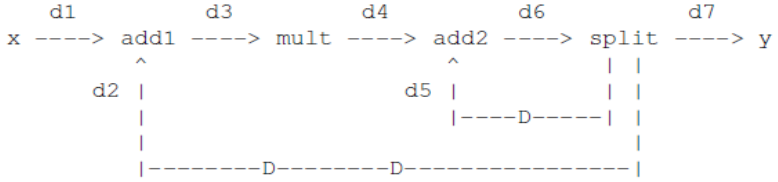
关键概念：

•异步： MARE中对并行表达的主要抽象是*任务*，通过*任务图*来描述业务。

• pfor_each/pfor_each_async/ptransform，并发容器：双向队列、队列、栈，通用并行算法：排序、搜索、BFS等。

•异构： 对OpenCL中的Buffer、Event等概念进行了高层抽象。

•支持同步数据流模式




```
mare::sdf_graph_ptr g = mare::create_sdf_graph();

mare::create_sdf_node(g, add1, mare::with_inputs(dc1,
dd_dc), mare::with_outputs(dc2));

mare::create_sdf_node(g, mult, mare::with_inputs(dc2),
      mare::with_outputs(dc3));

mare::launch_and_wait(g, num_iterations);
```

• 面向CPU+GPU+DSP，异构编程通过封装OpenCL实现。

• 面向Android移动设备，优化性能和功耗。动机：适应程序运行时移动设备上资源的动态变化。

• 支持丰富的并行模式表达：数据并行、任务并行、流水线、流式、异构。

HiHCS+LiteTask实践

关键问题：

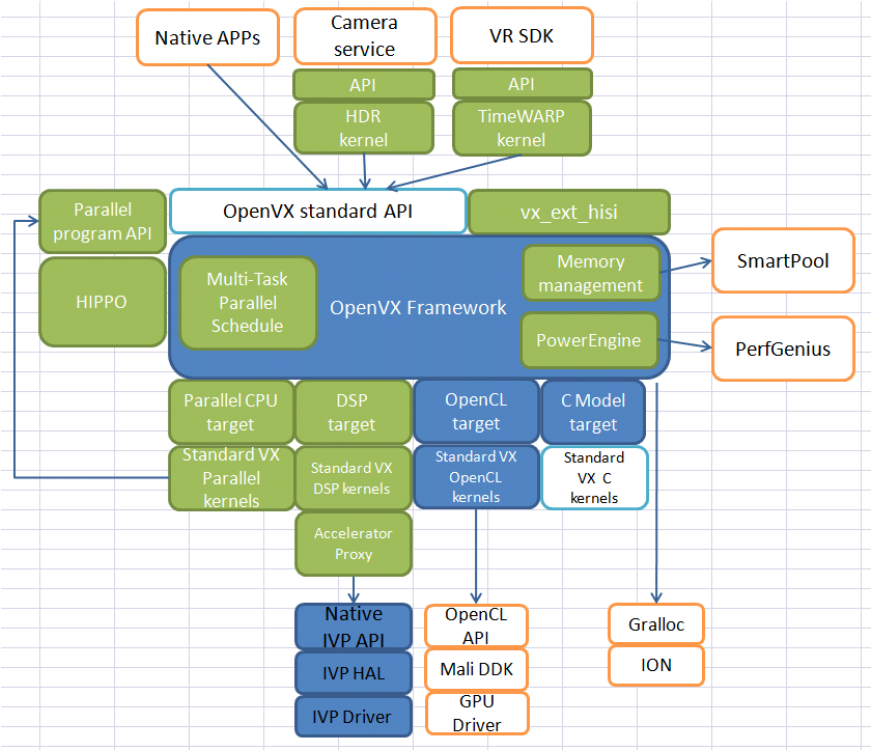
•充分利用所有异构硬件；

•多核CPU能力发挥；

•一套异构并行架构；

当前在Kirin960上运行。

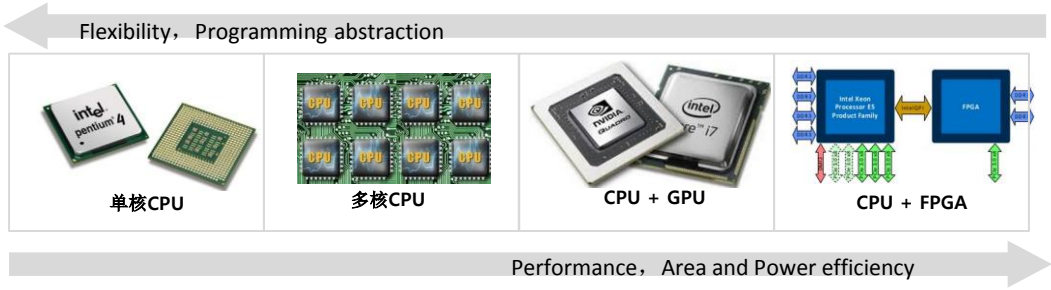
手机上什么样的应用适合这种框架？

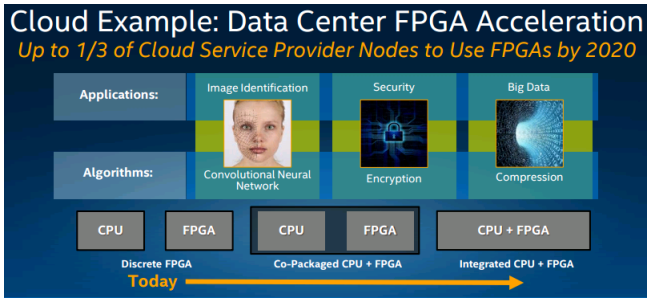


目录

- 异构计算简介
- 异构计算的软件挑战
  - 编程框架的挑战：从算法级到系统级
  - 新硬件架构对软件的挑战
- 关键的软件技术

重新认识FPGA的价值：过去和现在的差异





•FPGA v.s. GPU/CPU：

1. **软件定义的硬件架构**：GPU/CPU硬件固定，其并行性设计是适应固定硬件。而FPGA的硬件逻辑可以通过软件动态改变，从硬件的角度来适配软件，从而获得更高的计算性能。
2. **更高并行性& 更高的能效比**：FPGA拥有更丰富的计算资源组件，从而能够满足更多并行计算需求。并且能够充分发掘软件算法中的并行性，降低功耗。

•新架构融合CPU+FPGA的发展趋势：

1. 异构核首次作为一等公民：通过CPU+FPGA的融合设计，由主机+外设的Offloading模式转变为异构多核片上系统设计，CPU与FPGA地位等同，通信方式由板级转向片内。

2. OpenCL带来了FPGA的编程革命：提高了FPGA的可编程性，将程序员从复杂的硬件电路设计中解救出来，更专注于系统/算法的设计。

Intel预测2020年1/3的计算节点采用FPGA。

新架构对软件的挑战：

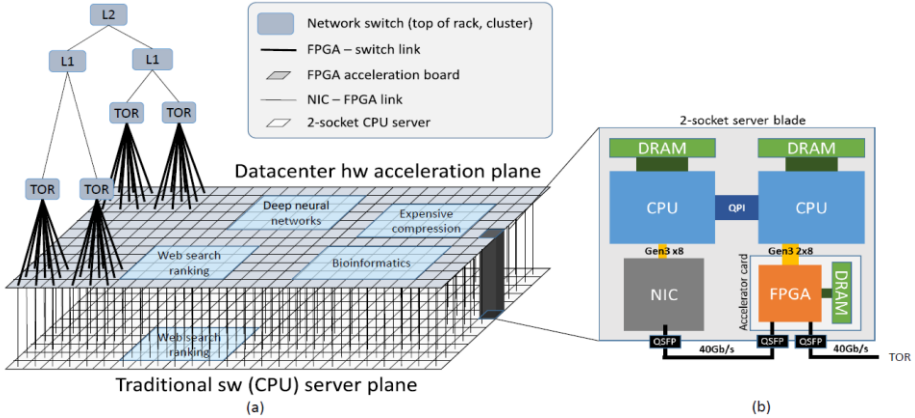
1）支持CPU+FPGA融合并行的计算框架，包括FPGA编译和高层语言设计。

2）大规模分布式FPGA资源管理与调度

3）高性能FPGA算法库

随着FPGA的广泛使用，成本逐步降低，作为软件定义实现快速在线指令优化，对整个编译器、运行框架、OS产生巨大影响

核心是认识FPGA成为计算的一等公民，而不是Offloading



Microsoft Catapult: 应用于微软数据中心的FPGA加速，其中FPGA使用的是Altera Straix V，支持FPGA的点对点直接通信。

核心理念：

FPGA主动计算，分布式计算框架要融入到FPGA中，而不是把任务offloading到FPGA上。

新硬件架构，主要是CAPI和QPI互联：

CPU和加速器接口是重要的，这将改变原来的offloading模型，加速器真正的主动参与计算（同一内存空间，直接操作虚拟内存，高带宽访问）。通信加速比变了。

建议关注新硬件架构对分布式并行软件的影响，

•比如什么样的软件以前不适合加速，现在适合了，它的特征是什么。

•对云架构的影响，之前提分布式加速（Cloud-Scale Acceleration），但跨机器的加速（基于加速资源池）还是受限网络时延。OpenCAPI的互联是否是的加速资源池是独立自主，而非受控于HOST。

目录

- 异构计算简介
- 异构计算的软件挑战
  - 编程框架的挑战：从算法级到系统级
  - 新硬件架构对软件的挑战
- 关键的软件技术

业界公司都发力异构并行，其背后驱动力各不相同。

应用厂商：掌握应用场景，注重编程产能和跨平台，非常清晰的**并行计算模型**。

- Ericsson: Dataflow Model
- Google: TensorFlow
- Facebook: FBLearner Flow
- Amazon： DSSTNE

系统/中间件厂商：追求开发环境的完备性，注重**并行计算模型**。

- Microsot: Concurrent Runtime, C++AMP, Visual Studio, Orleans
- Apple： Metal/OpenCL/GCD

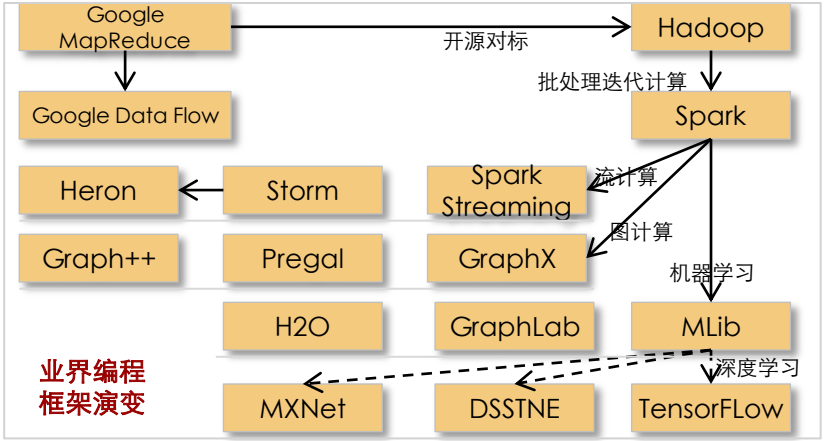
硬件厂商：挖掘硬件并行潜力，重性能。必须自研！

- Intel: TBB,Cilk,OpenMP, OpenCL, OpenVX, MPI
- Nvidia :CUDA, 加速库
- Qualcomm: Symphony

标准组织和社区:

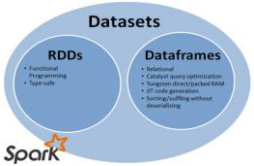
- OpenCL
- OpenVX
- OpenMP
- C++1x

分布式任务计算框架的架构演变



● 业界计算框架的3大趋势

1. **分布式计算模型的统一**。超越MR/stream等单点模型，数据流成为分布式高层的统一模型，计算框架开始在模型层融合。
   低时延驱动着针对小数据块和小任务的优化；高层能融合运行时优化的分布式数据抽象成为主流。性能是永恒动力！
2. **异构硬件驱动着架构设计改变**。特别是在极致性能场景开始引入GPU/FPGA等，模型和调度都考虑如何屏蔽硬件差异。
3. **框架上云，驱动业务全面云化**。阿里云BatchCompute，在公有云上提供分布式任务计算框架服务。SPARK、 深度学习框架也都开始以云服务形态呈现，与DCOS融合是关键。



•从1.6到2.0的三个关键特性都是为了**提高性能**。

•从SPARK2.0开始：Datasets将取代RDD，提供更高的性能。

•Batch和streaming两种模型将统一为一套。

新模型：性能

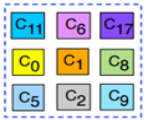
APACHE STORM™

HERON

•Topology consumes 600 cores at 20-30% CPU in Storm.

•Enhancing Storm would take too long and no other system met their scaling, throughput and **latency** needs.

新架构：轻量化、低时延



Micropartitioning a computational stage more finely than one partition per core has significant benefits. Micropartitioning enables better load balancing and **faster completion times.**

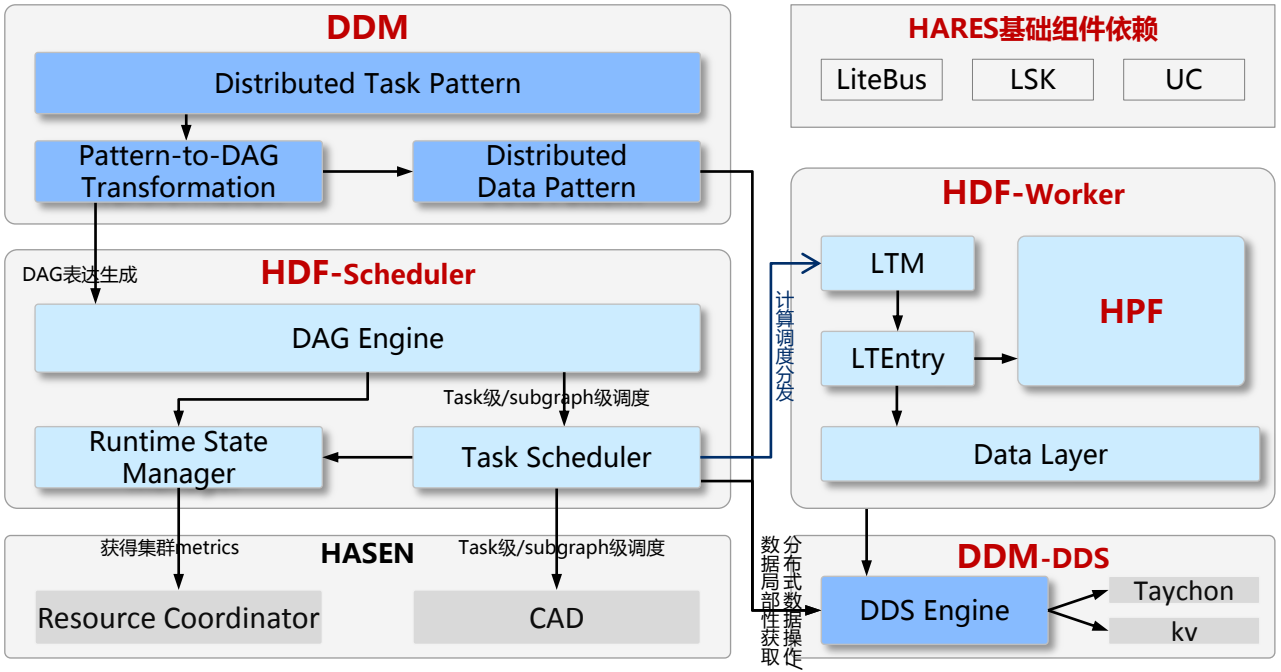
Building Software Systems at Google and Lessons Learned, ***Jeff Dean****.*

新理念：低时延

LiteTask的驱动力：产品面临低时延计算与异构硬件的挑战

产品场景	产品价值	异构硬件	计算特征与平台诉求
5G基站	•无线频谱资源高效利用：核心是解决小区调度百倍计算量挑战。 •更低TTI时延计算保证：TTI级计算资源动态响应，高QoS并行计算。	•4*16核 ARM64，自研Neon和加速器。 •15核2线程HiDSP ，HAC加速器。	•0.125ms (TTI)下的百任务DAG图并行计算。 •资源实时响应的动态DAG图，计算任务图随输入和配置实时变化。
SDN统一控制器	•高性能集中重路由：是运营商比拼的硬指标和排他项，从小时级到分钟级到秒级。 •高性能生存性分析：支撑网络敏捷运维，客户竞标关键项，从天到小时级。	•单机 Intel X86（60核）。 •多机 Intel X86。	• 松散数据依赖下的万级任务迭代计算，任务粒度50-200ms，多层嵌套并行。 •存在多台控制器，多计算共享计算资源。
终端	•海思Kirin 960芯片异构框架：充分利用CPU/GPU/DSP等异构资源，HDR时延小于33.3ms，TimeWrap时延小于5.5ms。 •下一代终端计算平台：针对智慧手机，端云结合的轻量级分布式并行AI计算框架。	•Android，ARM64+GPU，大小核。 •云侧：Intel X86+Nvidia GPU，单机多加速卡。	•异构的数据流图并行，支持数据重用优化和异构硬件上高效调度和资源管理。 • 神经网络层次不断加深，层内和层间需要组合的“数据并行+模型并行” 。大规模参数通信和更新，BSP不适合，需要异步或半异步模型。
平安城市以图搜图	•高密度计算：10核X86 CPU支持4路1080p视频实时分析。 •极速查询反应：30s完成500w视频目标的查询计算。	•2x SIGMA，80个Intel XEON 10核CPU。 •GPU。	•百万级非均衡μs/ms级任务并行计算，流水线并行内嵌数据并行。 •高QoS多流分布式计算，波峰与波谷计算量相差数10倍，动态变化。

LiteTask架构设计逻辑试图（一层）



LiteTask架构主要由三部分组成。

1. DDM：深层分布式计算模型
Deep-level Distributed Model

包括任务模型和数据模型，以及数据存储的分布式数据集（DDS），DDS负责数据的put/get/update以及局部性操作。计算模型最终转化为DAG表达的计算图。

Parallel_for, pipeline, Graph

2. HDF: 高吞吐分布式计算框架
High-performance and Distributed computing Framework

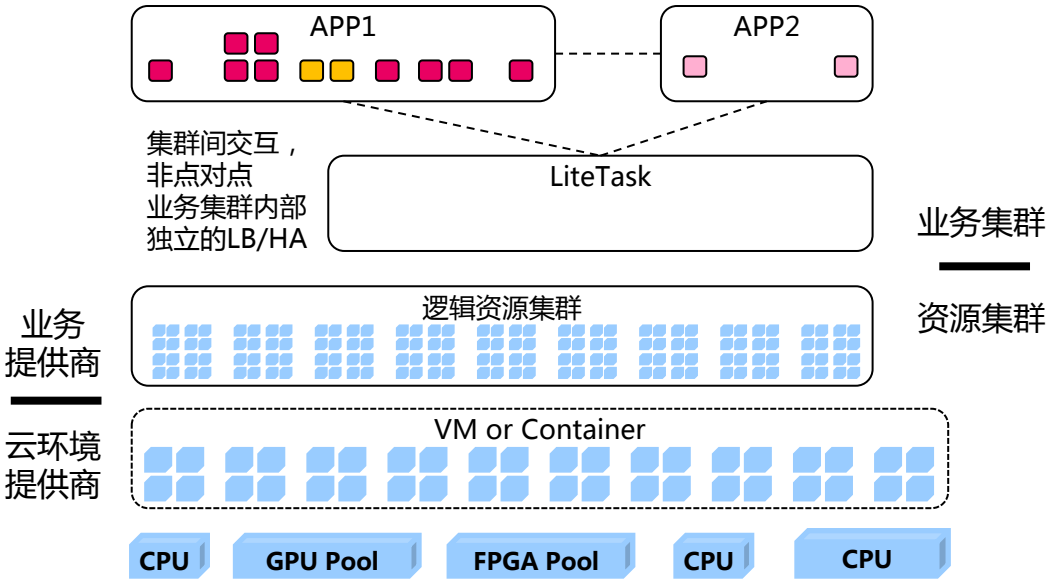
由Scheduler和Worker组成，轻量级(百节点<10%)。支持ms级微任务的大规模分布式计算，追求低时延！

LTM为节点上任务管理器，负责任务控制和容错；LTEntry为微任务入口，可调用HPF实现节点上的并行计算。

3. HPF: 异构并行计算框架
Heterogeneous and Parallel computing Framework

支持多核X86/ARM64/GPU/FPGA上的并行计算，超轻量级，支持μs级任务计算，可独立部署于单机环境。

设计理念：统一的资源集群+独立的业务集群



- 每个业务可以独立的弹性伸缩，业务之间的流量通过集群来适应底层能力差异
- 不同的业务集群允许采用不同的框架实现，更好的兼容演进和集成
- 统一的资源池化，允许不同的业务框架灵活共享与隔离资源

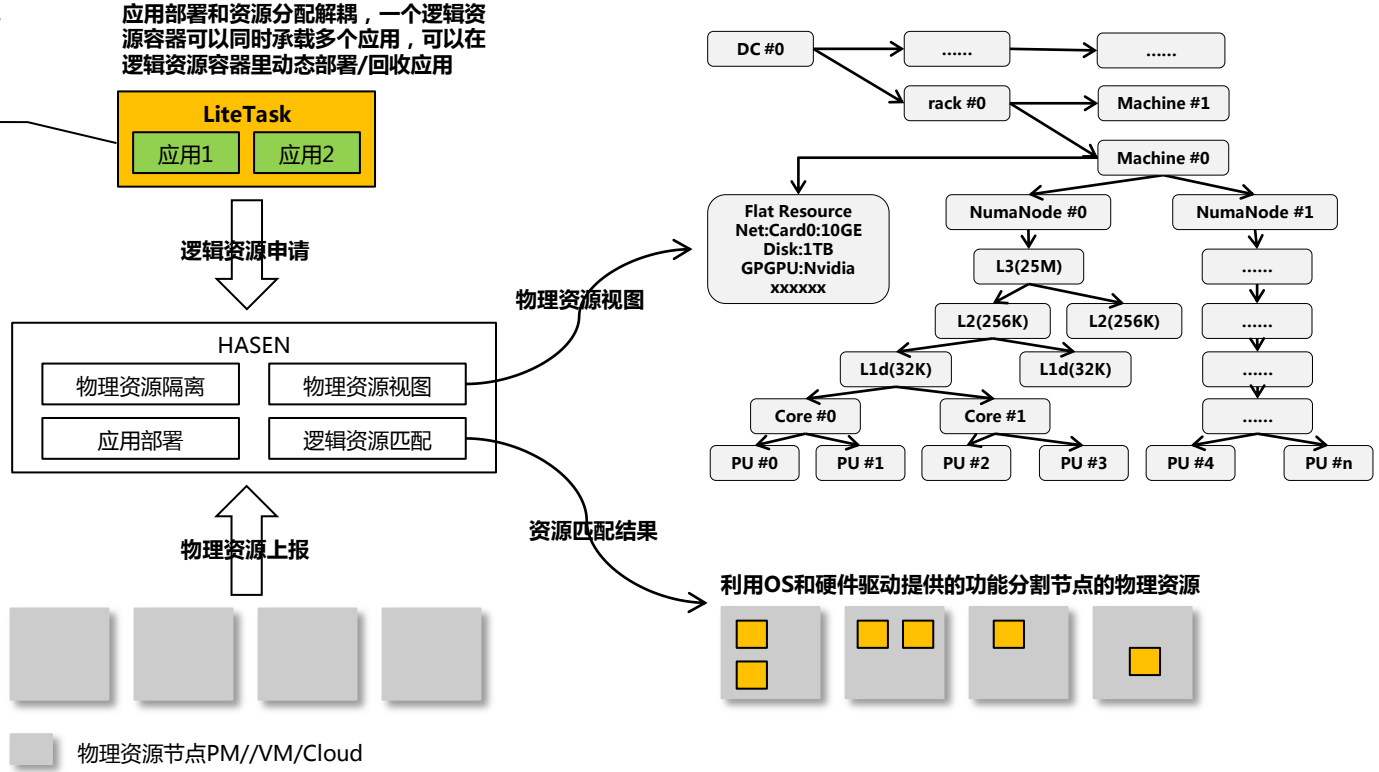
设计之初即考虑云

HASEN： 硬件资源抽象，支持加速资源池化

资源容器里可以包含多种逻辑资源，用户可以直接定义并申请逻辑资源容器，某些逻辑资源支持动态调整

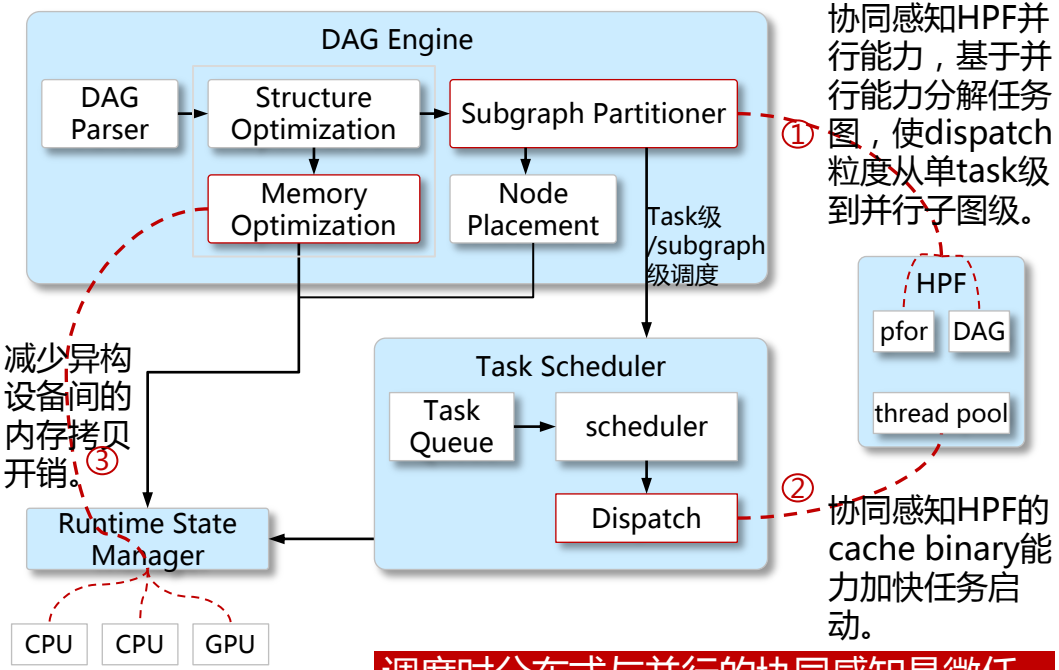
CpuInfo {
  指令集：X86_64
  core数：4个
  频率：2000hz
  FPU：SSE
  ……
}
L3CacheSize：2MB
MemInfo {
  大小：64G
  带宽：25Gb/s
  ……
}
NetIO：3.5Gb/s
DiskInfo {
  大小：150Gb
  IO带宽：200Mb/s
}
GPGPU {
  种类：NVIDIA
  queue个数：32
  ……
}
FPGA {
  Crypto: 200Mbps
  ……
}
DSP {
}

应用部署和资源分配解耦，一个逻辑资源容器可以同时承载多个应用，可以在逻辑资源容器里动态部署/回收应用



资源池化的目的是实现资源的按需使用，技术手段是软硬件解耦，通过定义逻辑资源抽象，可以将应用和物理运行环境解耦。

关键特性：轻量级分布式任务计算框架



调度时分布式与并行的协同感知是微任务轻量级框架的关键和差异化所在。

微任务轻量化的3个维度及关键技术：

1. 框架资源开销低

① 高吞吐：DAG Subgraph Partitioner，通过“并发任务子图”支持高吞吐调度，满足大规模微任务计算分发的压力需求。

单调度器10-100K级吞吐 vs. 当前框架逐个task分发K级吞吐 （避免调度器本身的分布式）

② 启动快：HPF并行线程池缓存任务，支持μs级任务启动。 *vs.当前框架ms级启动（JVM）*

③ 低内存：DAG Memory Optimization:内存复用和异构内存优化，感知底层硬件，减少内存占用。

2. 支持计算低时延

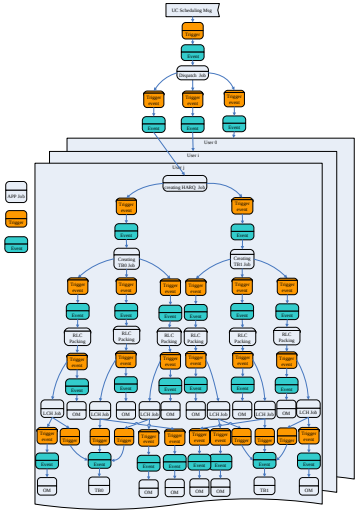
森林模型调度，针对模型分解的微任务通过多维度调度器组合充分利用系统资源，减少尾延迟。

ms级@百节点 vs. 当前框架秒级任务@百节点

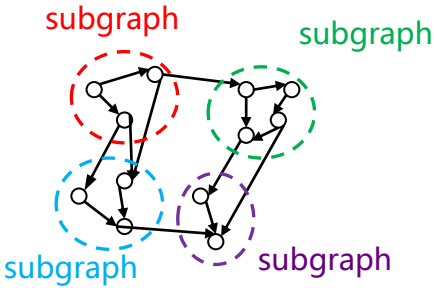
3. 架构的可重构设计

灵活可组合的架构组件，可集成多种实现策略，快速支持新模型。

关键特性：DAG符号计算引擎，静态分析与动态执行相结合



业务模型的DAG图表达

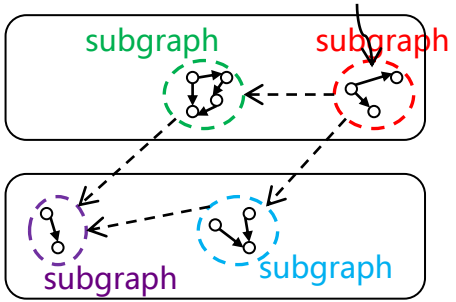


DAG静态分析

基于Job依赖以及数据通信开销，将原DAG划分成子DAG（job list），作为策略传给DAG动态执行引擎。

•通信开销感知的job融合：job list间的数据依赖小。

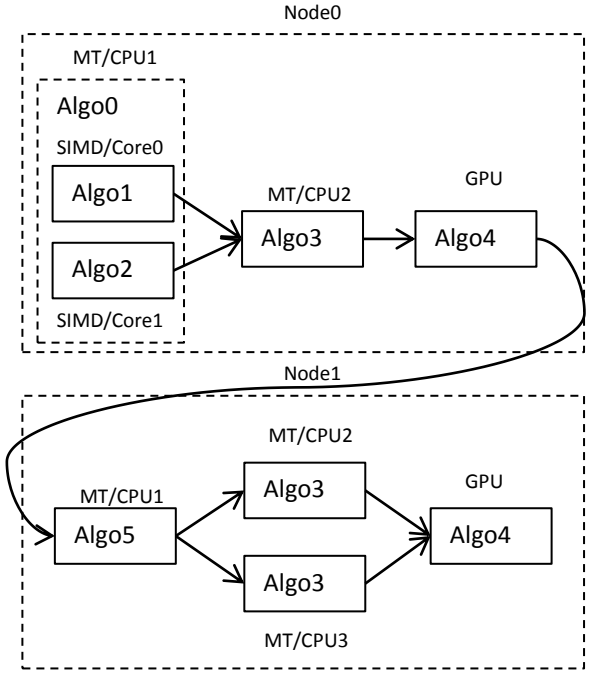
•存储层次感知的子图并行度分析

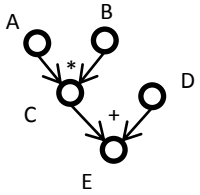


DAG动态执行

高效的依赖解析，多种分发策略均衡。以subgraph提交给任务调度器。

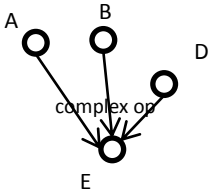
Graph的高效调度





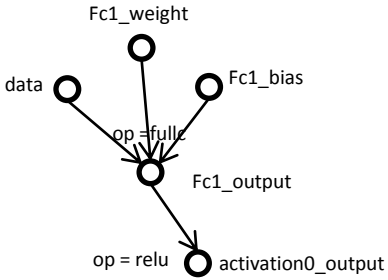
细粒度图

结点融合

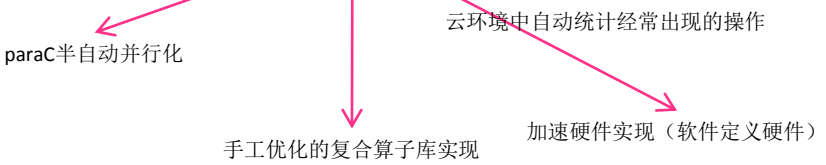


挑战：高效调度细粒度图

代码生成



粗粒度图

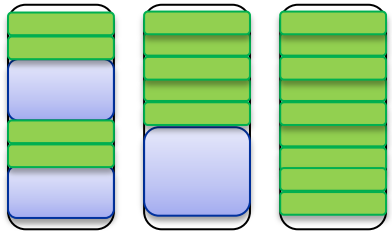


算法组件Graph(静态/动态)调度：
算法组件可调度至CPU/GPU/SIMD、组件可融合、降低开销

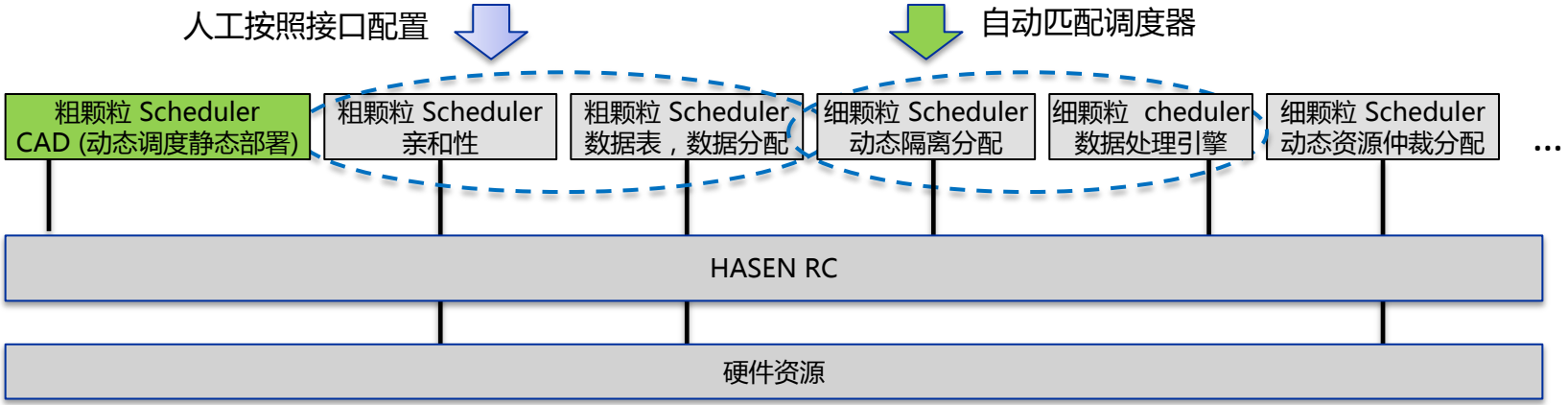
关键特性：支持低时延计算的森林模型调度，多维度调度器组合

森林模型，各个物种独立生存法则，森林天道统筹成有机整体。

阳光	集群芯片能力
土地面积	处理时间
大树	粗颗粒模块、任务
小树、灌木	细颗粒任务



目标：微任务对集群资源的充分利用 ，缓解尾延迟，达到低时延。



不同调度器侧重不同，全系统不同任务使用不同调度器，综合平衡各方需求

亲和调度、时延稳定最短 与 负载均衡

满规格运行 与 轻负载运行变化感知

静态部署 与 动态调度

总结

- 异构硬件编程框架：从算法级到系统级的演变原因
- FPGA带来的挑战要从软件和硬件的系统来看。
- 以LiteTask为例看软件技术挑战。
  - 算法设计：人的智慧+领域公共库
  - 异构的计算模型：parallel_for, pipeline, graph
  - 异构资源池化：业务集群-资源集群，适应不同分布式硬件。
  - 轻量化：调度性能，内存开销
- FPGA领域：软件和硬件协同发力应对挑战
- 我们策略：标准（多样化参与）、自研

Thank You!