GPU 并行计算基础

——从一名工程师的视角

吴 超

中国科学技术大学超级计算中心

2023 年 6 月 1 日

1. 并行计算简介
2. GPU 硬件基础
3. GPU 软件编程
4. 小结与展望
5. 参考文献

1 并行计算简介

2 GPU 硬件基础

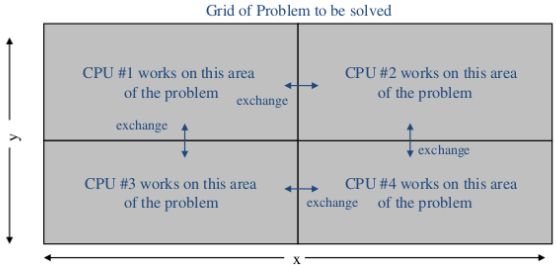
3 GPU 软件编程

4 小结与展望

5 参考文献

什么是并行计算

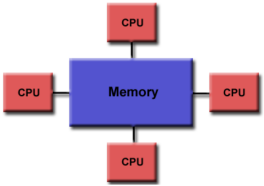
- 简单地讲，并行计算（Parallel Computing）就是在并行计算机上所做的计算 [陈 11]
- 从工程角度，并行计算可以看作使用多处理器或多计算机联合工作，解决一个通用计算任务的过程



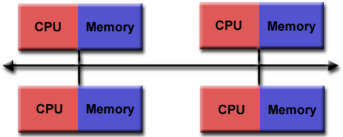
- 通常，并行计算可与高性能计算、超级计算视为同义词

什么是并行计算机

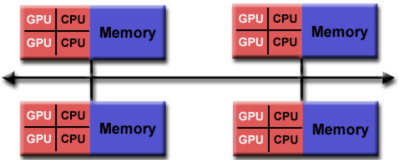
- 多处理器 [LLNb]



- 多计算机 [LLNb]



- 混合结构（带协处理器）[LLNb]



为什么使用并行计算

- 更快的速度：并行集群可以由廉价的商品组件构建，投入更多的资源任务将缩短完成时间

$$T_p \approx \frac{T_s}{N}$$

- 更高的容量：许多计算问题是如此之大和复杂，尤其是在计算机内存有限的情况下，单台计算机不可能解决

$$C_p \approx N * C_s$$

为什么使用并行计算（硬件）

- 超级计算机

图 1: Frontier(美国)[LAB]

图 2: 太湖之光 (中国)[李]

为什么使用并行计算（硬件）

- 个人电脑

图 3: 笔记本电脑

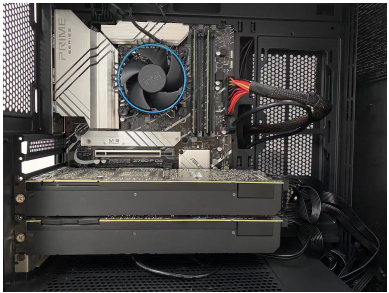


图 4: 工作站电脑

为什么使用并行计算（硬件）

- 手机

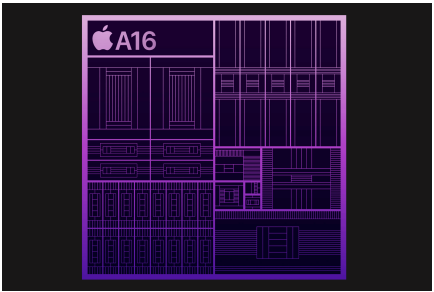


图 5: 苹果 A16 芯片

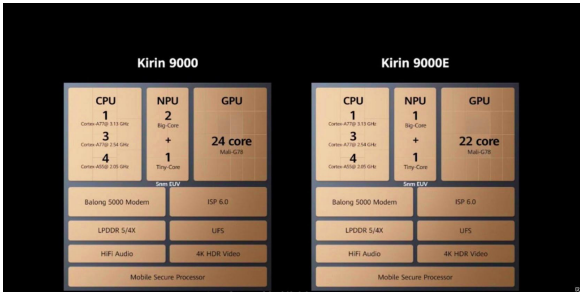
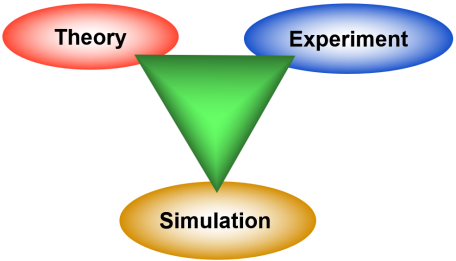


图 6: 华为麒麟芯片

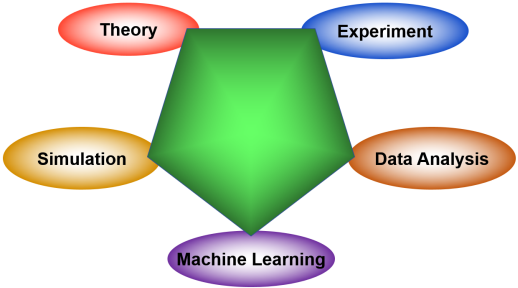
怎样使用并行计算（软件）

- 科学研究的三大支柱 [Yel]



- 理论
- 实验
- 数值模拟

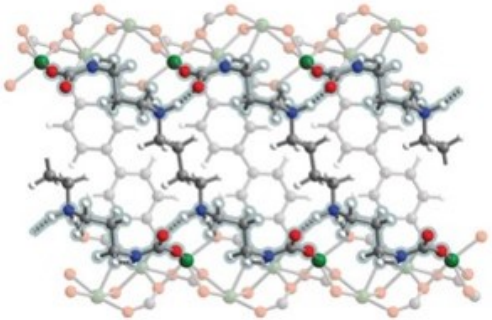
- 科学研究的第五范式 [Yel]



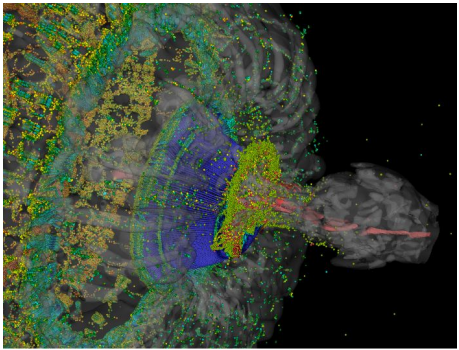
- ＋ 数据分析
- ＋ 机器学习

怎样使用并行计算（软件：数值模拟）

- 第一性原理 [Yel]

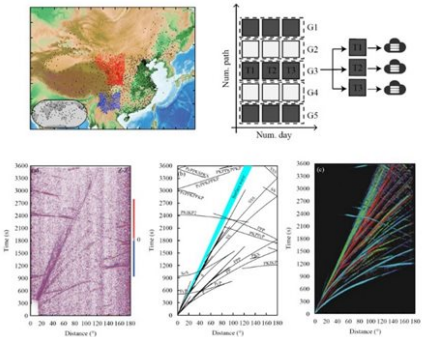


- 核物理模拟 [LLNa]

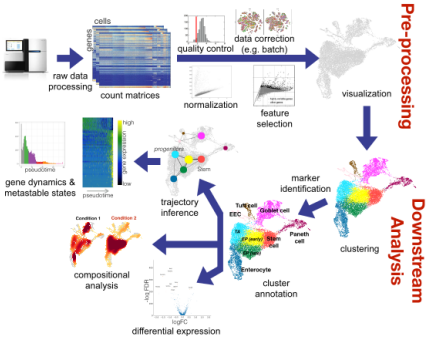


怎样使用并行计算（软件：数据分析）

- 背景噪声成像 [中]

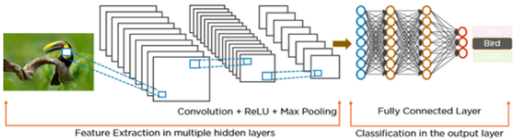


- 基因数据分析 [the]

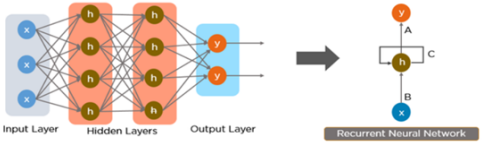


怎样使用并行计算（软件：机器学习）

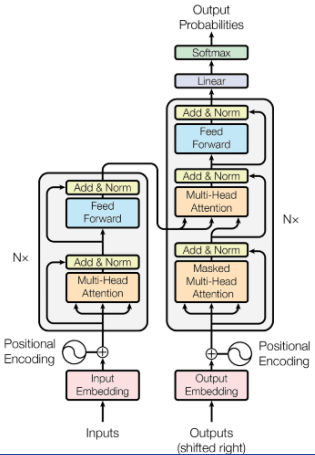
- CNN[Tri]



- RNN[Tri]



- Transformer[Cri]



1 并行计算简介

2 GPU 硬件基础

3 GPU 软件编程

4 小结与展望

5 参考文献

微处理器发展趋势

- 1971 年第一块微处理器 4004 在 Intel 公司诞生，78 年推出 8086 芯片（X86 指令集），79 年推出 8088 应用于个人电脑
- 性能限制：主频从 KHz 到 MHz 到 GHz，主频提升主导了 CPU 的技术路线，主频越高意味着功耗更高、发热越严重

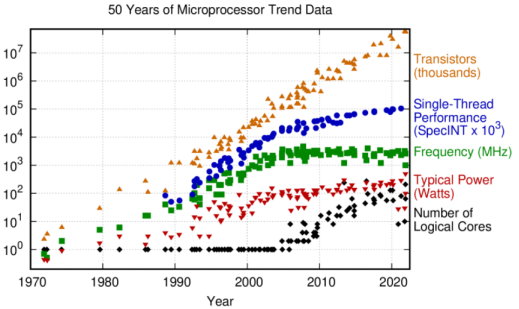


图 7: 微处理器发展趋势 [kar]

多核与众核

- 多核处理器 (Multicore Processor)：是在单个计算组件中加入两个或以上的独立实体中央处理单元。这些核心可以分别独立地执行程序指令，利用并行计算的能力加快程序的执行速度
- 众核处理器 (Manycore Processor)：众核处理器是专为高度并行处理而设计的特殊类型的处理器，包含大量更简单的、独立处理器内核（从数十核到上千核甚至更多）。众核处理器广泛用于嵌入式计算机和高性能计算

CPU vs GPU

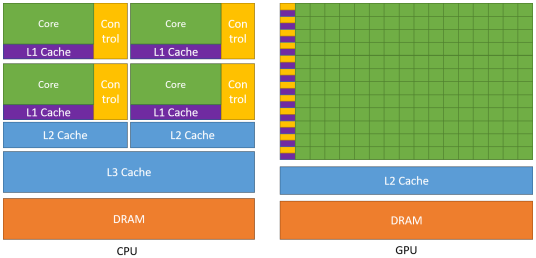
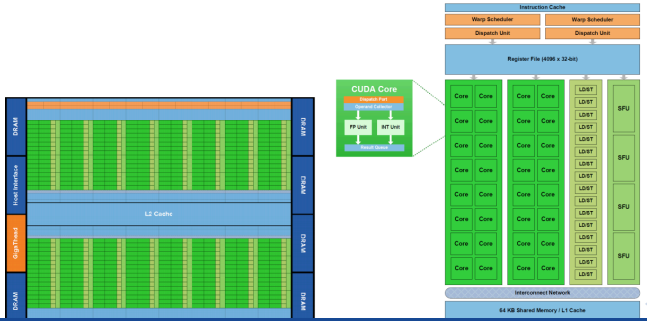


图 8: 微架构示意图[NVIa]

- CPU 芯片 Cache 占据大量空间，有复杂的控制逻辑和诸多优化电路，适合逻辑复杂的任务
- GPU 采用了数量众多的计算单元和超长的流水线，只有非常简单的控制逻辑，适合高并行度、大数据量计算

GPU 架构

- Fermi 硬件架构 [CGM18]
  - SP(Streaming Processor)：流处理器，是 GPU 最基本的处理单元，从 Fermi 架构开始也被叫做 CUDA core
  - SM（Streaming Multiprocessor)：一个 SM 由多个 CUDA core 组成，不同架构的 SM 有不同数量的 CUDA core
  - GPU 由可扩展的 SM 阵列组成（Fermi 中 16 个）



GPU 演进

- 架构演进：Tesla -> Fermi(2010) -> Kepler -> Maxwell -> Pascal -> Volta -> Turing -> Ampere(2020)->Hopper(2022)
- 性能提升

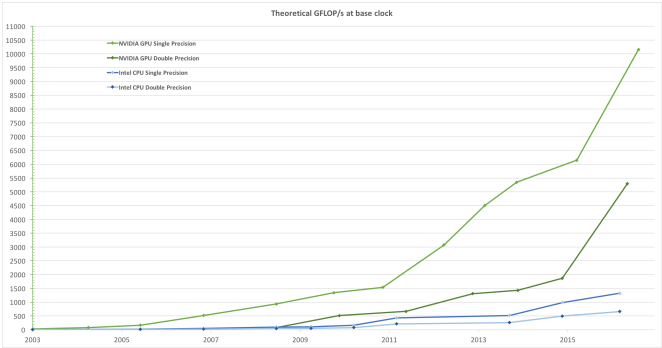


图 9: CPU vs GPU 理论计算性能对比

1 并行计算简介

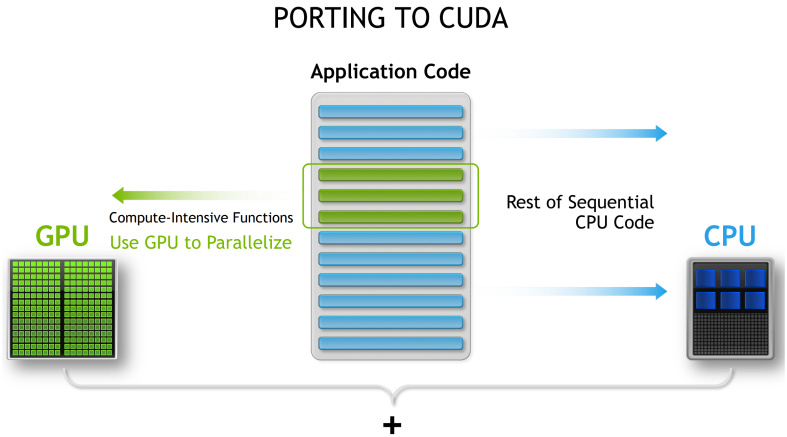
2 GPU 硬件基础

3 GPU 软件编程
CUDA 编程框架
NVIDIA HPC SDK

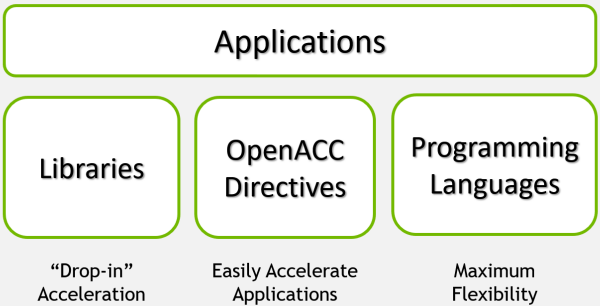
4 小结与展望

5 参考文献

移植思路


PORTING TO CUDA
Application Code
Compute-Intensive Functions
Use GPU to Parallelize
GPU
Rest of Sequential
CPU Code
CPU
+

开发方法


3 Ways to Accelerate Applications
Applications
Libraries
OpenACC Directives
Programming Languages
"Drop-in" Acceleration
Easily Accelerate Applications
Maximum Flexibility
© NVIDIA 2013

1 并行计算简介

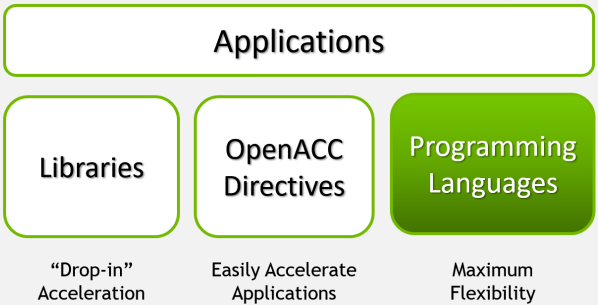
2 GPU 硬件基础

3 GPU 软件编程
CUDA 编程框架
NVIDIA HPC SDK

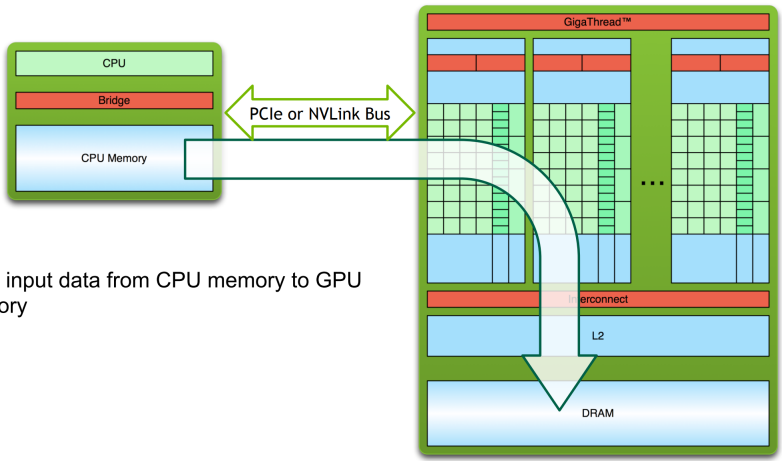
4 小结与展望

5 参考文献

开发方法

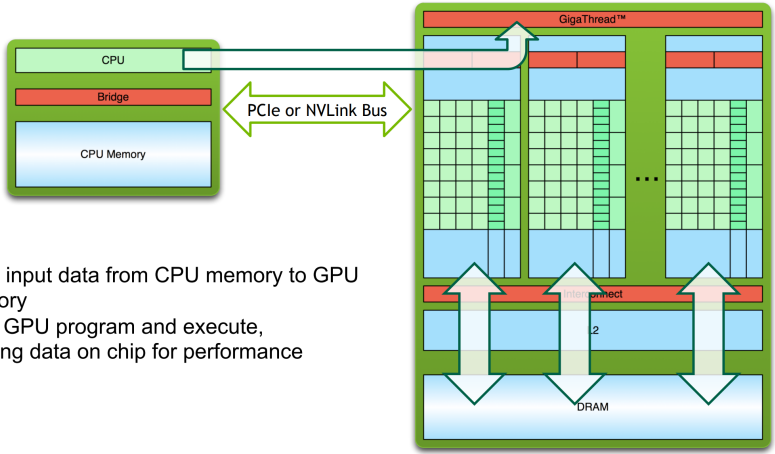

3 Ways to Accelerate Applications
Applications
Libraries
OpenACC Directives
Programming Languages
"Drop-in" Acceleration
Easily Accelerate Applications
Maximum Flexibility
© NVIDIA 2013

通用流程



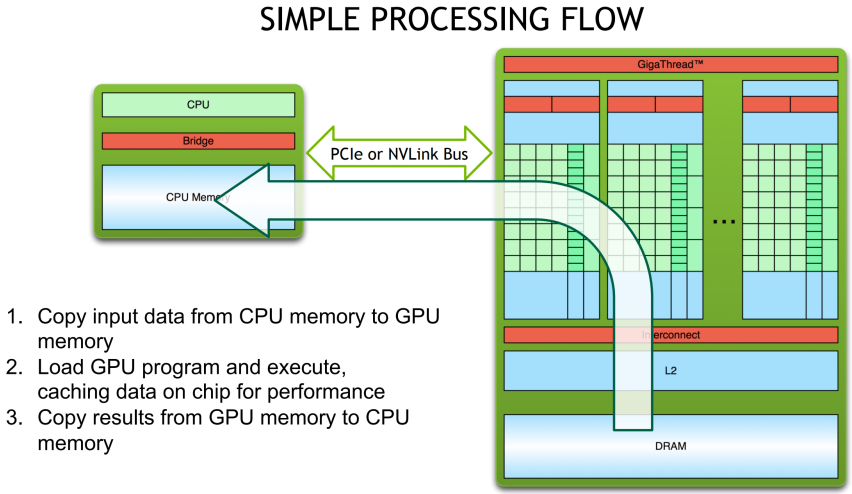
1. Copy input data from CPU memory to GPU memory

通用流程



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

通用流程



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

一个简单的例子

CUDA hello_world 代码

```
1  #include <stdio.h>
2  #include <cuda.h>
3
4  __global__ void hello_world() {
5      printf("Hello, World\n");
6  }
7  int main() {
8      hello_world<<<1,1>>>();
9      return 0;
10 }
```

- cuda.h 头文件
- ___global___ 修饰符：函数从主机端调用，在设备端执行；类似的还有 ___device___，___host___ 等
- <<<, >>> 指定函数运行时线程分配

```
(base) wuchao@USTCNIC[17:23]:~/test$ nvcc hello.cu -o hello
(base) wuchao@USTCNIC[17:23]:~/test$ ./hello
(base) wuchao@USTCNIC[17:23]:~/test$
```

一个简单的例子

- 同步问题：GPU 与 CPU 的协同
- cudaDeviceSynchronize显式同步

CUDA hello_world 代码（修正）

```
1       ...
2  int main() {
3      hello_world<<<1,1>>>();
4      // GPU与CPU同步
5      cudaDeviceSynchronize();
6      return 0;
7  }
```

```
(base) wuchao@USTCNIC[17:24]:~/test$ nvcc hello.cu -o hello
(base) wuchao@USTCNIC[17:24]:~/test$ ./hello
Hello world!
(base) wuchao@USTCNIC[17:24]:~/test$
```

线程并行

- CUDA 核函数被线程阵列的所有线程并行执行 SPMD (single process multiple data)
  - 所有线程运行相同的代码
  - 每个线程根据 ID 决定读取内存的位置，执行的路径，如何保存结果

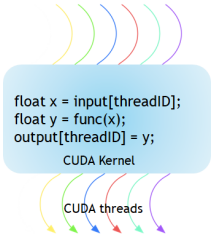
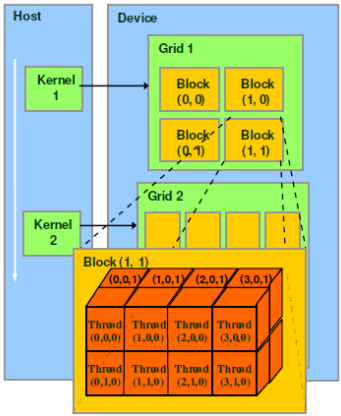


图 10: 核函数执行 [Gup]

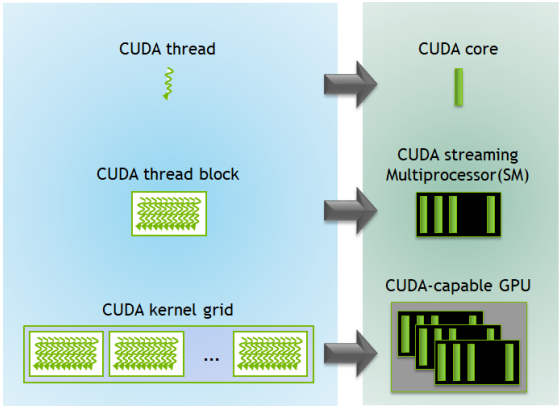
线程组织

- Thread：一个 CUDA 的并行程序会被以许多个 thread 来执行 [三]
- Block：数个 thread 会被群组成一个 block，同一个 block 中的 thread 可以同步，也可以通过 shared memory 进行通信
- Grid：多个 block 构成 grid，线程网格内的线程共享 global memory



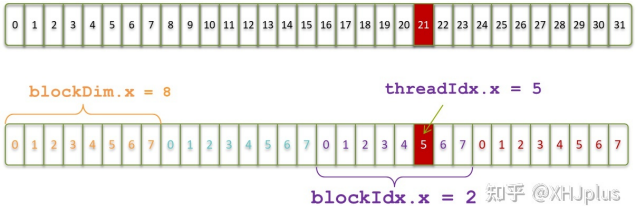
软硬联系

- warp(线程束) 是最基本的执行单元，一个 warp 包含 32 个并行 thread，这些 thread 以不同数据资源执行相同的指令 [三]
- 当一个 kernel 被执行时，grid 中的线程块被分配到 SM 上，一个线程块的 thread 只能在一个 SM 上调度，SM 一般可以调度多个线程块
- 一个 CUDA core 可以执行一个 thread，一个 SM 的 CUDA core 会分成几个 warp（即 CUDA core 在 SM 中分组），由 warp scheduler 负责调度



索引计算

- CUDA 中每个线程都可以计算出全局索引
- 内建 threadIdx, blockDim, blockIdx, gridDim 用于索引计算
- 全局及局部索引用于解决“我是谁、我在哪、我要干啥”
- KernelFunction $<<<4,8>>>$ 线程索引计算




```
int index = threadIdx.x + blockIdx.x * blockDim.x;
          =       5     +      2      * 8;
          = 21;
```

索引计算

- 一维 grid，一维 block

```
int idx = blockIdx.x * blockDim.x + threadIdx.x;
```

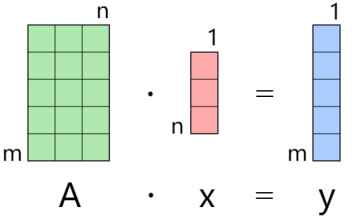
- 一维 grid，二维 block

```
int idx = blockIdx.x * blockDim.x * blockDim.y
 + threadIdx.y * blockDim.x + threadIdx.x;
```

- 二维 grid，二维 block

```
int blockId = blockIdx.x + blockIdx.y * gridDim.x;
int idx = blockId * (blockDim.x * blockDim.y)
  + (threadIdx.y * blockDim.x) + threadIdx.x;
```

另一个简单的例子



另一个简单的例子

CUDA naive gemv 代码

```
__global__ void gemvKernel(float *a, float *b, float *c, int n) {
    int row=threadIdx.x+blockDim.x*blockIdx.x;
    float sum=0;

    if(row<n) {
        for(int j=0;j<n;j++) {
            sum=sum+a[row*n+j]*b[j];
        }
    }
    c[row]=sum;
}
```

另一个简单的例子

main 函数调用 CUDA naive gemv

```
int main() {
                    ...
    cudaMemcpy(d_a, a, sz_a, H2D);
    cudaMemcpy(d_b, b, sz_b, H2D);

    int dimblk = 32;
    int dimgrd = (n + dimblk - 1)/dimblk;

    gemvKernel<<<dimgrd, dimblk>>>(d_a, d_b, d_c, n);

    cudaMemcpy(c, d_c, sz_c, D2H);
                    ...
}
```

1 并行计算简介

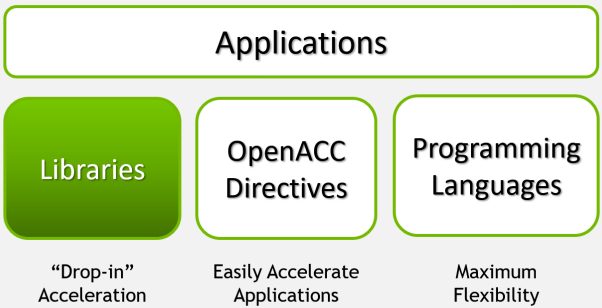
2 GPU 硬件基础

3 GPU 软件编程
CUDA 编程框架
NVIDIA HPC SDK

4 小结与展望

5 参考文献

开发方法


3 Ways to Accelerate Applications
Applications
Libraries
OpenACC Directives
Programming Languages
"Drop-in" Acceleration
Easily Accelerate Applications
Maximum Flexibility
© NVIDIA 2013

常用数学库 [NVIc]

Math Libraries

GPU-accelerated math libraries lay the foundation for compute-intensive applications in areas such as molecular dynamics, computational fluid dynamics, computational chemistry, medical imaging, and seismic exploration.

cuBLAS

GPU-accelerated basic linear algebra (BLAS) library

Learn More

cuFFT

GPU-accelerated library for Fast Fourier Transforms

Learn More

CUDA Math Library

GPU-accelerated standard mathematical function library

Learn More

cuRAND

GPU-accelerated random number generation (RNG)

Learn More

cuSOLVER

GPU-accelerated dense and sparse direct solvers

Learn More

cuSPARSE

GPU-accelerated BLAS for sparse matrices

Learn More

cuTENSOR

GPU-accelerated tensor linear algebra library

Learn More

AmgX

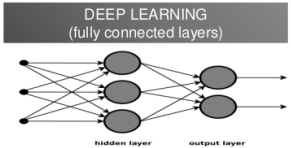
GPU-accelerated linear solvers for simulations and implicit unstructured methods

Learn More

cuBLAS

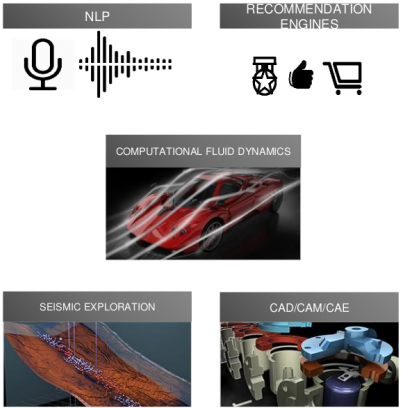
- GPU 加速稠密线性代数库
- 完整支持 BLAS 编程接口
  - 支持 152 个标准函数（包括单精度、双精度、单精度复数、双精度复数版本）
  - 支持半精度（FP16）矩阵、整型（INT8）矩阵和混合精度矩阵乘
  - 支持小规模矩阵批处理计算
  - XT 接口支持多卡分布式计算

```
cublasStatus_t cublasSgemv(cublasHandle_t handle, cublasOperation_t trans,
                           int m, int n,
                           const float           *alpha,
                           const float           *A, int lda,
                           const float           *x, int incx,
                           const float           *beta,
                           float           *y, int incy)
```



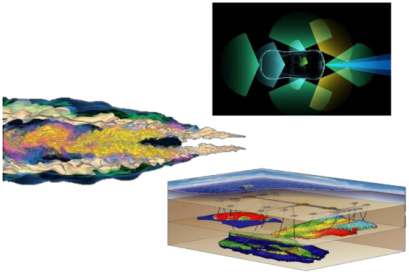
cuSPARSE

- 基于 GPU 优化的稀疏矩阵库
  - 优化的稀疏矩阵计算：矩阵-向量乘法、矩阵-矩阵乘法、三角分解等
  - 支持多种稀疏矩阵存储格式（CSR、COO 等）
  - 支持稀疏矩阵的 LU 和 Cholesky 分解
  - 支持半精度（FP16）稀疏矩阵-向量乘



cuSOLVER

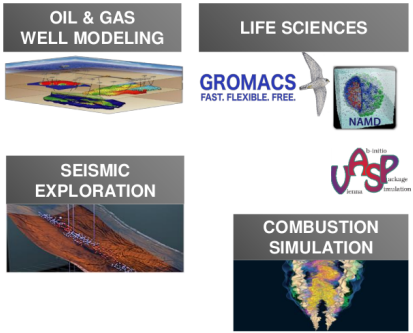
- GPU 线性求解器库
- 支持稠密和稀疏矩阵的直接求解
  - 支持稠密矩阵的 Cholesky 分解、LU 分解、QR 分解、SVD 分解和特征值分解
  - 支持稀疏矩阵的直接分解和特征值分解
  - 应用于大量工业和科学应用中，包括集成电路模拟和计算流体力学等



Sample Applications
- Computer Vision
- CFD
- Newton's method
- Chemical Kinetics
- Chemistry
- ODEs
- Circuit Simulation

cuFFT

- GPU 平台快速傅里叶变换库
  - 对应 CPU 架构的 FFTW 库
  - 支持实数、复数，单双精度多种数据类型
  - 包括 1D、2D、3D 批处理变换
  - 支持半精度（FP16）数据类型
  - 支持灵活的输入输出数据排布
  - XT 支持多卡计算（最多 8 卡）



Some Tips

- 优先使用 Nvidia 公司推出的 GPU 计算库，而不是自行研制
  - 性能
  - 鲁棒性
  - 多卡支持
- 研发中可先调用 CPU 库熟悉接口及用法，后替换成调用相应的 GPU 库
- 留意 GPU 库的计算环境创建的开销

1 并行计算简介

2 GPU 硬件基础

3 GPU 软件编程

4 小结与展望

5 参考文献

小结

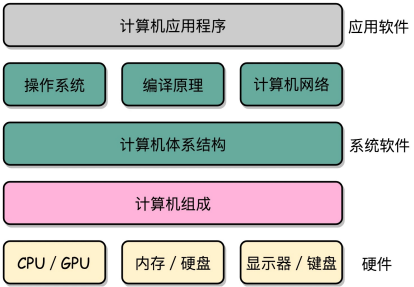


图 11: 并行计算相关基础课程 [LTr]

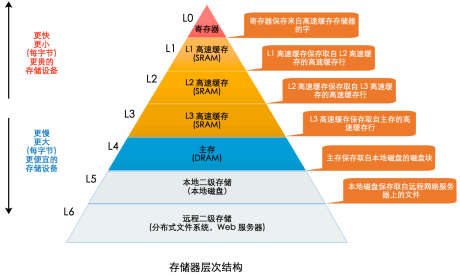
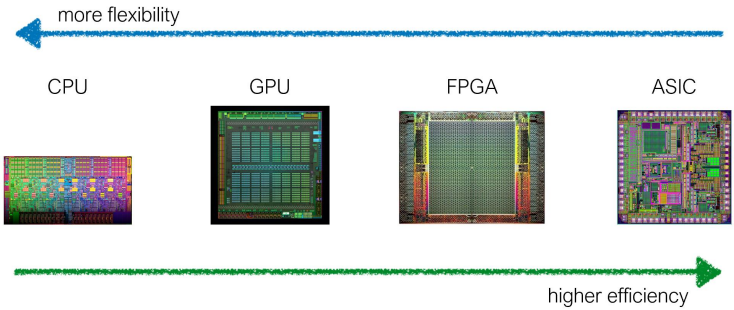


图 12: 存储器层次结构 [黄]

展望



- CPU GPU TPU NPU DPU->? PU
- 芯片国产化（排名不分先后）
  - 老牌: 龙芯 曙光 神威 飞腾
  - 互联网新贵: 寒武纪 华为 阿里

1 并行计算简介

2 GPU 硬件基础

3 GPU 软件编程

4 小结与展望

5 参考文献

参考文献 I

[CGM18] Nishanth Chandran, Durgaprasad Gangodkar, and Ankush Mittal.
A review on gpu programming strategies and recent trends in gpu computing.
2018.

[Cri] Stefania Cristina.
The transformer model.
https://machinelearningmastery.com/the-transformer-model.

[Gup] Pradeep Gupta.
Cuda refresher: The cuda programming model.
https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model.

[kar] karlrupp.
microprocessor-trend-data.
https://github.com/karlrupp/microprocessor-trend-data.

[LAB] OAK RIDGE NATIONAL LABORATORY.
Early frontier users seize exascale advantage, grapple with grand scientific challenges.
https://www.eurekalert.org/news-releases/990054.

[LLNa] LLNL.
Advancing discovery science and innovation.
https://wci.llnl.gov/stockpile-science/high-performance-computing/proprietary-software.

[LLNb] LLNL.
Introduction to parallel computing tutorial.
https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial.

参考文献 II

[LTr] LTracer.
深入浅出计算机组成原理.
https://blog.csdn.net/u013283985/article/details/123712660.

[NVIa] NVIDIA.
Cuda c++ programming guide.
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[NVIb] NVIDIA.
Cuda libraries documentation.
https://docs.nvidia.com/cuda-libraries/index.html.

[NVIc] NVIDIA.
High performance computing hpc sdk.
https://developer.nvidia.com/hpc-sdk.

[the] theislab.
single-cell-tutorial.
https://github.com/theislab/single-cell-tutorial.

[Tri] Ashutosh Tripathi.
What is the main difference between rnn and cnn.
https://ashutoshtripathi.com/2021/07/12/the-main-difference-between-rnn-vs-cnn-nlp.

参考文献 III

[Yel] Kathy Yelick.
Applications of parallel computers.
https://sites.google.com/lbl.gov/cs267-spr2021.

[三] 三七和酒.
理解 cuda 中的 thread,block,grid 和 warp.
https://zhuanlan.zhihu.com/p/123170285.

[中] 中国地震局.
博士生导师简介（王伟涛）.
https://www.cea-igp.ac.cn/yjbw/boshishengdaoshijianjie/279242.html.

[李] 央视记者李峥.
“神威 • 太湖之光”应用成果再次入围“戈登 • 贝尔”奖提名.
http://m.news.cctv.com/2017/06/16/ARTIlqcgYW84pzJGlZHtuj5K170616.shtml.

[陈 11] 陈国良.
并行计算：结构 • 算法 • 编程（第 3 版）.
北京：高等教育出版社, 北京, 2011.

[黄] 黄枫谷.
计算机的存储器系统.
https://huminxi.netlify.app/2019/04/19/

谢谢!