异构计算与 MIC、CUDA 编译使用简介

张文帅 (wszhang@ustc.edu.cn)

中国科学技术大学超级计算中心

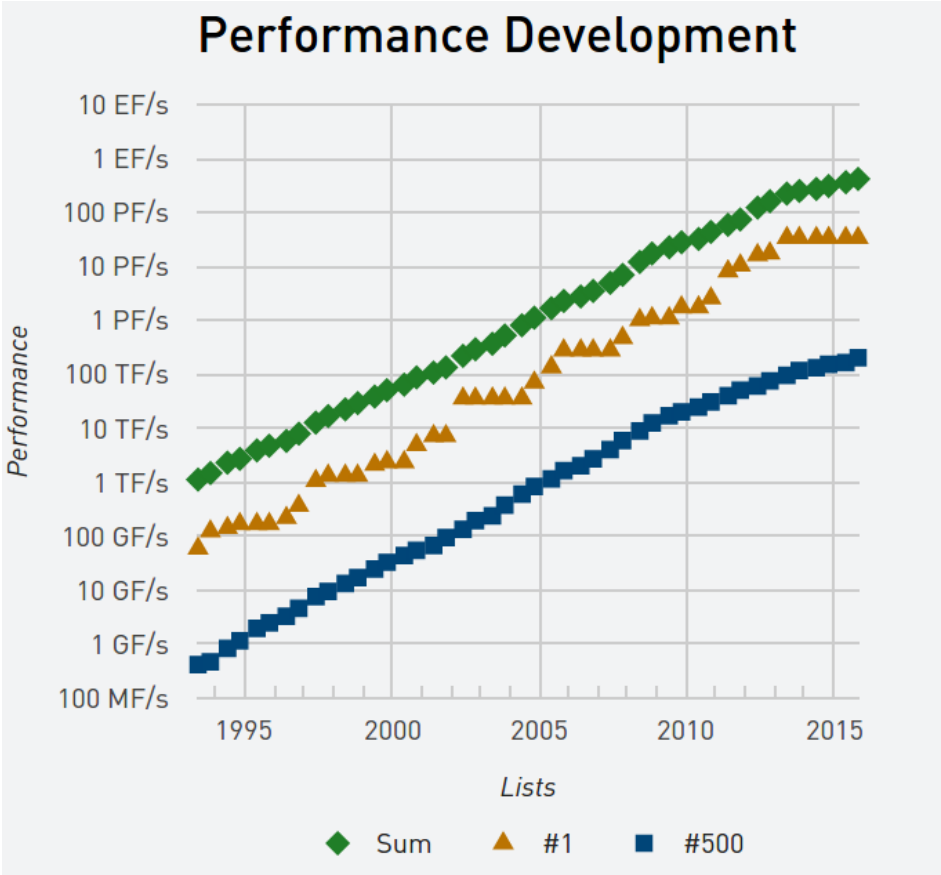
2016 年 10 月 14 日

目录

1. 异构众核计算
   - 并行计算的现状与结构分类
   - 异构计算的实现方式
2. Intel 众核 (MIC) 计算
   - MIC 简介
   - MIC 计算运行方式
3. CUDA 计算与程序编译运行
   - CUDA 计算简介
   - CUDA 编译环境搭建
   - CUDA 程序编译
   - cuBLAS 函数库的使用
   - 作业提交

并行计算现状

众核异构并行计算发展迅速，近期 Top 500 超级计算机中有 104 台使用众核异构加速卡
[66 Nvidia GPU (CUDA) + 29 Intel Xeon Phi (MIC) + 4 Nvidia & Xeon Phi + 3 ATI Radeon + 2 PEZY-SC(1024 cores)]

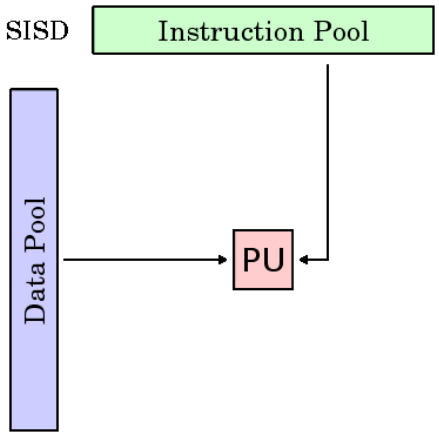


Top500 http://top500.org

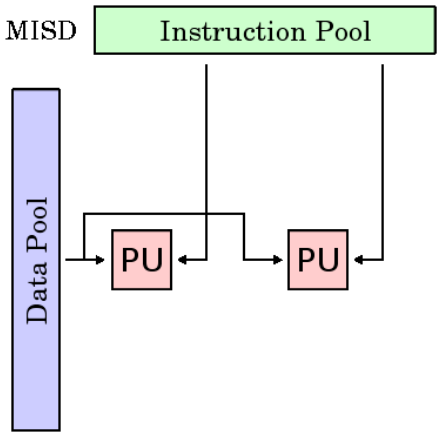
Tianhe-2 包含 16000 节点，每节点 2 Ivy Bridge chips and 3 Xeon Phi chips。

计算体系结构分类

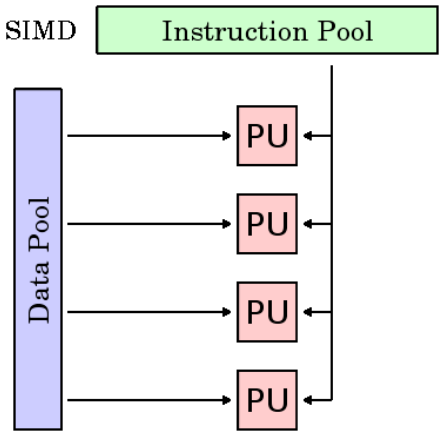
Single Instruction Single Data (SISD)



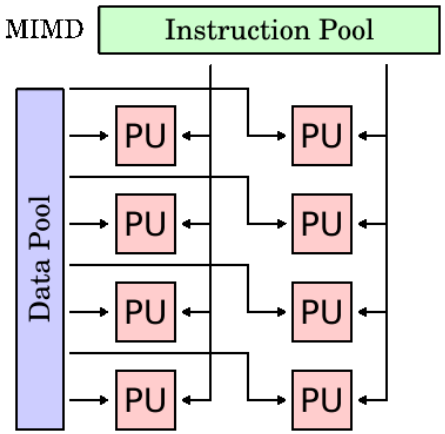
Multiple Instruction Single Data (MISD)



Single Instruction Multiple Data (SIMD)



Multiple Instruction Multiple Data (MIMD)



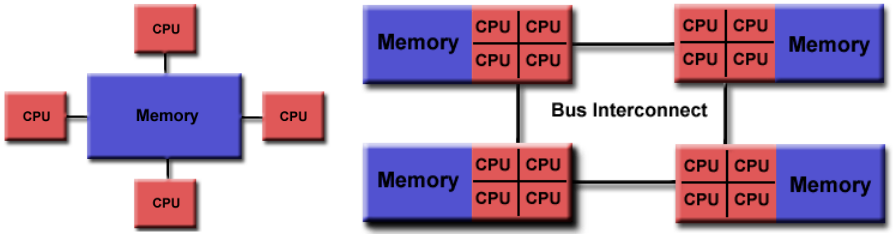
[https://computing.llnl.gov/tutorials/parallel_comp]

异构计算可以较好的实现 MIMD，在 CUDA 编程中，一个 warp(包含 32 线程) 为一个基本调度单元，只能执行同一个指令任务，属于 SIMD; 同时多个不同 block 任务可以在不同的 SM 流多处理器中执行，构成 MIMD。

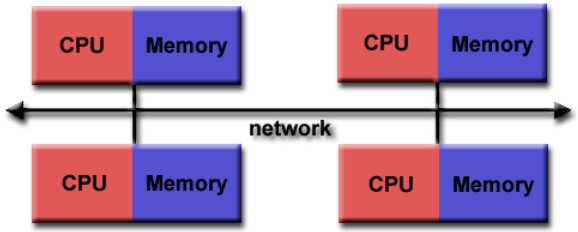
并行计算分类

主流的并行计算框架按照存储方式，可以分成

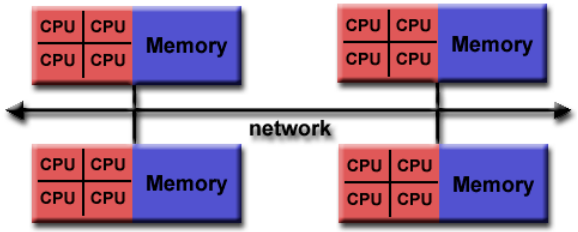
- 共享式存储，单一地址空间 (物理或逻辑)



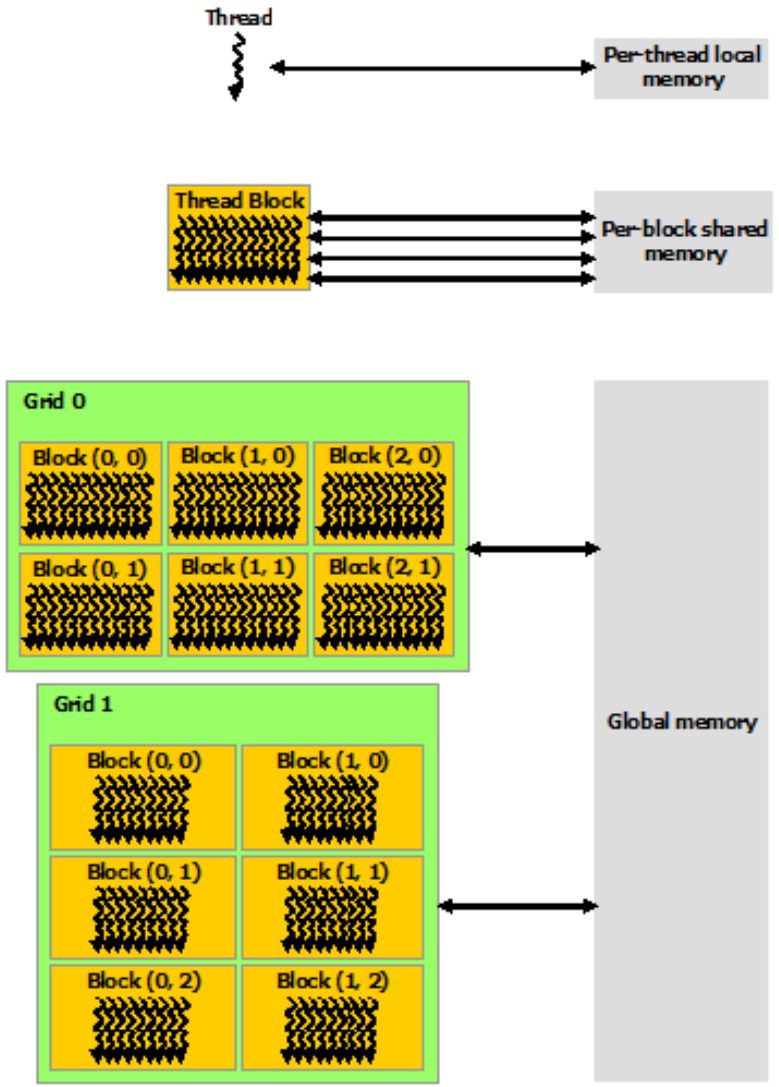
- 分布式存储



- 混合分布式共享存储 (主流)



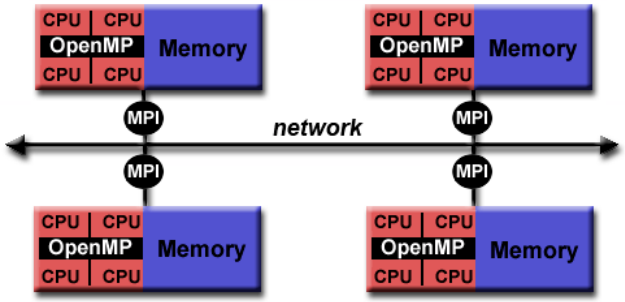
GPU 内存模型



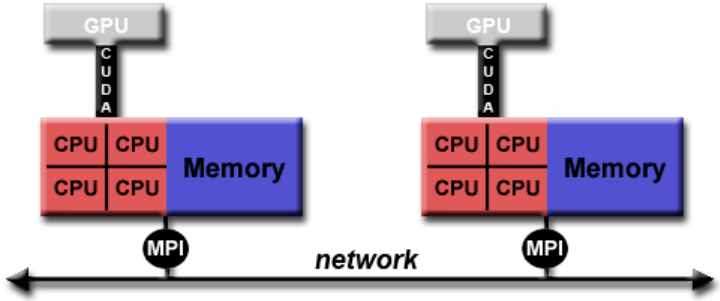
并行计算分类

按照计算机系统的硬件架构组成，区分为

- 通用架构并行
  - 同构多核并行 (CPU)

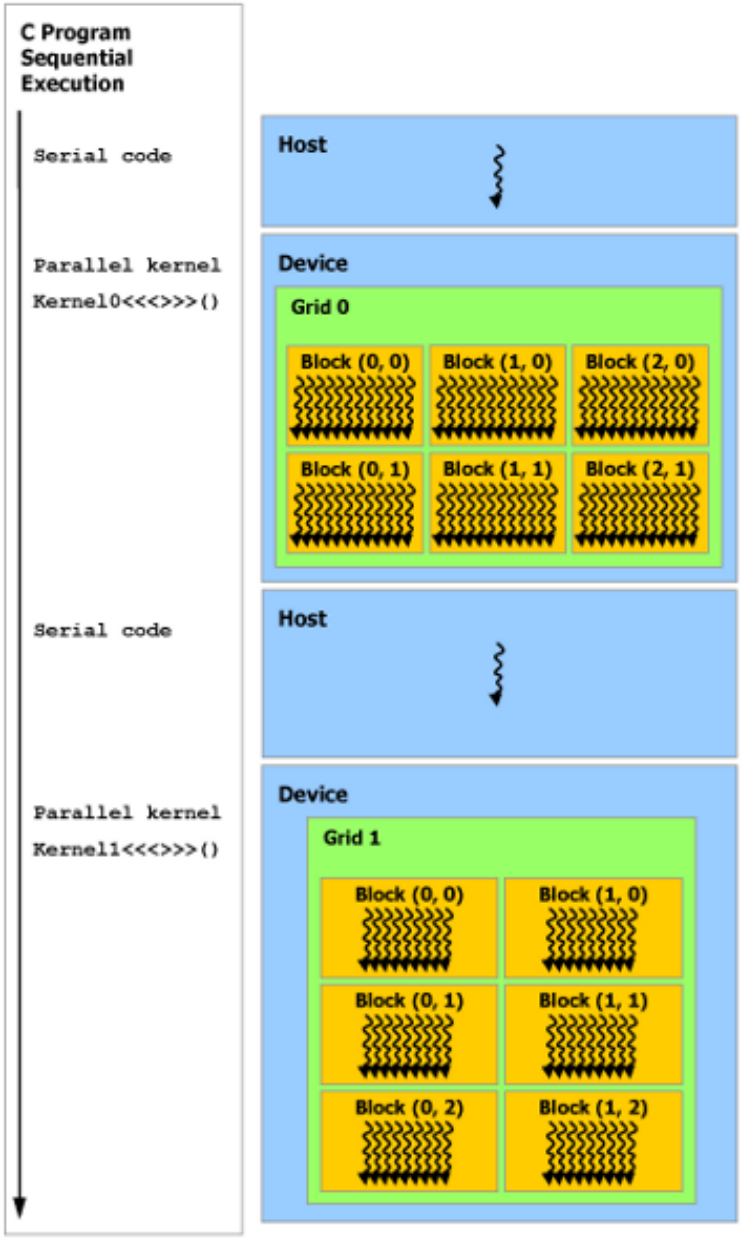


  - 异构众核并行 (CPU+GPU / CPU+MIC)



- 专用架构并行
  - CPU+FPGA 异构

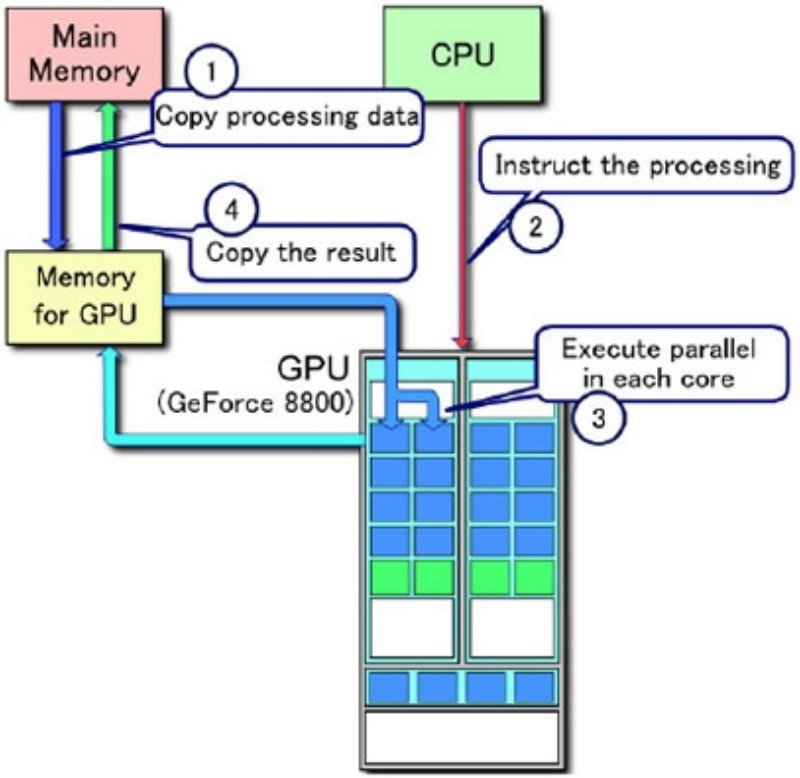
异构计算并行设计 (CUDA 为例)



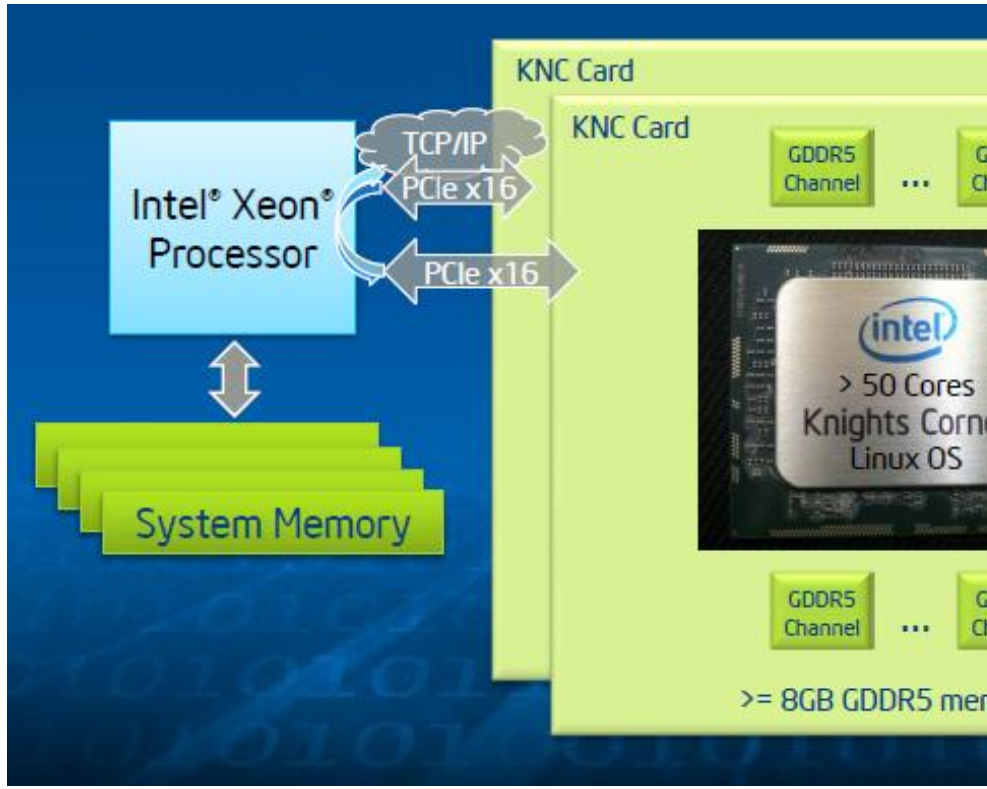
异构计算执行流程

通常异构计算中的加速卡部分执行流程分为四个部分，如 CUDA 程序的执行流程:

CUDA



MIC



对于简单使用编译指导语句的 MIC 计算，在 offload 模式下数据的拷贝操作被隐藏，但依然存在，故而存储带宽经常成为制约速度的主要因素，未来与主机高速共享内存是正在发展的解决方案之一。此外，MIC 程序还具有 native 执行模式，可以登录到卡上 linux 系统执行程序，具有更好的易用性和通用性。

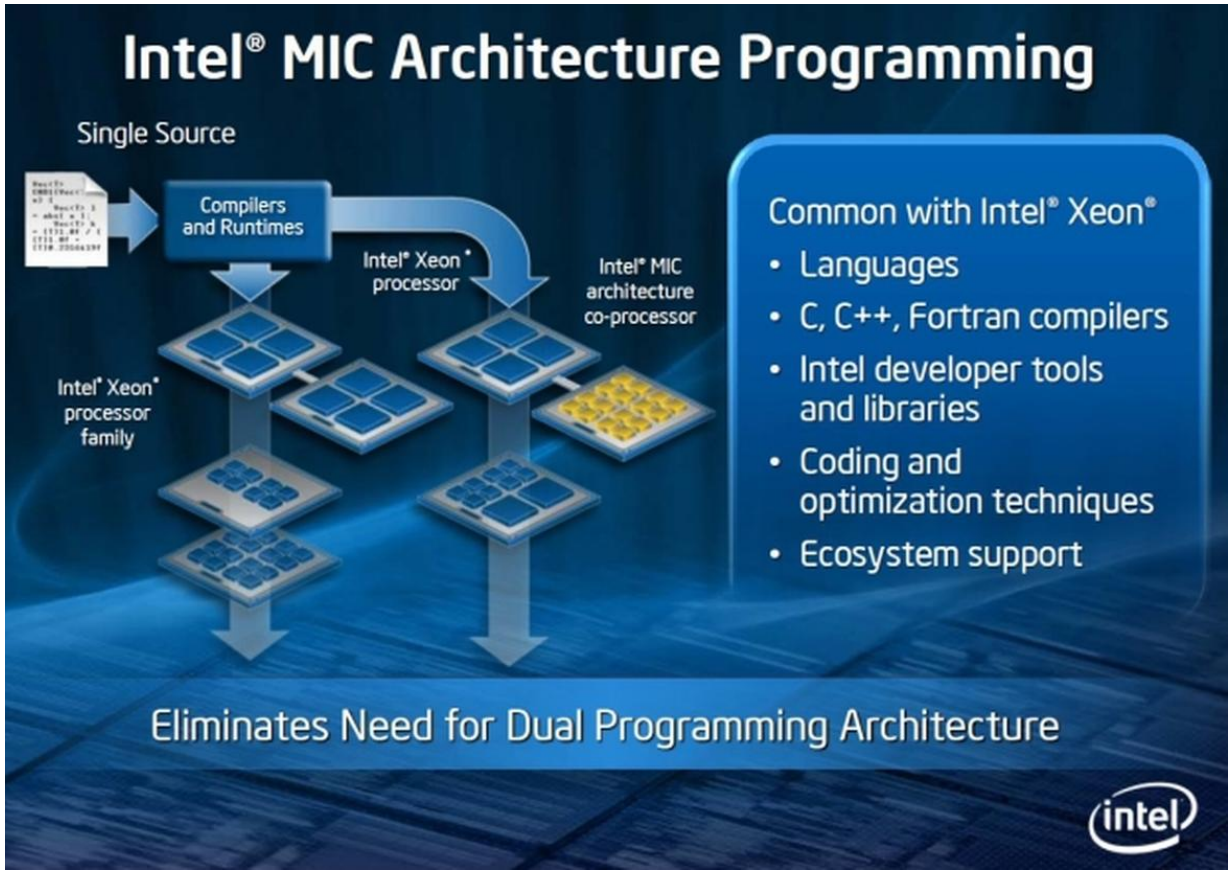
目录

1. 异构众核计算
   - 并行计算的现状与结构分类
   - 异构计算的实现方式
2. Intel 众核 (MIC) 计算
   - MIC 简介
   - MIC 计算运行方式
3. CUDA 计算与程序编译运行
   - CUDA 计算简介
   - CUDA 编译环境搭建
   - CUDA 程序编译
   - cuBLAS 函数库的使用
   - 作业提交

MIC 简介

Mang Intergrated Core，译名：集成众核。
目前已经发展了两代：

- 2013 年发布 Knights Corner （KNC）
- 2016 年发布 Knight Landing（KNL）



第一代 MIC：KNC

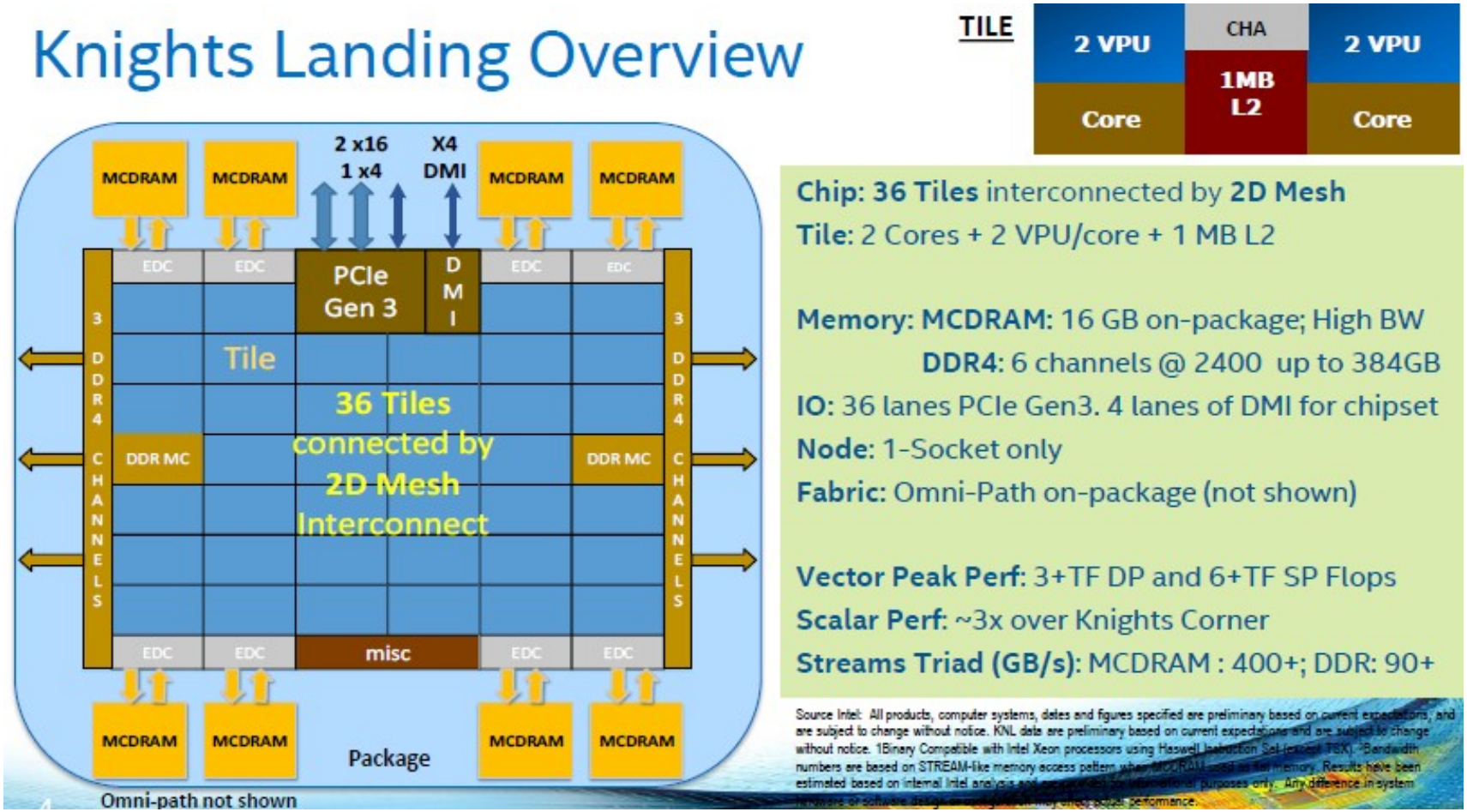
目前超算中心具有 4 个 KNC 加速卡，对比 CPU 指标为：

表 1: CPU和MIC的指标对比

Intel Xeon	CPU	MIC
型号	Xeon E5-2650v2	Xeon Phi 7110P
核心数目	8	61
线程数目	16	244
核心频率	2.6GHz	1.238GHz
L1数据cache	8*32KB	61*32KB
L1指令cache	8*32KB	61*32KB
L2 cache	8*256KB	61*512KB
LLC Cache	20MB	/
内存	外置DDR3	8GB GDDR5
设计最大功耗	95W	300W

MIC 第二代：KNL

相比第一代 KNC 需要协助其他 CPU 完成加速，第二代 KNL 可以作为独立的 CPU 使用。



native 模式

登录到 MIC 卡中，之后操作与 CPU 没有关系，将 MIC 如同 CPU 一般工作运行:

```
[wszhang@node46 ~]$ssh mic0
[wszhang@node46-mic0 ~]$cat /proc/cpuinfo | grep
processor| wc -l
244
```

但是在 MIC 系统中不能编译，编译需要在 CPU 端进行，只需编译时添加 -mmic 参数即可:

```
#icc −mmic −openmp −o pi−omp−native pi−omp.c
```

注意：编译 OpenMP 时需要链接 MIC 版的 libiomp5.so

offload 模式

类似于使用 GPU 作为计算设备的程序, 即程序在 CPU 端启动并运行, 中间遇到需要加速计算的部分则转移到 MIC 上运行, 运行结束后再返回 CPU 端, 不需要登录 MIC 卡即可使用。使用的编程方法类似 OpenMP, 添加一些指定 MIC 作为加速卡的语句如:

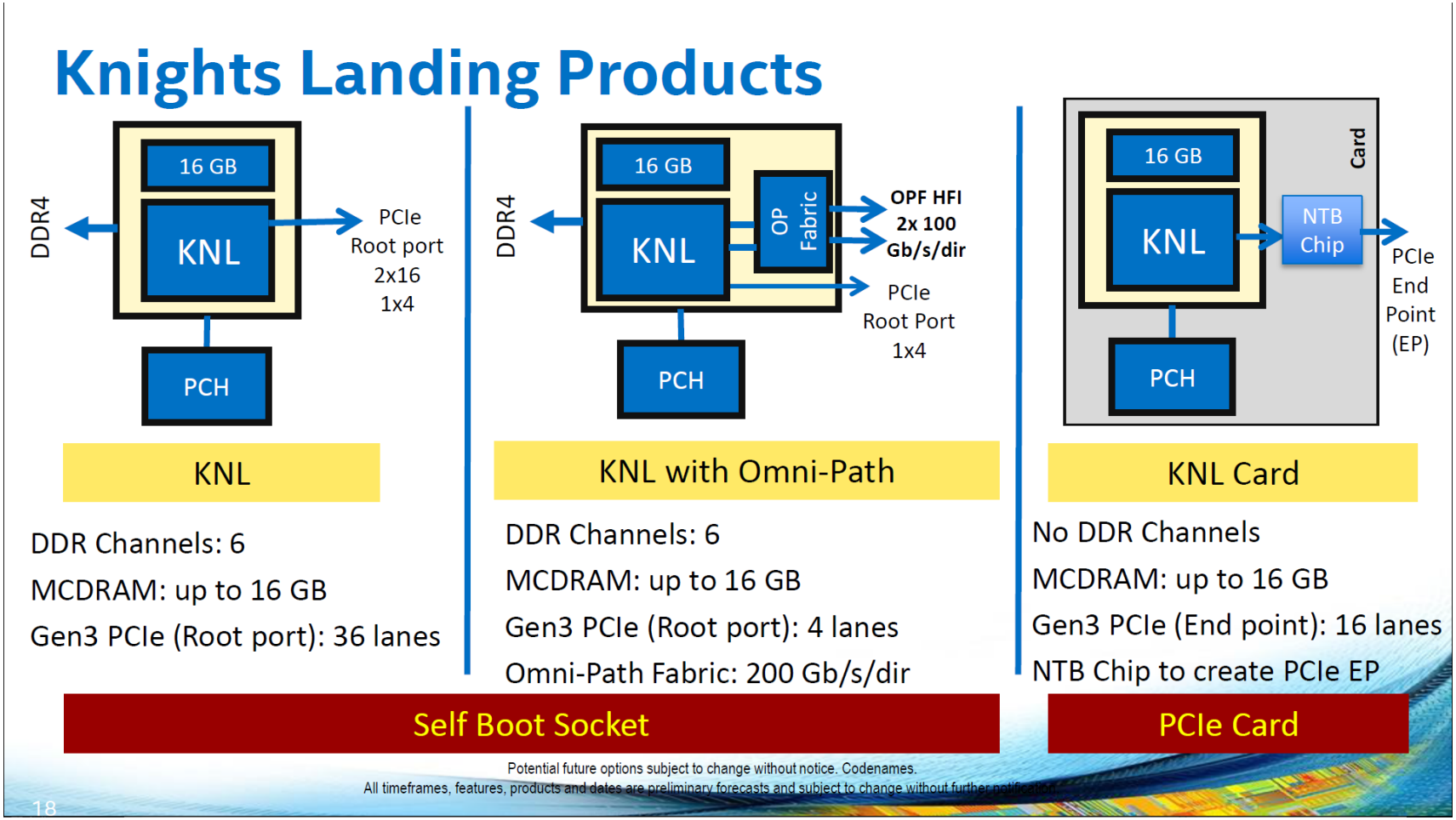
```
#pragma offload target(mic)
```

LSF 作业提交

中科大的 MIC 节点位于 ChinaGrid 集群的 node45 与 node46。提交任务并无不同，利用“bsub −q mic”选项将任务提交至 mic 队列即可，如下:

bsub −q mic −o %J.log −e %J.err executable

KNL 运行方式

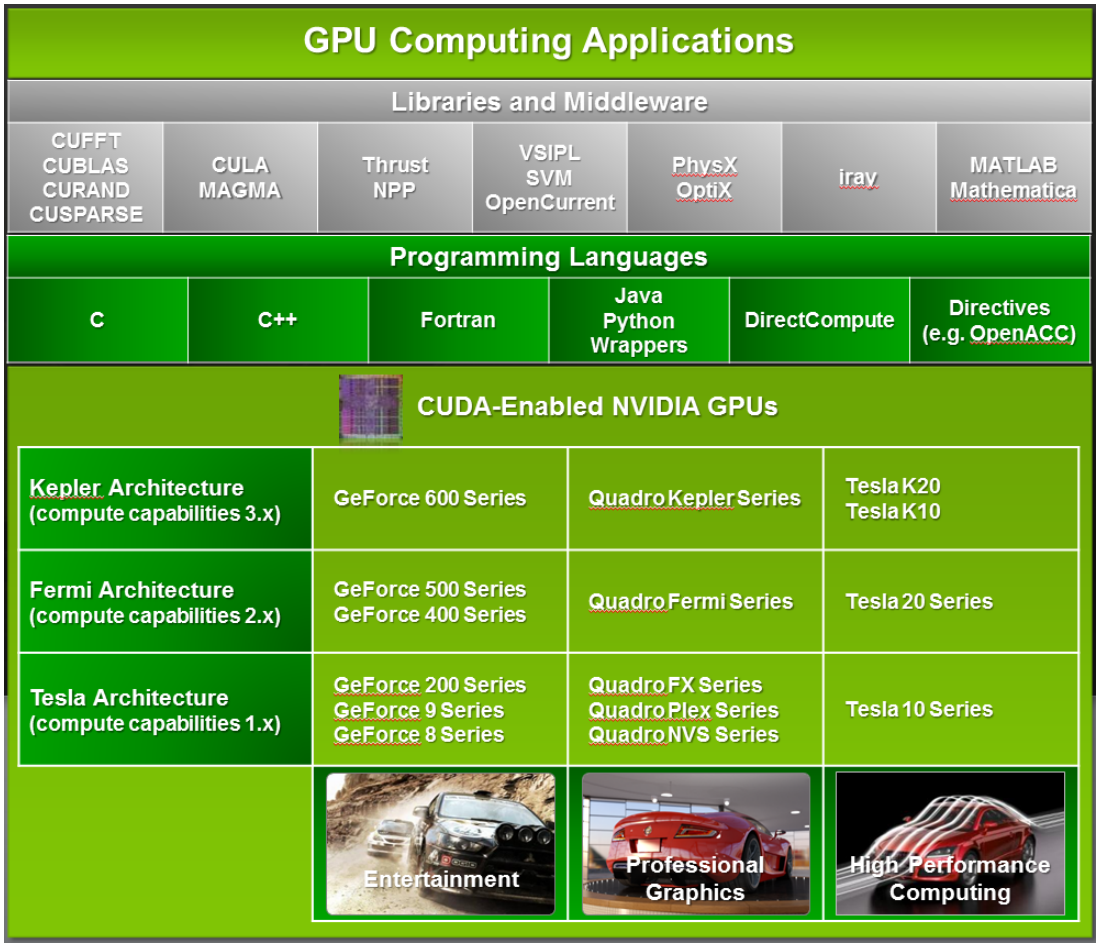

Knights Landing Products
DDR4
16 GB
KNL
PCH
PCIe
Root port
2x16
1x4
KNL
DDR4
16 GB
KNL
PCH
OP Fabric
OPF HFI
2x 100
Gb/s/dir
PCIe
Root Port
1x4
KNL with Omni-Path
Card
16 GB
KNL
PCH
NTB Chip
PCIe
End
Point
(EP)
KNL Card
DDR Channels: 6
MCDRAM: up to 16 GB
Gen3 PCIe (Root port): 36 lanes
DDR Channels: 6
MCDRAM: up to 16 GB
Gen3 PCIe (Root port): 4 lanes
Omni-Path Fabric: 200 Gb/s/dir
No DDR Channels
MCDRAM: up to 16 GB
Gen3 PCIe (End point): 16 lanes
NTB Chip to create PCIe EP
Self Boot Socket
PCIe Card
Potential future options subject to change without notice. Codenames.
All timeframes, features, products and dates are preliminary forecasts and subject to change without further notification.
18

目录

1. 异构众核计算
   - 并行计算的现状与结构分类
   - 异构计算的实现方式
2. Intel 众核 (MIC) 计算
   - MIC 简介
   - MIC 计算运行方式
3. CUDA 计算与程序编译运行
   - CUDA 计算简介
   - CUDA 编译环境搭建
   - CUDA 程序编译
   - cuBLAS 函数库的使用
   - 作业提交

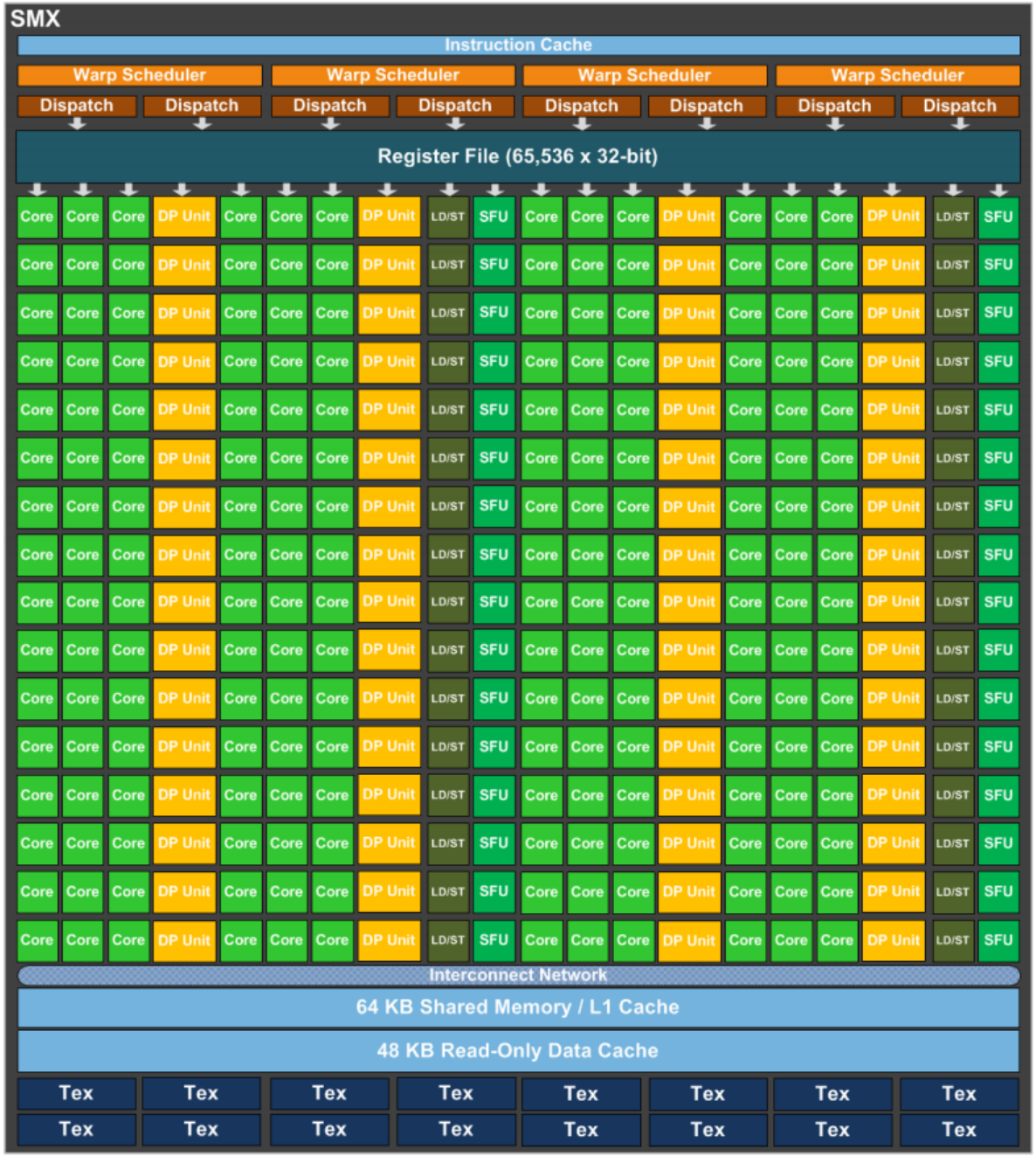
GPU 应用

GPU 应用领域广泛，如教育科研，影视创作，智能机器学习等。在本校专注最多的量子化学领域，也陆续支持 GPU 加速，如主流的 Abinit，Quantum Espress，以及 VASP (Now 2.5 to 4X Faster on Tesla K80)。



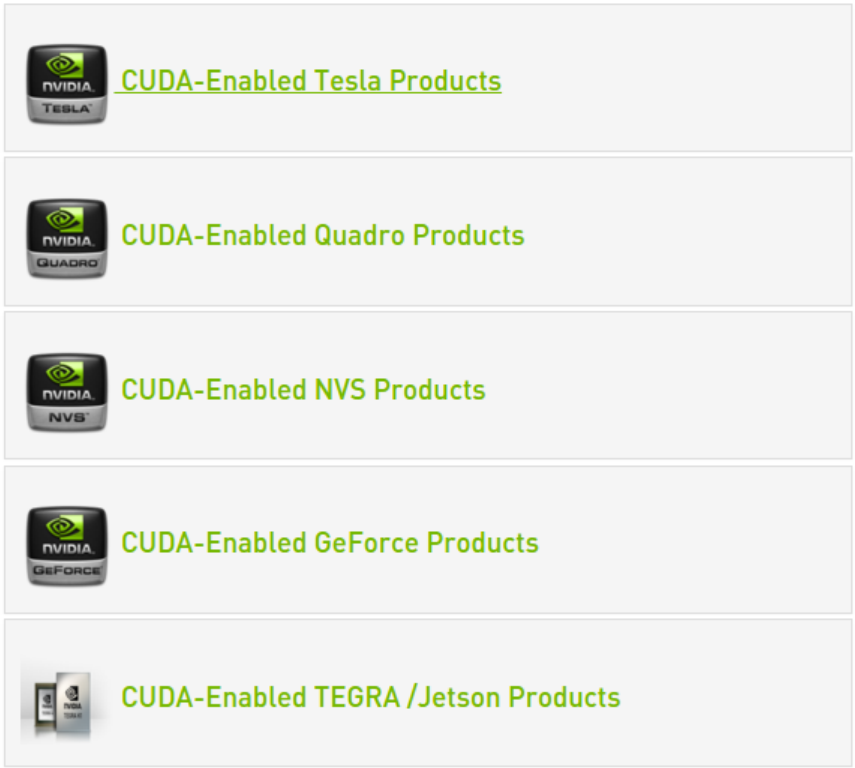
GPU 硬件微架构 (Kepler GK110)

GPU SMX 192 SP and 64 DP units, 32 special function and 32 load/store units



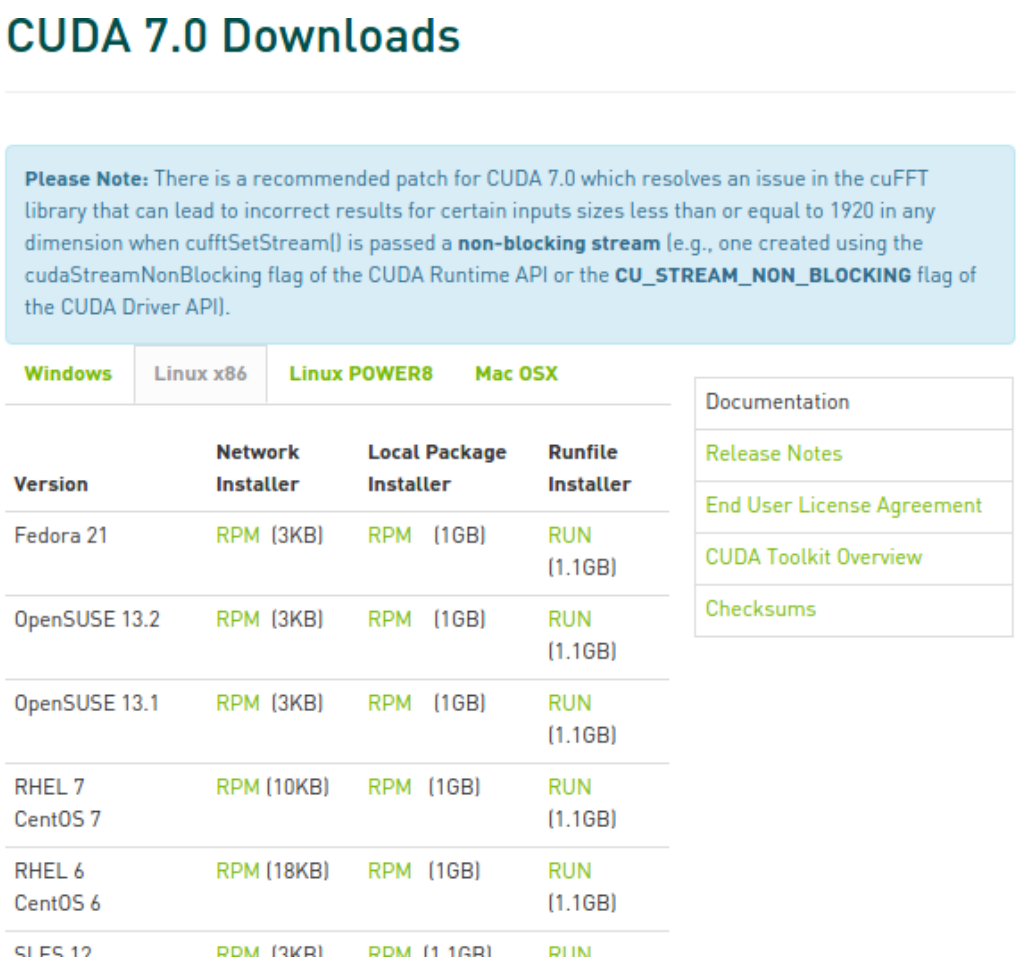
CUDA 安装准备

- 确保具有 root 权限
- 确认已经安装 gcc 编译器
- 确认硬件支持 CUDA：https://developer.nvidia.com/cuda-gpus



CUDA 安装准备

- 确认系统支持 CUDA：https://developer.nvidia.com/cuda-toolkit-archive，选择希望安装的 CUDA 版本，下载自己系统对应的软件包，或者下载通用的.run 后缀的软件包。



CUDA 7.0 Downloads

Please Note: There is a recommended patch for CUDA 7.0 which resolves an issue in the cuFFT library that can lead to incorrect results for certain inputs sizes less than or equal to 1920 in any dimension when cufftSetStream() is passed a **non-blocking stream** (e.g., one created using the cudaStreamNonBlocking flag of the CUDA Runtime API or the **CU_STREAM_NON_BLOCKING** flag of the CUDA Driver API).

Windows | Linux x86 | Linux POWER8 | Mac OSX

Version	Network Installer	Local Package Installer	Runfile Installer
Fedora 21	RPM (3KB)	RPM (1GB)	RUN (1.1GB)
OpenSUSE 13.2	RPM (3KB)	RPM (1GB)	RUN (1.1GB)
OpenSUSE 13.1	RPM (3KB)	RPM (1GB)	RUN (1.1GB)
RHEL 7 CentOS 7	RPM (10KB)	RPM (1GB)	RUN (1.1GB)
RHEL 6 CentOS 6	RPM (18KB)	RPM (1GB)	RUN (1.1GB)
SLES 12	RPM (3KB)	RPM (1.1GB)	RUN

Documentation
Release Notes
End User License Agreement
CUDA Toolkit Overview
Checksums

CUDA 安装

1. 禁用图形显示界面
   - # service lightdm stop
     或
   - # init 3
2. 运行安装程序
   - # sh cuda_7.0.28_linux.run
3. 按提示选择安装组件，并设置安装路径
   - 可以全部 yes，并默认路径

CUDA 环境变量

1. 配置 PATH
   - # export PATH=$PATH:/path-to-cuda/bin
2. 配置 LD_LIBRARY_PATH
   - # export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path-to-cuda/lib64
   - # export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path-to-cuda/lib64/stubs (含-lcuda 库)

CUDA 组件

如果选择通用软件包，并执行

```
./cuda_7.0.28_linux.run -extract=~/Path
```

可以看到会产生三个 CUDA 安装组件：

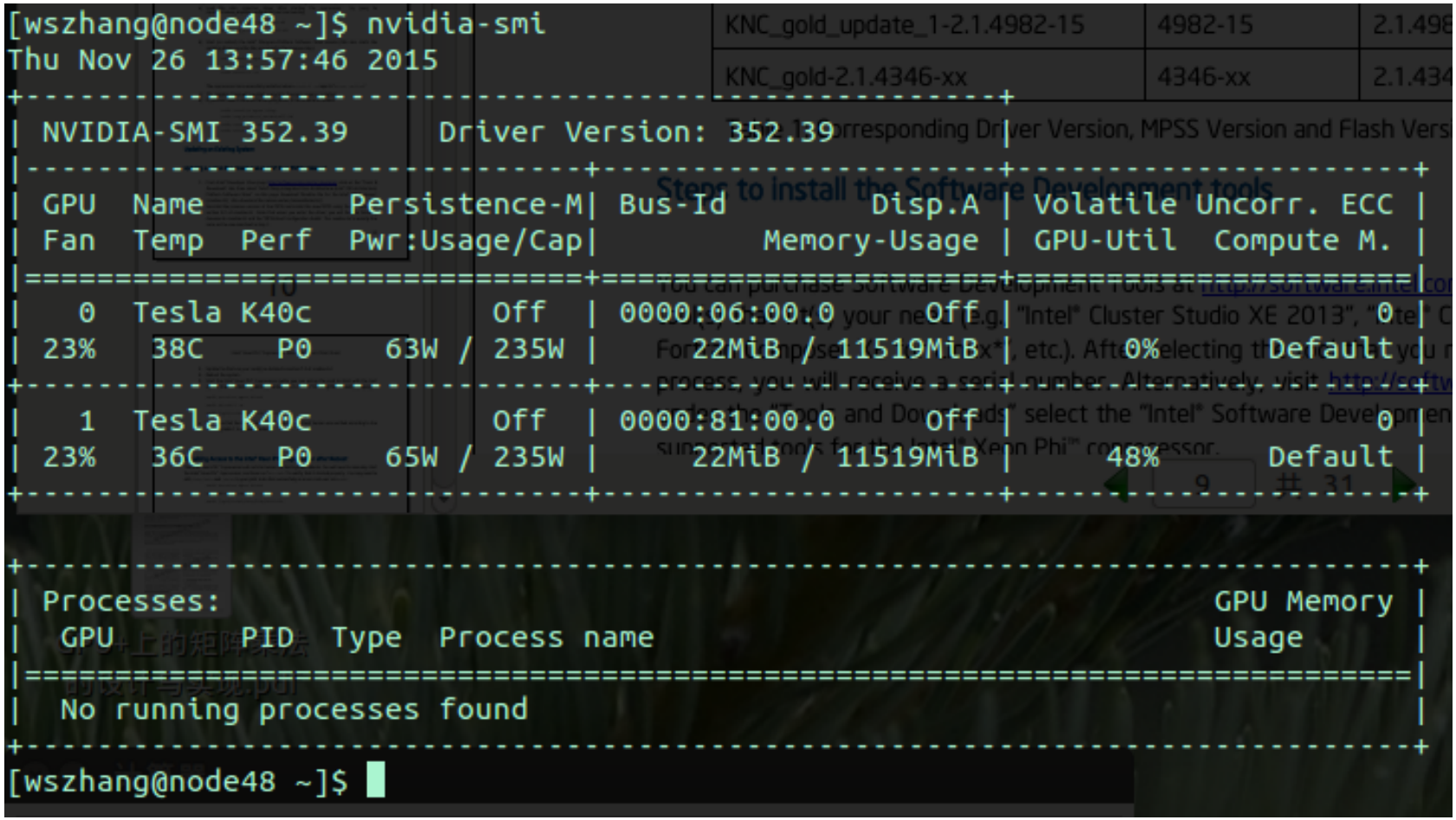
- 设备驱动
  NVIDIA-Linux-x86_64-346.46.run
- CUDA Toolkit 程序开发包
  cuda-linux64-rel-7.0.28-19326674.run
- 程序开发事例包
  cuda-samples-linux-7.0.28-19326674.run

CUDA Toolkit 组件

- Devlopment Tools
  - NVCC，PTXAS，cuobjdump，Nsight Eclipse
- Libraries
  - cuBLAS，cuFFT，cuRAND，cuSPARES, NPP 等
- Tools
  - CUDA-GDB，CUDA-MEMCHECK，Visual Profiler，NVIDIA-SMI，NVML

nvidia-smi

使用此命令可以查看显卡驱动与运行状态，详细参数: nvidia-smi -a



Occupancy Calculator

CUDA 占用率计算器可以计算在某个 CUDA 内核下 GPU 中多处理器的占用情况，即活动 Warp 数与 GPU 支持的 Warp 最大数比率。

Just follow steps 1, 2, and 3 below! (or click here for help)

1.) Select Compute Capability (click):	5.3	(Help)
1.b) Select Shared Memory Size Config (bytes)		
1.c) Select Global Load Caching Mode	L2 only (cg)	

2.) Enter your resource usage:		
Threads Per Block	256	(Help)
Registers Per Thread	48	
Shared Memory Per Block (bytes)	4096	

(Don't edit anything below this line)

3.) GPU Occupancy Data is displayed here and in the graphs:		
Active Threads per Multiprocessor	1024	(Help)
Active Warps per Multiprocessor	32	
Active Thread Blocks per Multiprocessor	4	
Occupancy of each Multiprocessor	50%	

Physical Limits for GPU Compute Capability:	5.3
Threads per Warp	32
Max Warps per Multiprocessor	64
Max Thread Blocks per Multiprocessor	32
Max Threads per Multiprocessor	2048
Maximum Thread Block Size	1024
Registers per Multiprocessor	65536
Max Registers per Thread Block	32768
Max Registers per Thread	255
Shared Memory per Multiprocessor (bytes)	32768
Max Shared Memory per Block	32768
Register allocation unit size	256
Register allocation granularity	warp
Shared Memory allocation unit size	256
Warp allocation granularity	4

Allocated Resources	Per Block	Limit Per SM	= Allocatable Blocks Per SM
Warps (Threads Per Block / Threads Per Warp)	8	64	8
Registers (Warp limit per SM due to per-warp reg coun	8	20	4
Shared Memory (Bytes)	4096	32768	8

Note: SM is an abbreviation for (Streaming) Multiprocessor

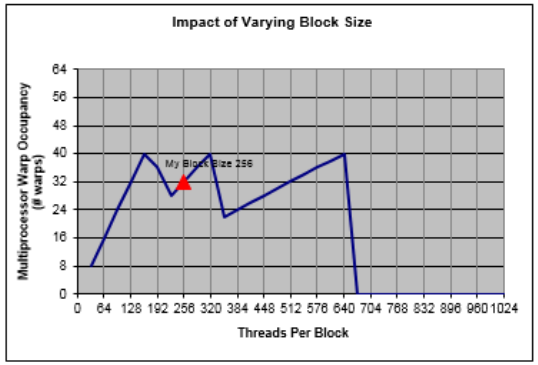
Maximum Thread Blocks Per Multiprocessor	Blocks/SM	* Warps/Block	= Warps/SM
Limited by Max Warps or Max Blocks per Multiprocessor	8		
Limited by Registers per Multiprocessor	4	8	32
Limited by Shared Memory per Multiprocessor	8		

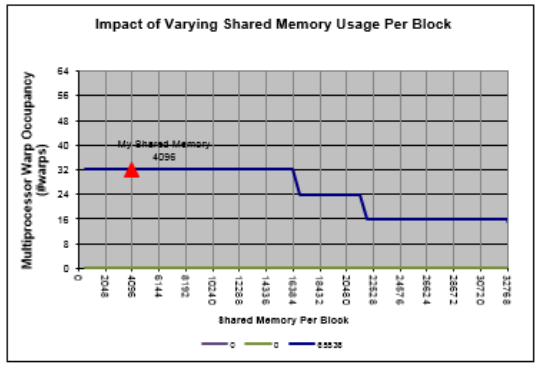
Note: Occupancy limiter is shown in orange

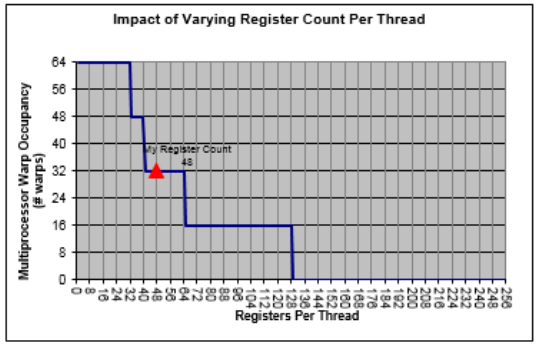
Physical Max Warps/SM = 64
Occupancy = 32 / 64 = 50%

CUDA Occupancy Calculator	
Version:	7.5
Copyright and License	

Your chosen resource usage is indicated by the red triangle on the graphs. The other data points represent the range of possible block sizes, register counts, and shared memory allocation.

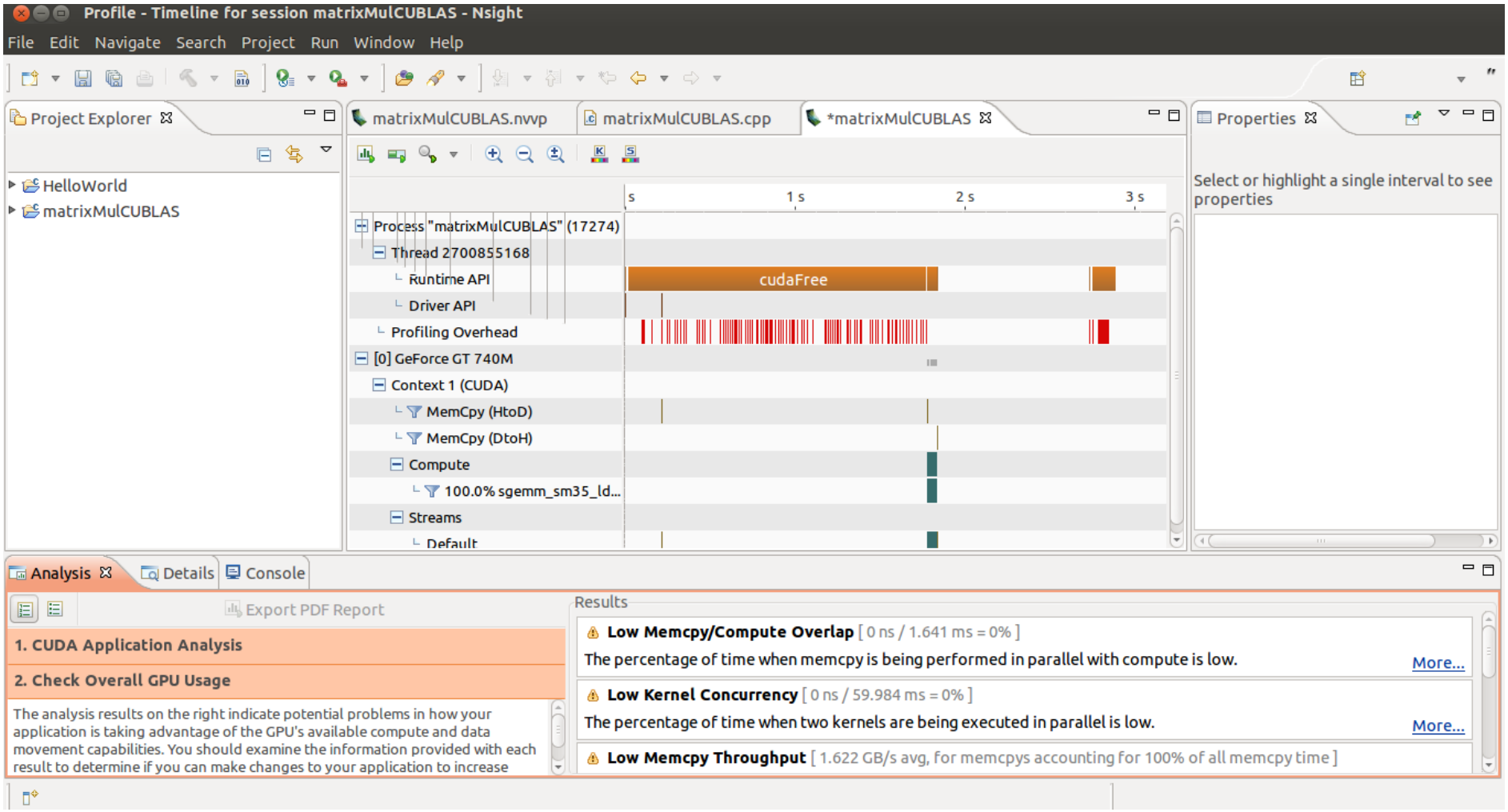






IDE : nsight

集成开发界面



CUDA 编译器 nvcc 输入文件

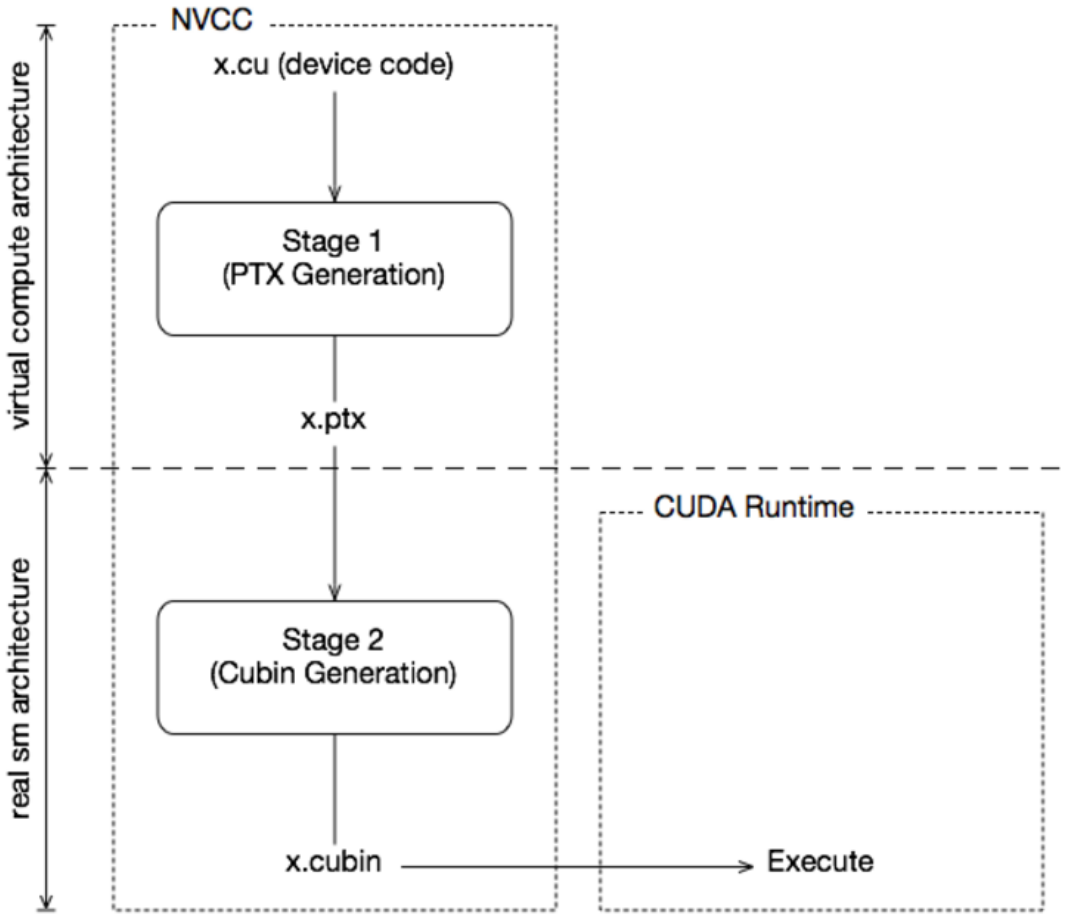
Input File Prefix	Description
.cu	CUDA source file, containing host code and device functions
.c	C source file
.cc, .cxx, .cpp	C++ source file
.gpu	GPU intermediate file
.ptx	PTX intermediate assembly file
.o, .obj	Object file
.a, .lib	Library file
.so	Shared object file

CUDA 编译选项

Phase	short nvcc Option	Default Output File Name
CUDA compilation to C/C++ source file	-cuda	.cpp.ii appended to source file name, as in x.cu.cpp.ii.
C/C++ preprocessing	-E	<result on standard output>
C/C++ compilation to object file	-c	suffix replaced by o on Linux/Mac, obj on Win
Cubin from CUDA source files	-cubin	Source file name with suffix replaced by cubin
Cubin from .gpu intermediate files	-cubin	Source file name with suffix replaced by cubin
Cubin from PTX intermediate files.	-cubin	Source file name with suffix replaced by cubin
PTX from CUDA source files	-ptx	Source file name with suffix replaced by ptx
PTX from .gpu intermediate files	-ptx	Source file name with suffix replaced by ptx
Fatbinary from source, PTX or cubin files	-fatbin	Source file name with suffix replaced by fatbin
GPU C code from CUDA source files	-gpu	Source file name with suffix replaced by gpu
Linking relocatable device code.	-dlink	a_dlink.obj on Win or a_dlink.o on other platforms
Cubin from linked relocatable device code.	-dlink-cubin	a_dlink.cubin
Fatbinary from linked relocatable device code	-dlink-fatbin	a_dlink.fatbin
Linking an executable	<no phase option>	
Constructing an object file archive, or library	-lib	a.lib on Windows or a.a on other platforms
make dependency generation	-M	<result on standard output>
Running an executable	–run	

CUDA 两步编译

为了程序兼容性，CUDA 编译被设计为两步，先在虚拟机架构下编译成类似汇编代码的 PTX 中间文件，而后在第二步中编译成最后的执行文件，PTX 中间件也可以运行时动态的再编译并运行。

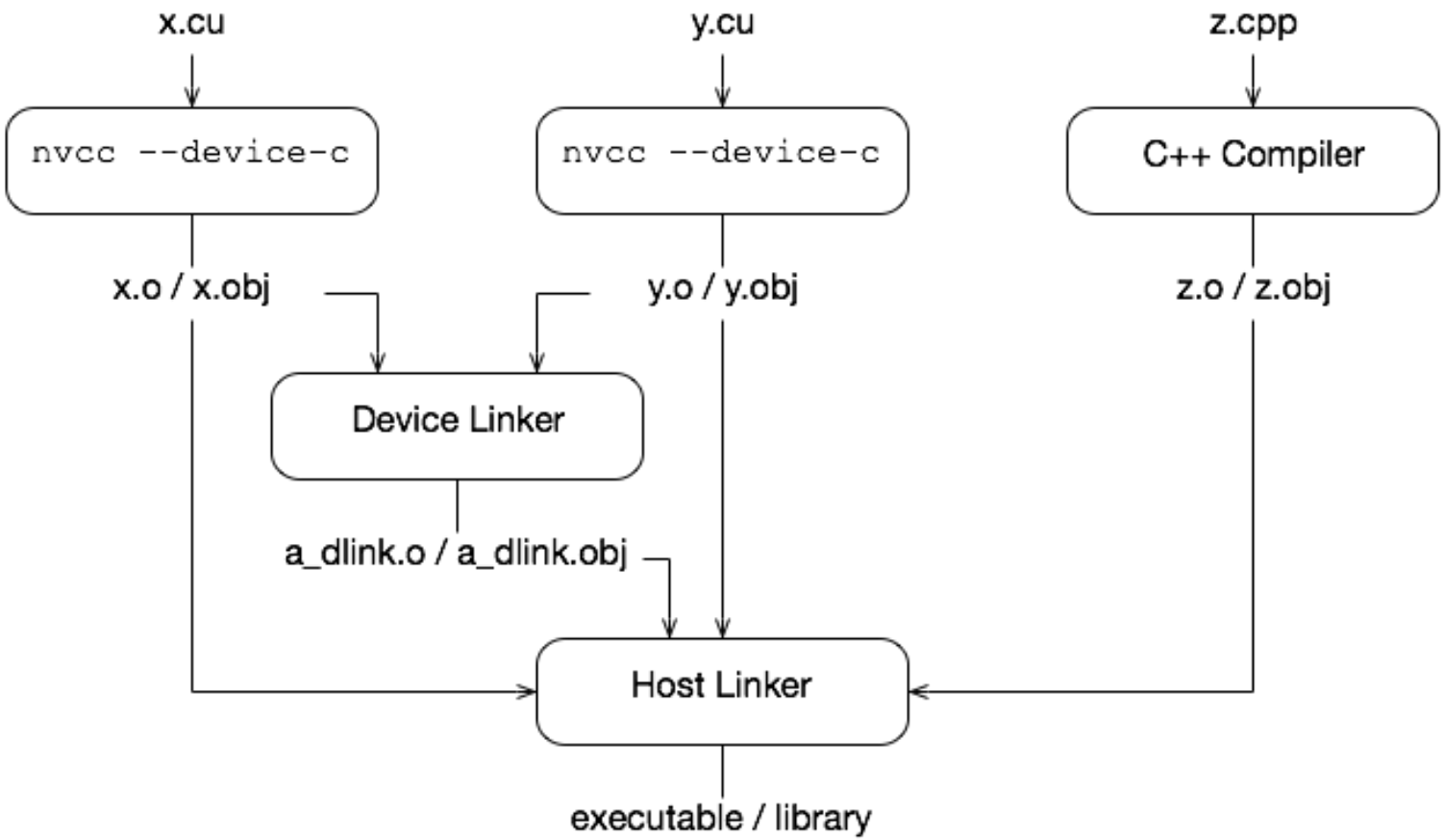


CUDA 中间虚拟架构

	Architecture	Feature
真实架构	sm_20	Basic features + Fermi support
	sm_30 and sm_32	+ Kepler support + Unified memory programming
	sm_35	+ Dynamic parallelism support
	sm_50, sm_52, and sm_53	+ Maxwell support
虚拟架构	compute_20	Basic features + Fermi support
	compute_30 and compute_32	+ Kepler support + Unified memory programming
	compute_35	+ Dynamic parallelism support
	compute_50, compute_52, and compute_53	+ Maxwell support

- –gpu-architecture (可简写为 -arch) arch
  - Specify the name of the class of NVIDIA virtual GPU architecture for which the CUDA input files must be compiled.
- –gpu-code (可简写为 -code) code
  - Specify the name of the NVIDIA GPU to assemble and optimize PTX for.

CUDA C/C++ 可以分别独立编译



CUDA 常用编译方法

一般 cuda 程序编译，虚拟架构版本号需小于等于真实架构版本号

```
nvcc x.cu –gpu-architecture=compute_20 –gpu-code=sm_20
```

多种架构编译

```
nvcc x.cu –gpu-architecture=compute_30 –gpu-code=compute_30,sm_30,sm_35
```

缩写

```
nvcc x.cu –gpu-architecture=sm_35
nvcc x.cu –gpu-architecture=compute_30
```

分别等价于

```
nvcc x.cu –gpu-architecture=compute_35 –gpu-code=sm_35,compute_35
nvcc x.cu –gpu-architecture=compute_30 –gpu-code=compute_30
```

CUDA，C/C++ 分别编译后链接

```
nvcc –gpu-architecture=sm_20 –device-c a.cu b.cu
nvcc –gpu-architecture=sm_20 –device-link a.o b.o –output-file link.o
g++ a.o b.o link.o –library-path=<path> –library=cudart
```

CUDA 程序编译运行 (ThreadBlock.cu) I

```
nvcc -gencode=arch=compute_30,code=sm_30 ThreadBlock.cu -O2 -o ThreadBlock.o
```

```
#include <stdio.h>
//#include <cuda_runtime.h>

__global__ void vadd ( float *a, float *b, float *c ,int nn ) {

  int ii = 3 * blockIdx.x + threadIdx.x;
  //int ii = blockDim.x * blockIdx.x + threadIdx.x;
  //int ii = threadIdx.x ;
  c[ii] = a[ii] + b[ii] ;
}

int main(void) {
  cudaError_t err = cudaSuccess;

  int nn = 6 ;

  size_t size = nn*sizeof(float);
  float *h_a = (float *)malloc(size);
  float *h_b = (float *)malloc(size);
  float *h_c = (float *)malloc(size);

  for (int i=0; i< nn; ++i )
  {
     h_a[i] = rand()/(float)RAND_MAX;
```

CUDA 程序编译运行 (ThreadBlock.cu) II

```
25      h_b[i] = rand()/(float)RAND_MAX;
26    }
27
28    float *d_a = NULL;
29    err = cudaMalloc( (void **)&d_a, size );
30    float *d_b = NULL;
31    err = cudaMalloc( (void **)&d_b, size );
32    float *d_c = NULL;
33    err = cudaMalloc( (void **)&d_c, size );
34
35    err = cudaMemcpy( d_a, h_a, size, cudaMemcpyHostToDevice );
36    if ( err != cudaSuccess )
37    {
38      fprintf(stderr, "no. 1122311 \n") ;
39    }
40
41    err = cudaMemcpy( d_b, h_b, size, cudaMemcpyHostToDevice );
42    if ( err != cudaSuccess )
43    {
44      fprintf(stderr, "no. 1122344 \n") ;
45    }
46
47    vadd<<<2,3>>>(d_a,d_b,d_c, nn);
48
49    err = cudaMemcpy(h_c,d_c,size,cudaMemcpyDeviceToHost);
50
51    for ( int i = 0; i < nn; ++i )
52    {
```

CUDA 程序编译运行 (ThreadBlock.cu) III

```
53      if ( fabs(h_a[i] + h_b[i] - h_c[i] )  > 1e-4   )
54      {
55      fprintf( stderr, " failed at element %d! \n", i );
56      } else
57      {
58      fprintf( stderr, " succeed at element %d! \n", i );
59      }
60    }
61
62    err = cudaFree(d_a);
63    err = cudaFree(d_b);
64    err = cudaFree(d_c);
65
66    free(h_a);
67    free(h_b);
68    free(h_c);
69
70    return 0;
71  }
```

Fortran 编译链接 cuBLAS 函数库 I

源文件

fortran.c #This file contains example Fortran bindings for the CUBLAS library
sgemm_speed.f90

gcc -O3 -DCUBLAS_USE_THUNKING -I/opt/cuda-7.5/include -c fortran.c
ifort -o sgemm_speed_cublas -O3 -fpp -DCUBLAS sgemm_speed.f90 **fortran.o**
-L/opt/cuda-7.5/lib64 **-lcublas**

sgemm_speed.f90

```
1 !
2 ! Simple Fortan90 program that multiplies 2 square matrices calling Sgemm
3 !  C = alpha A*B + beta C
4 !
5 program matrix_multiply
6
7 implicit none
8
9 ! Define the floating point kind to be  single_precision
10 integer, parameter :: fp_kind = kind(0.0)
11
12 ! Define
13 real (fp_kind), dimension(:,:), allocatable ::      A, B, C
```

Fortran 编译链接 cuBLAS 函数库 II

```
14real ::       time_start,time_end
15real (fp_kind)::       alpha=1._fp_kind,beta=1._fp_kind, c_right
16integer::  i,j,m1,m2
17
18
19do m1=512,10240,512
20
21 allocate(A(m1,m1))
22 allocate(B(m1,m1))
23 allocate(C(m1,m1))
24
25 ! Initialize the matrices A,B and C
26 A=1._fp_kind
27 B=2._fp_kind
28 C=3._fp_kind
29
30 ! With the prescribed inputs, each element of the C matrix should be equal to
       c_right
31 c_right= 2._fp_kind*m1+3._fp_kind
32
33! Compute the matrix product  computation
34 call cpu_time(time_start)
35
36#ifdef CUBLAS
37  call cublas_SGEMM ('n','n',m1,m1,m1,alpha,A,m1,B,m1,beta,C,m1)
38#else
39  call SGEMM ('n','n',m1,m1,m1,alpha,A,m1,B,m1,beta,C,m1)
40#endif
```

Fortran 编译链接 cuBLAS 函数库 III

```
41
42 call cpu_time(time_end)
43
44! Print timing information
45 print "(i5,1x,a,1x,f8.4,2x,a,f12.4)", m1, " time =",time_end-time_start, "
       MFLOPS=",1.e-6*2._fp_kind*m1*m1*m1/(time_end-time_start)
46
47! check the result
48     do j=1,m1
49      do i=1,m1
50       if ( abs(c(i,j)- c_right ) .gt. 1.d-8 ) then
51             print *, "sgemm failed", i,j, abs(c(i,j)- c_right )
52         exit
53       end if
54      end do
55     end do
56
57 deallocate(A,B,C)
58end do
59
60end program matrix_multiply
```

C 程序调用 cuBLAS 函数库 I

cublas.h

```
/*—————————————— CUDA ——————————————*/
#ifdef __CUDA

/* Includes, cuda */
#include <cuda_runtime.h>
#include <cublas_v2.h>
#include <host_defines.h>
#include "cuda.h"  // not necessary when compiling with nvcc
#include "cuComplex.h"

#ifdef __single
typedef cuComplex cucomplex;
#define makecucomplex make_cuComplex;
#else
typedef cuDoubleComplex cucomplex;
#define makecucomplex make_cuDoubleComplex;
#endif

    // Error handling macro
#define CUDA_CHECK(call) \
        if((call) != cudaSuccess) { \
            cudaError_t err = cudaGetLastError(); \
            cerr << "CUDA error calling \""#call"\", code is " << err << endl; \
```

C 程序调用 cuBLAS 函数库 II

```
24              my_abort(err); }
25
26 #define cuCHECK(call) \
27          if((call) != cudaSuccess) { \
28              cudaError_t err = cudaGetLastError(); \
29              cerr << "CUDA error calling \""#call"\", code is " << err << endl; \
30              my_abort(err); }
31
32 #define cublacsCHECK(call) \
33          if((call) != CUBLAS_STATUS_SUCCESS) { \
34              cerr << "CUDA error calling \""#call"\"  "  << endl; }
35 #endif
```

C 程序调用 cuBLAS 函数库 I

cublas.cpp

```
1
2#ifdef __CUDA
3
4void cuPAexpm( usrcomplex * h_expMp, usrcomplex * h_Mp, cucomplex * d_expMp,
      cucomplex * d_Mp, cucomplex * d_carray,
5         cucomplex * d_czero, cucomplex * d_cone, cucomplex * d_cngone, const int
              & mval)
6{
7
8             const cuDoubleComplex ddone      =    make_cuDoubleComplex(1.0,0.0);
9             const cuDoubleComplex ddzero     =    make_cuDoubleComplex(0.0,0.0);
10
11            cublasHandle_t handle1;
12            cublasStatus_t status;
13            cublacsCHECK(cublasCreate_v2(&handle1) );
14
15#ifndef __CUDA_NORM1
16            cudaMemcpy(d_Mp  , h_Mp  , MMsize, cudaMemcpyHostToDevice);
17
18#endif
19
20            cucomplex * M2p;
21            cucomplex * M4p;
```

C 程序调用 cuBLAS 函数库 II

```
22          cucomplex * M6p;
23          cucomplex * uM;
24          cucomplex * vM;
25
26
27          cudaMalloc( (void **)&M2p, MMsize);
28          cudaMalloc( (void **)&M4p, MMsize);
29          cudaMalloc( (void **)&M6p, MMsize);
30          cudaMalloc( (void **)&uM,  MMsize);
31          cudaMalloc( (void **)&vM,  MMsize);
32
33          cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                MMdim, MMdim,
34              d_cone, d_Mp, MMdim, d_Mp, MMdim, d_czero, M2p, MMdim) );
35          cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                MMdim, MMdim,
36              d_cone, M2p,  MMdim, M2p,  MMdim, d_czero, M4p, MMdim) );
37          cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                MMdim, MMdim,
38              d_cone, M4p,  MMdim, M2p,  MMdim, d_czero, M6p, MMdim) );
39
40          cucomplex * VmUM;
41          cucomplex * invVmUM ;
42          cucomplex * VpUM ;
43          for (int i=0; i<1; i++  )
44          {
45              cudaMalloc( (void **)&(VmUM),    MMsize );
46              cudaMalloc( (void **)&(invVmUM), MMsize );
```

C 程序调用 cuBLAS 函数库 III

```
47                 cudaMalloc( (void **)&(VpUM),     MMsize );
48             }
49
50 #ifdef __single
51 #else
52             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+12, M6p, 1, VmUM,
                   1) );
53
54             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+10, M4p, 1, VmUM,
                   1) );
55             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+8,  M2p, 1, VmUM,
                   1) );
56             cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                   MMdim, MMdim,
57                     d_cone, M6p,  MMdim, VmUM, MMdim, d_czero, vM, MMdim) );
58
59             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+6,  M6p,   1, vM,
                   1) );
60             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+4,  M4p,   1, vM,
                   1) );
61             cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+2,  M2p,   1, vM,
                   1) );
62             cublacsCHECK(cublasZaxpy(handle1, MMdim , d_carray+0,  d_IMp, MMdim
                   +1, vM, MMdim+1) );
63             cublacsCHECK(cublasZcopy(handle1, MMdim2, vM, 1, VmUM, 1) );
64 #endif
65
66 #ifdef __single
```

C 程序调用 cuBLAS 函数库 IV

```
67#else
68            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+13, M6p, 1, uM, 1)
                    );
69            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+11, M4p, 1, uM, 1)
                    );
70            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+9,  M2p, 1, uM, 1)
                    );
71            cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                  MMdim, MMdim,
72                     d_cone, M6p,  MMdim, uM, MMdim, d_czero, VpUM, MMdim) );
73
74            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+7,  M6p,   1, VpUM
                  , 1) );
75            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+5,  M4p,   1, VpUM
                  , 1) );
76            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_carray+3,  M2p,   1, VpUM
                  , 1) );
77            cublacsCHECK(cublasZaxpy(handle1, MMdim , d_carray+1,  d_IMp, MMdim
                  +1, VpUM, MMdim+1) );
78            cublacsCHECK(cublasZcopy(handle1, MMdim2, vM, 1, VmUM, 1) );
79            cublacsCHECK(cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim,
                  MMdim, MMdim,
80                              d_cone, d_Mp,  MMdim, VpUM, MMdim, d_czero, uM,
                                    MMdim) );
81            cublacsCHECK(cublasZcopy(handle1, MMdim2, uM, 1, VpUM, 1) );
82#endif
83
```

C 程序调用 cuBLAS 函数库 V

```
84            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_cngone,  uM,  1, VmUM, 1)
                    );
85            cublacsCHECK(cublasZaxpy(handle1, MMdim2, d_cone,    vM,  1, VpUM, 1)
                    );
86
87            cublacsCHECK( cublasGetVector(MMdim2, sizeof(cucomplex) , VmUM, 1,
                  h_expMp, 1) );
88
89#ifdef __single
90#else
91            const int batchsize = 1;
92            const int NNdim=MMdim;
93
94            int *PArray, *InfoArray;
95            cudacall(cudaMalloc(&PArray,     sizeof(int) * batchsize * NNdim ));
96            cudacall(cudaMalloc(&InfoArray,  sizeof(int) * batchsize ));
97            int h_InfoArray[batchsize];
98
99            //Random matrix with full pivots
100           cucomplex full_pivot[4*4] = {   0.5, 3, 4, 4,
101                                           1,  3, 10, 4,
102                                           4 , 9, 16, 4,
103                                           4,  4,  4, 4};
104           //Almost same as above matrix with first pivot zero
105           cucomplex zero_pivot[3*3] = { 0, 3, 4,
106                                         1, 3, 10,
107                                         4 , 9, 16 };
108
```

C 程序调用 cuBLAS 函数库 VI

```
109         cucomplex ** AA_hh = (cucomplex **)malloc( sizeof(cucomplex *) *
                batchsize );
110         cucomplex ** AA_hd = (cucomplex **)malloc( sizeof(cucomplex *) *
                batchsize );
111         cucomplex *  AA_d ;
112         cucomplex ** AA_dd;
113         AA_d = VmUM;
114         cudaMalloc( &AA_dd, sizeof(cucomplex*) * batchsize              );
115         AA_hd[0] = AA_d;
116         for (int i = 1; i < batchsize; i++)
117             AA_hd[i] = AA_hd[i-1] + NNdim*NNdim;
118
119         cudaMemcpy(AA_dd, AA_hd, sizeof(cucomplex *) * batchsize,
                cudaMemcpyHostToDevice );
120
121
122         cucomplex ** CC_hh = (cucomplex **)malloc(sizeof(cucomplex *) *
                batchsize );
123         cucomplex ** CC_hd = (cucomplex **)malloc(sizeof(cucomplex *) *
                batchsize );
124         cucomplex *  CC_h  = (cucomplex * )malloc(sizeof(cucomplex  ) * NNdim
                *NNdim * batchsize);
125         cucomplex *  CC_d ;
126         cucomplex ** CC_dd;
127         cudaMalloc( &CC_d,  sizeof(cucomplex ) * batchsize *NNdim*NNdim );
128         cudaMalloc( &CC_dd, sizeof(cucomplex*) * batchsize              );
129         for (int i = 0; i < batchsize; i++)
130             CC_hh[i] = CC_h + (i*NNdim*NNdim);
```

C 程序调用 cuBLAS 函数库 VII

```
131             CC_hd[0] = CC_d;
132             for (int i = 1; i < batchsize; i++)
133                 CC_hd[i] = CC_hd[i-1] + i*NNdim*NNdim;
134             cudaMemcpy(CC_dd,   CC_hd, sizeof(cucomplex *) * batchsize,
                    cudaMemcpyHostToDevice );
135
136             cublasZgetrfBatched(handle1,NNdim,AA_dd,NNdim,PArray,InfoArray,
                    batchsize);
137             cudaMemcpy(h_InfoArray,InfoArray,batchsize*sizeof(int),
                    cudaMemcpyDeviceToHost);
138             for (int i=0; i<batchsize; i++)
139             {
140                 if(h_InfoArray[i] != 0 )
141                         {
142                             cout << "cublas*getrfBatched is failed! i = " <<
                                    i << endl;
143                             cudaDeviceReset();
144                         }
145             }
146
147             cublasZgetriBatched(handle1,NNdim,(const cucomplex **)AA_dd,NNdim,
                    PArray,CC_dd,NNdim,InfoArray,batchsize);
148             cudaMemcpy(h_InfoArray,InfoArray,batchsize*sizeof(int),
                    cudaMemcpyDeviceToHost);
149             for (int i=0; i<batchsize; i++)
150             {
151                 if(h_InfoArray[i] != 0 )
152                         {
```

C 程序调用 cuBLAS 函数库 VIII

```
153                                 cout << "cublas*getriBatched is failed! i = " <<
                                        i << endl;
154                                 cudaDeviceReset();
155                             }
156             }
157
158             cublasZgemm(handle1,CUBLAS_OP_N, CUBLAS_OP_N, MMdim, MMdim, MMdim,
159                             d_cone, CC_d,  MMdim, VpUM, MMdim, d_czero,
                                    d_expMp, MMdim);
160
161 #endif
162
163             cudaMemcpy( h_expMp, d_expMp, batchsize*MMsize,
                        cudaMemcpyDeviceToHost);
164
165             cudaFree(PArray); cudaFree(InfoArray);
166             cudaFree(AA_d); cudaFree(AA_dd); free(AA_hd);
167             cudaFree(CC_d); cudaFree(CC_dd); free(CC_hd); free(CC_hh);
168
169             cudaFree(M2p); cudaFree(M4p); cudaFree(M6p); cudaFree(vM); cudaFree(
                        uM);
170             cudaFree(VmUM); cudaFree(invVmUM); cudaFree(VpUM);
171             cublasDestroy_v2(handle1);
172             cudaDeviceReset();
173
174 }
175 #endif
```

提交作业 (多 GPU 程序) I

```
bsub -m nodename -q k40 -e %J.err -o %J.log ./simpleMultiGPU
```

```
/*
 * Copyright 1993-2015 NVIDIA Corporation.  All rights reserved.
 *
 * Please refer to the NVIDIA end user license agreement (EULA) associated
 * with this source code for terms and conditions that govern your use of
 * this software. Any use, reproduction, disclosure, or distribution of
 * this software and related documentation outside the terms of the EULA
 * is strictly prohibited.
 *
 */

/*
 * This application demonstrates how to use the CUDA API to use multiple GPUs,
 * with an emphasis on simple illustration of the techniques (not on performance)
    .
 *
 * Note that in order to detect multiple GPUs in your system you have to disable
 * SLI in the nvidia control panel. Otherwise only one GPU is visible to the
 * application. On the other side, you can still extend your desktop to screens
 * attached to both GPUs.
 */

// System includes
#include <stdio.h>
```

提交作业 (多 GPU 程序) II

```
24 #include <assert.h>
25
26 // CUDA runtime
27 #include <cuda_runtime.h>
28
29 // helper functions and utilities to work with CUDA
30 #include <helper_functions.h>
31 #include <helper_cuda.h>
32 #include <timer.h>
33
34 #ifndef MAX
35 #define MAX(a,b) (a > b ? a : b)
36 #endif
37
38 #include "simpleMultiGPU.h"
39
40 ////////////////////////////////////////////////////////////////////////////////
41 // Data configuration
42 ////////////////////////////////////////////////////////////////////////////////
43 const int MAX_GPU_COUNT = 32;
44 const int DATA_N        = 1048576 * 32;
45
46 ////////////////////////////////////////////////////////////////////////////////
47 // Simple reduction kernel.
48 // Refer to the 'reduction' CUDA Sample describing
49 // reduction optimization strategies
50 ////////////////////////////////////////////////////////////////////////////////
51 __global__ static void reduceKernel(float *d_Result, float *d_Input, int N)
```

提交作业 (多 GPU 程序) III

```
52{
53    const int     tid = blockIdx.x * blockDim.x + threadIdx.x;
54    const int threadN = gridDim.x * blockDim.x;
55    float sum = 0;
56
57    for (int pos = tid; pos < N; pos += threadN)
58        sum += d_Input[pos];
59
60    d_Result[tid] = sum;
61}
62
63////////////////////////////////////////////////////////////////////////////////
64// Program main
65////////////////////////////////////////////////////////////////////////////////
66int main(int argc, char **argv)
67{
68    //Solver config
69    TGPUplan      plan[MAX_GPU_COUNT];
70
71    //GPU reduction results
72    float     h_SumGPU[MAX_GPU_COUNT];
73
74    float sumGPU;
75    double sumCPU, diff;
76
77    int i, j, gpuBase, GPU_N;
78
79    const int  BLOCK_N = 32;
```

提交作业 (多 GPU 程序) IV

```
80      const int THREAD_N = 256;
81      const int  ACCUM_N = BLOCK_N * THREAD_N;
82
83      printf("Starting simpleMultiGPU\n");
84      checkCudaErrors(cudaGetDeviceCount(&GPU_N));
85
86      if (GPU_N > MAX_GPU_COUNT)
87      {
88          GPU_N = MAX_GPU_COUNT;
89      }
90
91      printf("CUDA-capable device count: %i\n", GPU_N);
92
93      printf("Generating input data...\n\n");
94
95      //Subdividing input data across GPUs
96      //Get data sizes for each GPU
97      for (i = 0; i < GPU_N; i++)
98      {
99          plan[i].dataN = DATA_N / GPU_N;
100     }
101
102     //Take into account "odd" data sizes
103     for (i = 0; i < DATA_N % GPU_N; i++)
104     {
105         plan[i].dataN++;
106     }
107
```

提交作业 (多 GPU 程序) V

```
108     //Assign data ranges to GPUs
109     gpuBase = 0;
110 
111     for (i = 0; i < GPU_N; i++)
112     {
113         plan[i].h_Sum = h_SumGPU + i;
114         gpuBase += plan[i].dataN;
115     }
116 
117     //Create streams for issuing GPU command asynchronously and allocate memory (
            GPU and System page-locked)
118     for (i = 0; i < GPU_N; i++)
119     {
120         checkCudaErrors(cudaSetDevice(i));
121         checkCudaErrors(cudaStreamCreate(&plan[i].stream));
122         //Allocate memory
123         checkCudaErrors(cudaMalloc((void **)&plan[i].d_Data, plan[i].dataN *
                sizeof(float)));
124         checkCudaErrors(cudaMalloc((void **)&plan[i].d_Sum, ACCUM_N * sizeof(
                float)));
125         checkCudaErrors(cudaMallocHost((void **)&plan[i].h_Sum_from_device,
                ACCUM_N * sizeof(float)));
126         checkCudaErrors(cudaMallocHost((void **)&plan[i].h_Data, plan[i].dataN *
                sizeof(float)));
127 
128         for (j = 0; j < plan[i].dataN; j++)
129         {
130             plan[i].h_Data[j] = (float)rand() / (float)RAND_MAX;
```

提交作业 (多 GPU 程序) VI

```
131         }
132     }
133
134     //Start timing and compute on GPU(s)
135     printf("Computing with %d GPUs...\n", GPU_N);
136     StartTimer();
137
138     //Copy data to GPU, launch the kernel and copy data back. All asynchronously
139     for (i = 0; i < GPU_N; i++)
140     {
141         //Set device
142         checkCudaErrors(cudaSetDevice(i));
143
144         //Copy input data from CPU
145         checkCudaErrors(cudaMemcpyAsync(plan[i].d_Data, plan[i].h_Data, plan[i].
                dataN * sizeof(float), cudaMemcpyHostToDevice, plan[i].stream));
146
147         //Perform GPU computations
148         reduceKernel<<<BLOCK_N, THREAD_N, 0, plan[i].stream>>>(plan[i].d_Sum,
                plan[i].d_Data, plan[i].dataN);
149         getLastCudaError("reduceKernel() execution failed.\n");
150
151         //Read back GPU results
152         checkCudaErrors(cudaMemcpyAsync(plan[i].h_Sum_from_device, plan[i].d_Sum,
                ACCUM_N *sizeof(float), cudaMemcpyDeviceToHost, plan[i].stream));
153     }
154
155     //Process GPU results
```

提交作业 (多 GPU 程序) VII

```
156     for (i = 0; i < GPU_N; i++)
157     {
158         float sum;
159 
160         //Set device
161         checkCudaErrors(cudaSetDevice(i));
162 
163         //Wait for all operations to finish
164         cudaStreamSynchronize(plan[i].stream);
165 
166         //Finalize GPU reduction for current subvector
167         sum = 0;
168 
169         for (j = 0; j < ACCUM_N; j++)
170         {
171             sum += plan[i].h_Sum_from_device[j];
172         }
173 
174         *(plan[i].h_Sum) = (float)sum;
175 
176         //Shut down this GPU
177         checkCudaErrors(cudaFreeHost(plan[i].h_Sum_from_device));
178         checkCudaErrors(cudaFree(plan[i].d_Sum));
179         checkCudaErrors(cudaFree(plan[i].d_Data));
180         checkCudaErrors(cudaStreamDestroy(plan[i].stream));
181     }
182 
183     sumGPU = 0;
```

提交作业 (多 GPU 程序) VIII

```
184
185     for (i = 0; i < GPU_N; i++)
186     {
187         sumGPU += h_SumGPU[i];
188     }
189
190     printf("  GPU Processing time: %f (ms)\n\n", GetTimer());
191
192     // Compute on Host CPU
193     printf("Computing with Host CPU...\n\n");
194
195     sumCPU = 0;
196
197     for (i = 0; i < GPU_N; i++)
198     {
199         for (j = 0; j < plan[i].dataN; j++)
200         {
201             sumCPU += plan[i].h_Data[j];
202         }
203     }
204
205     // Compare GPU and CPU results
206     printf("Comparing GPU and Host CPU results...\n");
207     diff = fabs(sumCPU - sumGPU) / fabs(sumCPU);
208     printf("  GPU sum: %f\n  CPU sum: %f\n", sumGPU, sumCPU);
209     printf("  Relative difference: %E \n\n", diff);
210
211     // Cleanup and shutdown
```

提交作业 (多 GPU 程序) IX

```
212     for (i = 0; i < GPU_N; i++)
213     {
214         checkCudaErrors(cudaSetDevice(i));
215         checkCudaErrors(cudaFreeHost(plan[i].h_Data));
216 
217         // cudaDeviceReset causes the driver to clean up all state. While
218         // not mandatory in normal operation, it is good practice.  It is also
219         // needed to ensure correct operation when the application is being
220         // profiled. Calling cudaDeviceReset causes all profile data to be
221         // flushed before the application exits
222         cudaDeviceReset();
223     }
224 
225     exit((diff < 1e-5) ? EXIT_SUCCESS : EXIT_FAILURE);
226 }
```

作业状态输出 I

```
Sender: LSF System <lsfadmin@node48>
Subject: Job 27734: <./simpleMultiGPU> Done

Job <./simpleMultiGPU> was submitted from host <node48> by user <wszhang> in
     cluster <chinagrid>.
Job was executed on host(s) <node48>, in queue <k40>, as user <wszhang> in
     cluster <chinagrid>.
</home/nic/wszhang> was used as the home directory.
</home/nic/wszhang/OPT/expm/cuda_samples/0_Simple/simpleMultiGPU> was used as the
      working directory.
Started at Wed Nov 25 21:03:01 2015
Results reported at Wed Nov 25 21:03:15 2015

Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
./simpleMultiGPU
------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time   :      4.56 sec.
    Max Memory :         1 MB
    Max Swap   :        32 MB
```

作业状态输出 II

```
25
26     Max Processes   :         1
27     Max Threads     :         1
28
29The output (if any) follows:
30
31Starting simpleMultiGPU
32CUDA-capable device count: 2
33Generating input data...
34
35Computing with 2 GPUs...
36   GPU Processing time: 10.902000 (ms)
37
38Computing with Host CPU...
39
40Comparing GPU and Host CPU results...
41   GPU sum: 16777280.000000
42   CPU sum: 16777294.395033
43   Relative difference: 8.580068E-07
44
45PS:
46Read file <27734.err> for stderr output of this job.
```

联系信息

- **中国科学技术大学超算中心:**

  办公室科大东区新图书馆一楼东侧 126 室
  电话:0551-63602248
  信箱:sccadmin@ustc.edu.cn
  主页:http://scc.ustc.edu.cn