基于 GPU 异构平台的 AES 并行计算

何振忠

School of Computer
South China Normal University

6/3, 2013

Mind Map


基于GPU异构平台的AES并行计算
从OpenCL开始
什么是OpenCL
为什么选择OpenCL
进度
组建AMD流计算环境
用OpenCL实现AES加解密
不同平台之间比较
OpenCL实现AES优化
从最优算法开始：Furious
OpenCL矢量
内存访问的优化
no functions
precomputation
后续
查找国内外有关GPU并行加速文献，比较一下我们项目的优势和不足
继续优化AES的并行加速
尝试其它分组加密算法
搭建一个台式机的测试环境

Outline

Open Computing Language

OpenCL 为异构平台提供一个编写程序，特别是并行程序的开放框架标准。

KHRONOS GROUP

OpenCL
The Open Standard for Heterogeneous Parallel Programming

组成

OpenCL 由两部分组成：

- 编写内核程序的语言
  OpenCL C 是基于 ISO/IEC 9899:1999 C 标准做了一定的扩展和限制.

Built-in Scalar Data Types [6.1.1]

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

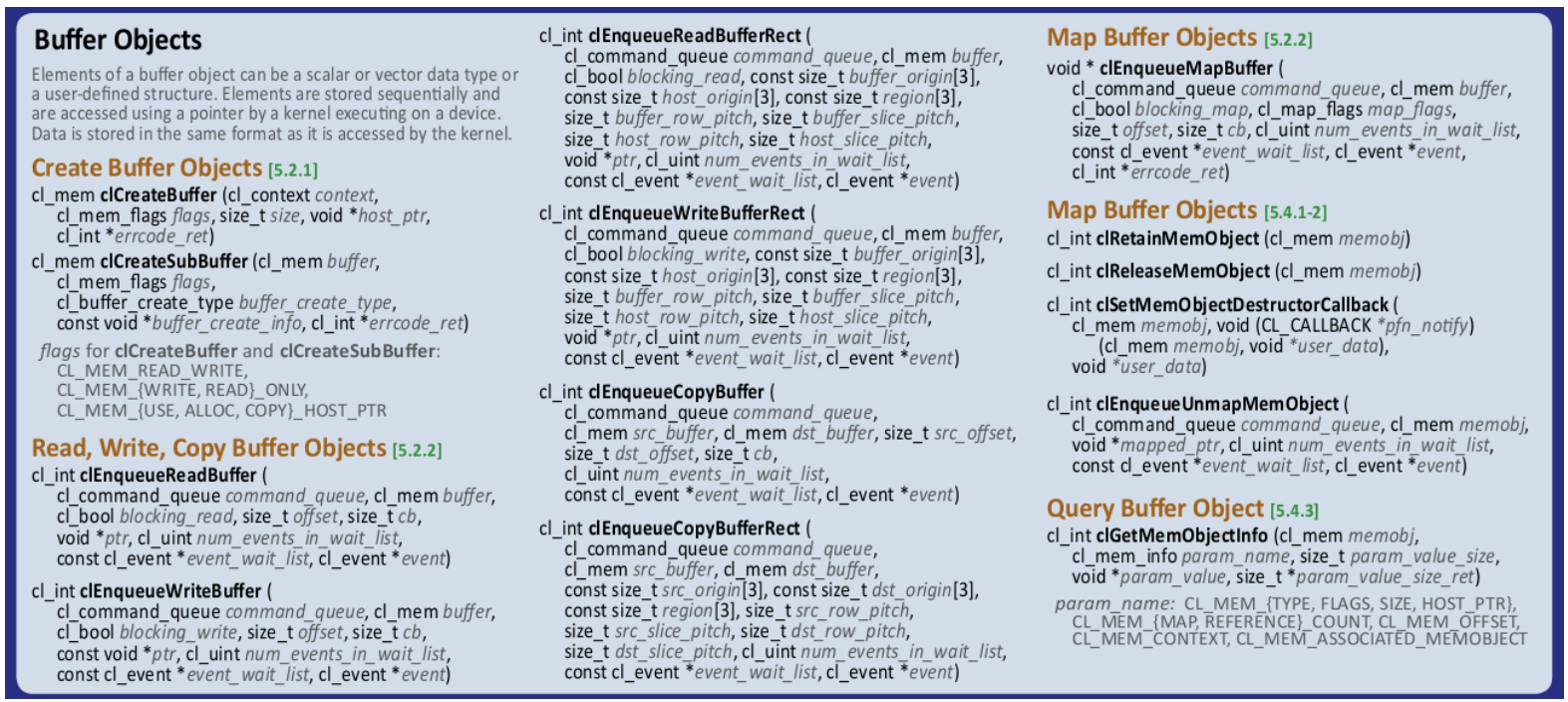
Built-in Vector Data Types [6.1.2]

OpenCL Type	API Type	Description
char*n*	cl_char*n*	8-bit signed
uchar*n*	cl_uchar*n*	8-bit unsigned
short*n*	cl_short*n*	16-bit signed
ushort*n*	cl_ushort*n*	16-bit unsigned
int*n*	cl_int*n*	32-bit signed
uint*n*	cl_uint*n*	32-bit unsigned
long*n*	cl_long*n*	64-bit signed
ulong*n*	cl_ulong*n*	64-bt unsigned
float*n*	cl_float*n*	32-bit float

组成

OpenCL 由两部分组成：

- 定义并控制平台的 API
  适用于各种类型异构处理器的坐标数据和基于任务并行计算 API.



Buffer Objects

Elements of a buffer object can be a scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

Create Buffer Objects [5.2.1]

cl_mem **clCreateBuffer** (cl_context *context*,
cl_mem_flags *flags*, size_t *size*, void *host_ptr,
cl_int *errcode_ret)

cl_mem **clCreateSubBuffer** (cl_mem *buffer*,
cl_mem_flags *flags*,
cl_buffer_create_type *buffer_create_type*,
const void *buffer_create_info, cl_int *errcode_ret)

flags for **clCreateBuffer** and **clCreateSubBuffer**:
CL_MEM_READ_WRITE,
CL_MEM_{WRITE, READ}_ONLY,
CL_MEM_{USE, ALLOC, COPY}_HOST_PTR

Read, Write, Copy Buffer Objects [5.2.2]

cl_int **clEnqueueReadBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, size_t *offset*, size_t *cb*,
void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_write*, size_t *offset*, size_t *cb*,
const void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueReadBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_read*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueWriteBufferRect** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_write*, const size_t *buffer_origin*[3],
const size_t *host_origin*[3], const size_t *region*[3],
size_t *buffer_row_pitch*, size_t *buffer_slice_pitch*,
size_t *host_row_pitch*, size_t *host_slice_pitch*,
void *ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBuffer** (
cl_command_queue *command_queue*,
cl_mem *src_buffer*, cl_mem *dst_buffer*, size_t *src_offset*,
size_t *dst_offset*, size_t *cb*,
cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

cl_int **clEnqueueCopyBufferRect** (
cl_command_queue *command_queue*,
cl_mem *src_buffer*, cl_mem *dst_buffer*,
const size_t *src_origin*[3], const size_t *dst_origin*[3],
const size_t *region*[3], size_t *src_row_pitch*,
size_t *src_slice_pitch*, size_t *dst_row_pitch*,
size_t *dst_slice_pitch*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

Map Buffer Objects [5.2.2]

void * **clEnqueueMapBuffer** (
cl_command_queue *command_queue*, cl_mem *buffer*,
cl_bool *blocking_map*, cl_map_flags *map_flags*,
size_t *offset*, size_t *cb*, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event,
cl_int *errcode_ret)

Map Buffer Objects [5.4.1-2]

cl_int **clRetainMemObject** (cl_mem *memobj*)

cl_int **clReleaseMemObject** (cl_mem *memobj*)

cl_int **clSetMemObjectDestructorCallback** (
cl_mem *memobj*, void (CL_CALLBACK *pfn_notify)
(cl_mem *memobj*, void *user_data),
void *user_data)

cl_int **clEnqueueUnmapMemObject** (
cl_command_queue *command_queue*, cl_mem *memobj*,
void *mapped_ptr, cl_uint *num_events_in_wait_list*,
const cl_event *event_wait_list, cl_event *event)

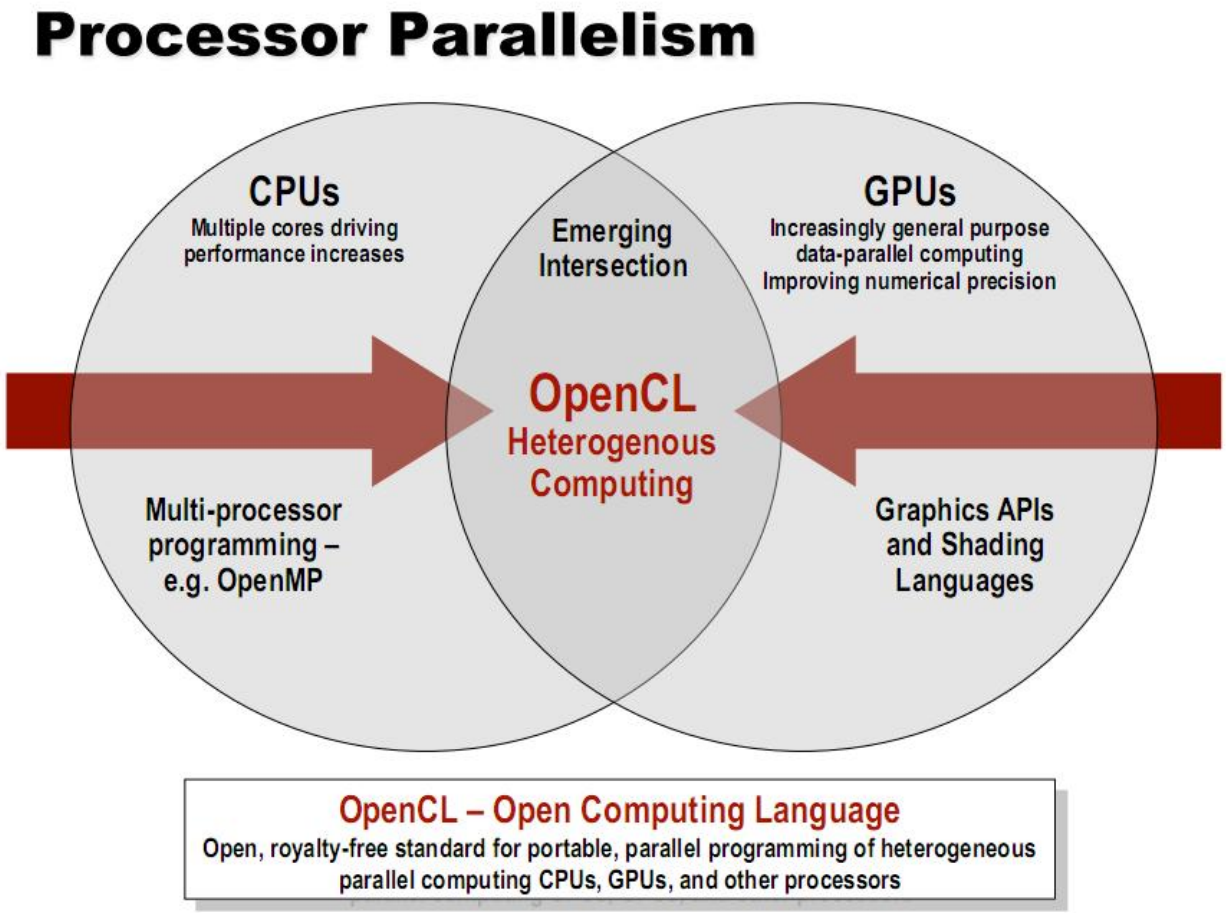
Query Buffer Object [5.4.3]

cl_int **clGetMemObjectInfo** (cl_mem *memobj*,
cl_mem_info *param_name*, size_t *param_value_size*,
void *param_value, size_t *param_value_size_ret)

param_name: CL_MEM_{TYPE, FLAGS, SIZE, HOST_PTR},
CL_MEM_{MAP, REFERENCE}_COUNT, CL_MEM_OFFSET,
CL_MEM_CONTEXT, CL_MEM_ASSOCIATED_MEMOBJECT

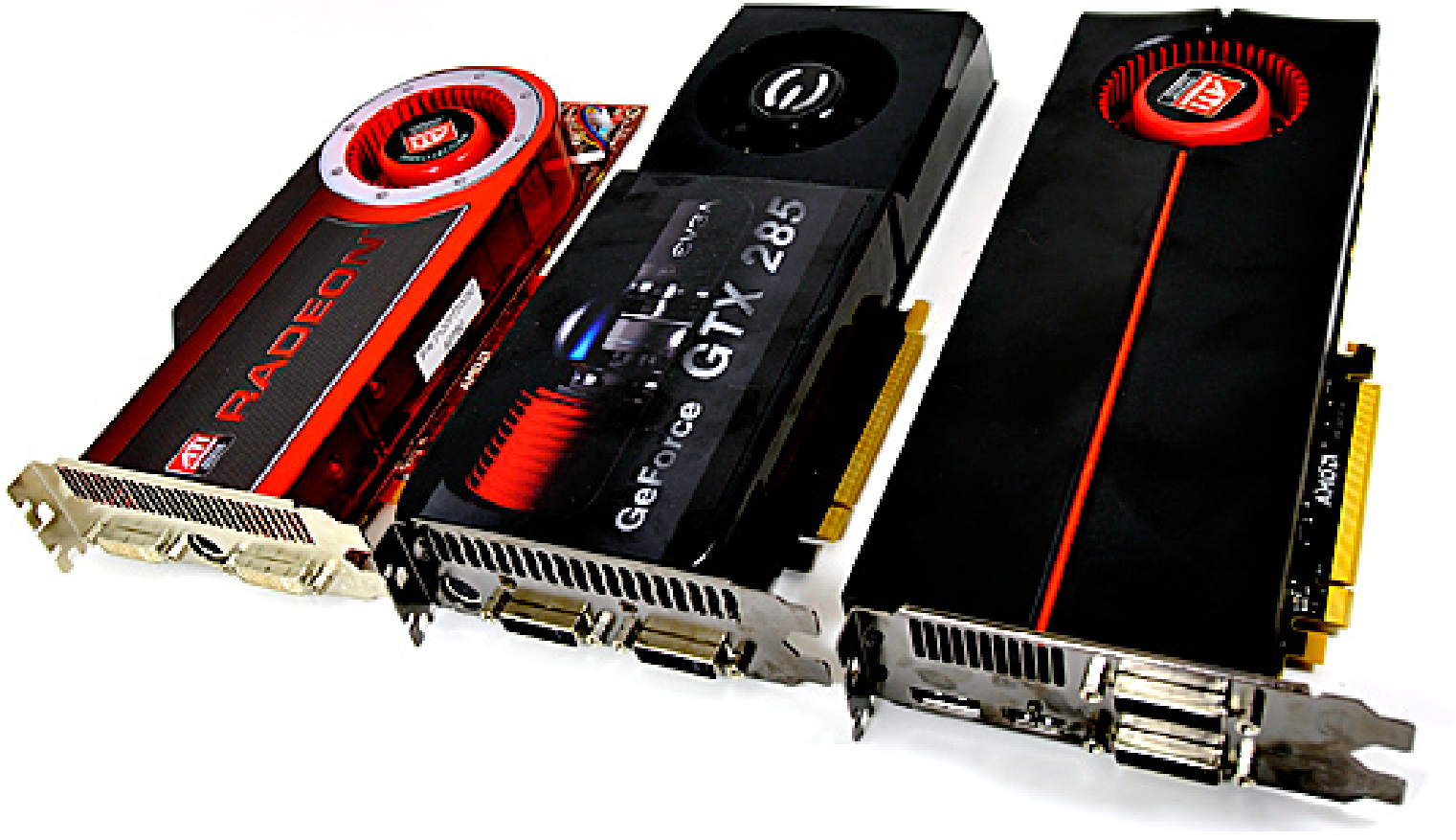
OpenCL 可以充分利用设备的并行性

- GPU 的运算核心数量要远远超过高端 CPU 的核心数量
- GPU 是通过大量并行线程之间交织运算隐藏全局访问的延迟



OpenCL 为程序员提供了平台独立性

问题：不同平台，不同厂商，不同产品型号的 GPU 往往有着不同的架构。
解决：OpenCL 支持各种各样的并行处理器的组合平台。



Windows

- 安装 ATI driver
- AMD Stream SDK
- IDE: MS Visual Studio 2010

Linux

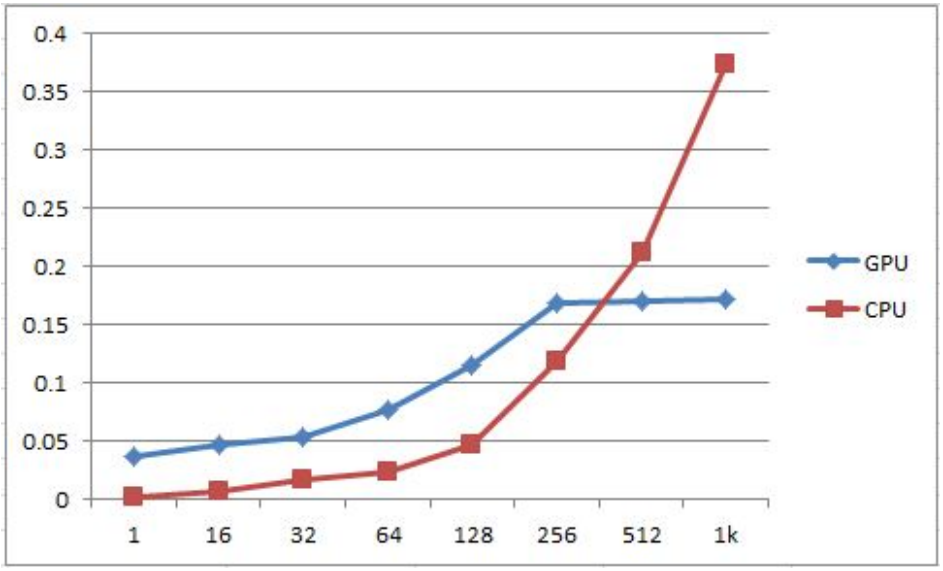
- 安装 ATI driver
- AMD Stream SDK
- 将库文件路径添加到环境变量中

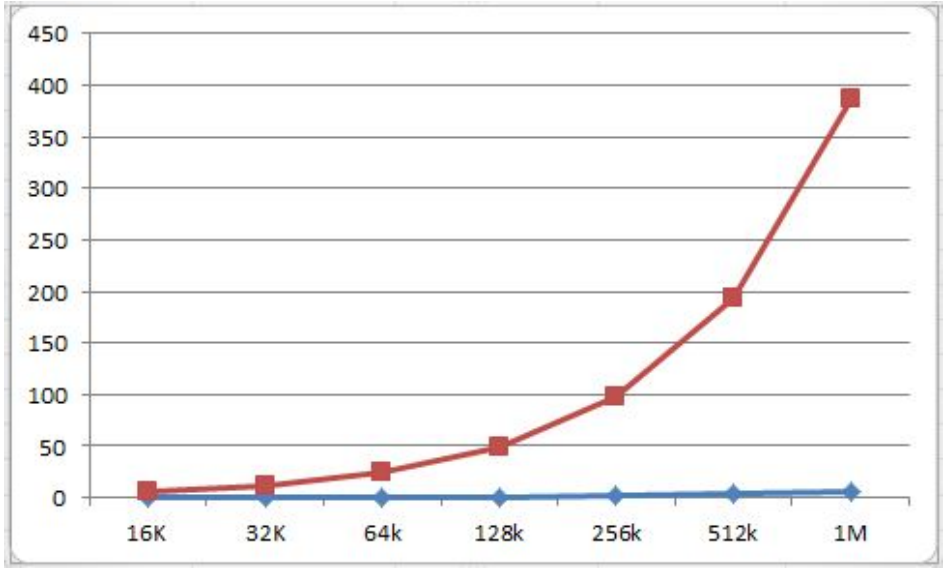
AES Encryption

在 CPU 完成密钥扩展后，将明文和扩展密钥复制到 GPU 上进行 AES 加密。

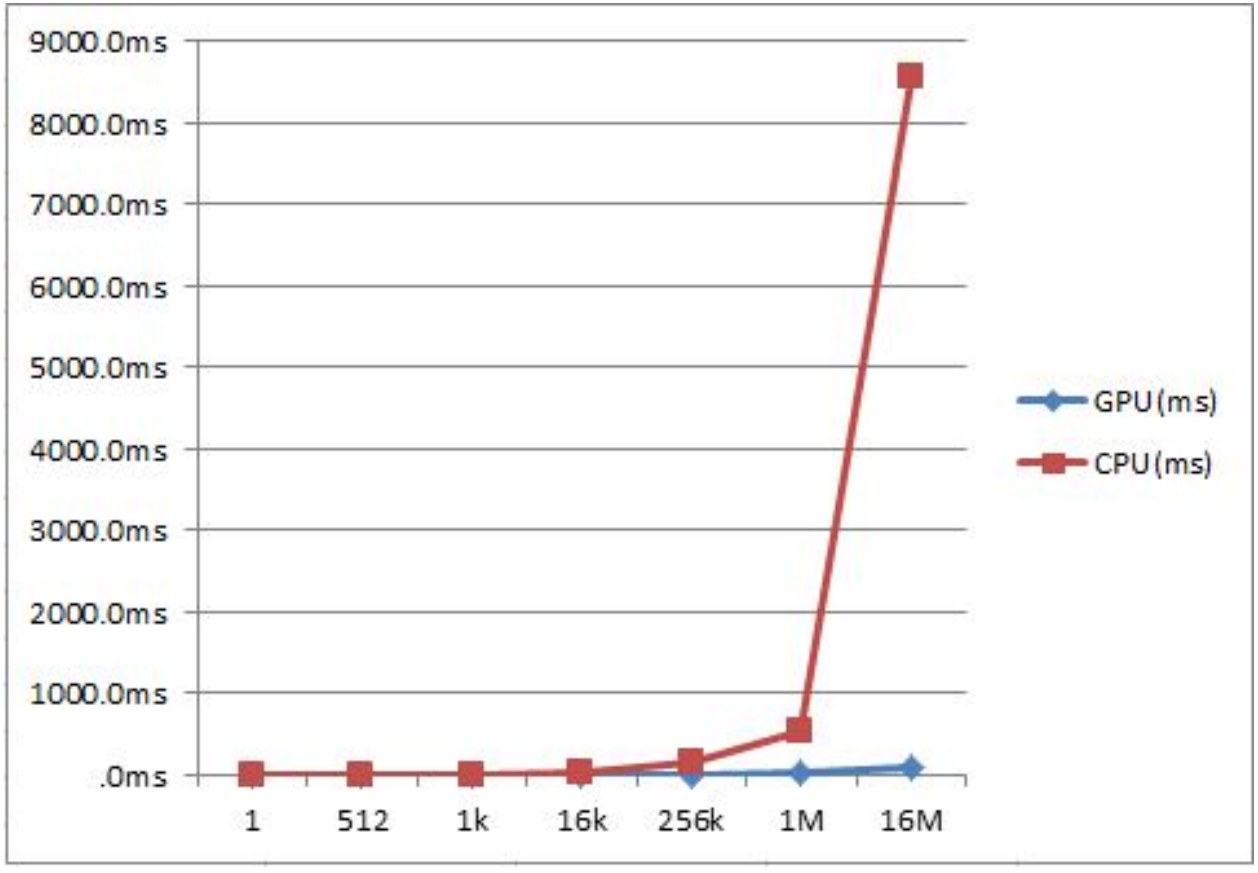
不同平台之间比较

用相同的密钥加密不同长度的明文



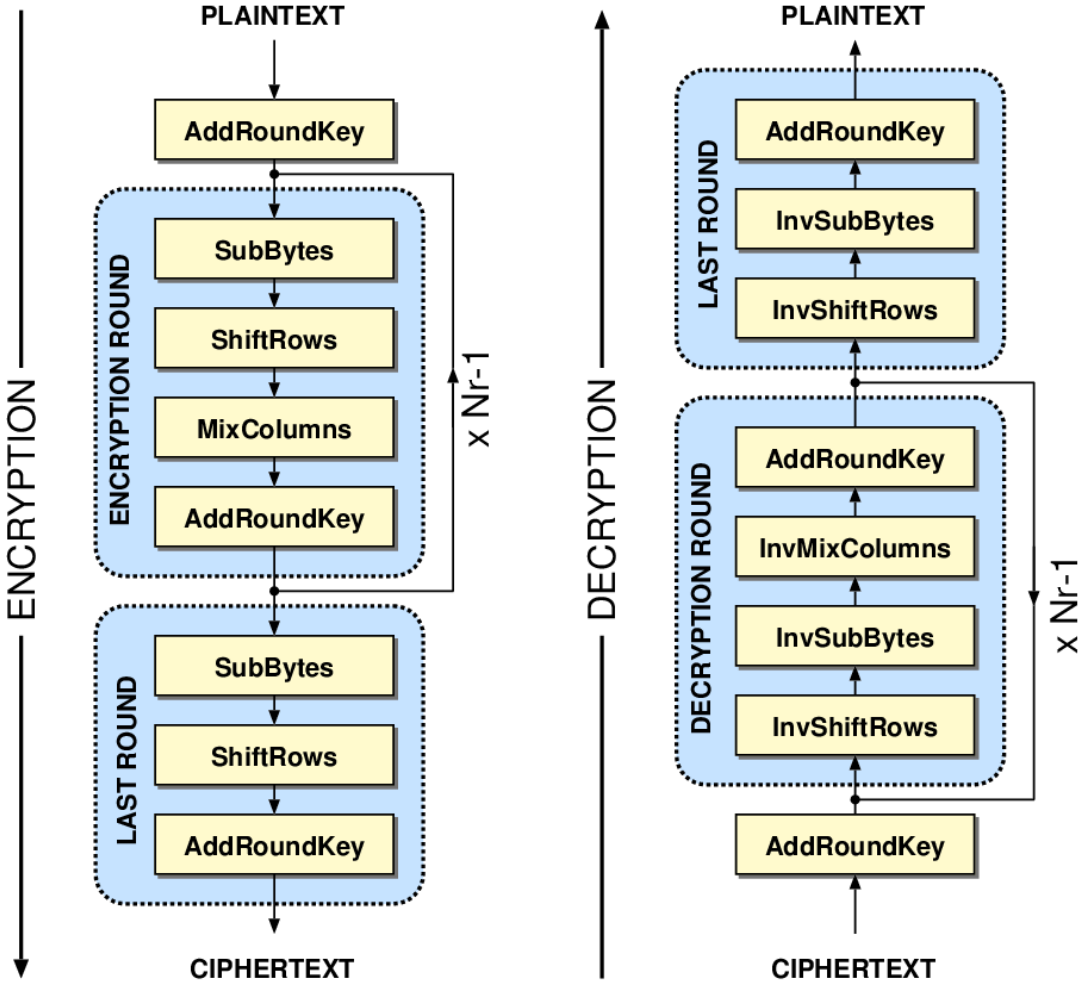


N 个明文，K 个密钥


9000.0ms
8000.0ms
7000.0ms
6000.0ms
5000.0ms
4000.0ms
3000.0ms
2000.0ms
1000.0ms
.0ms
1
512
1k
16k
256k
1M
16M
GPU(ms)
CPU(ms)

Furious

Furious



OpenCL 矢量

矢量化允许一个线程同时执行多个操作。
矢量化在 AMD 的 GPU 上效果更明显。

Global Memory

Global Memory 的访存模式也会在很大程度上影响程序的性能。
2 种需要把数据写回 Global Menory 的情况：

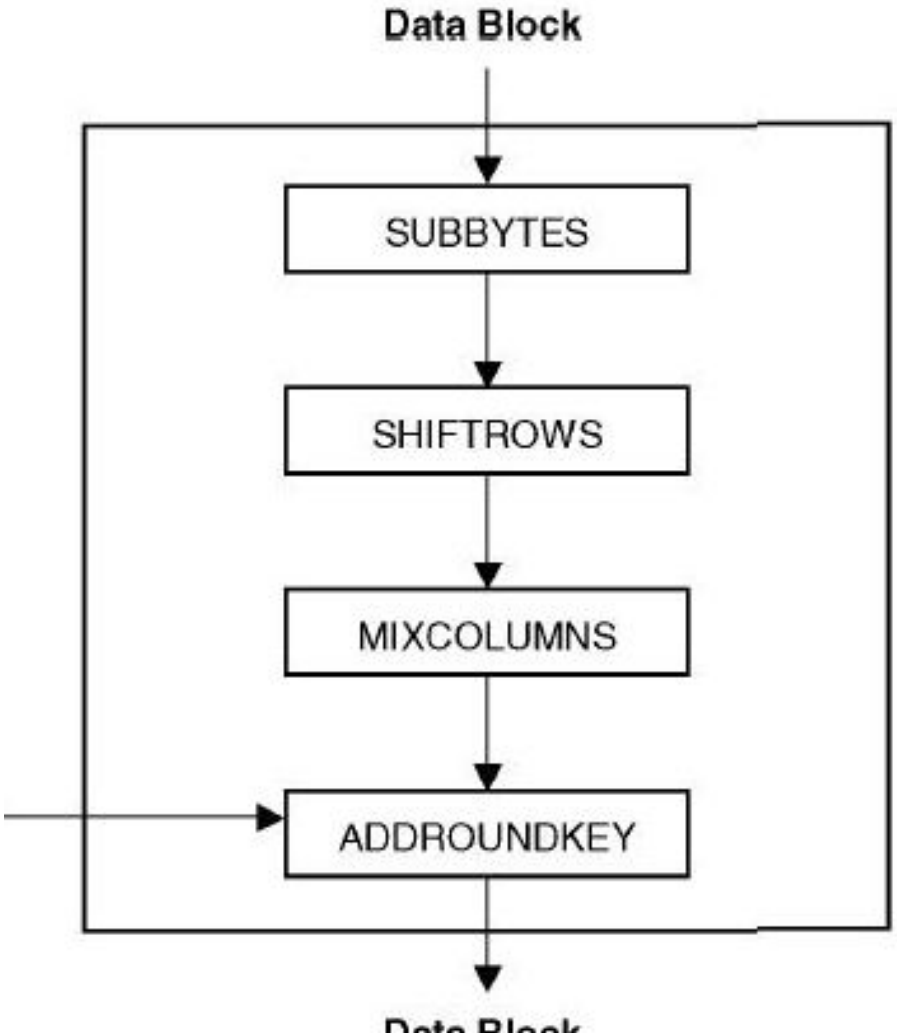
- Kernel 需要保存作为输出的数据
- 需要在 Group 之间交换数据

Local Memory

Local Memory 是指在 GPU 上对每个 Thread Group 有一块可以进行快速访问的内存.

No Functions

将 AddRoundKey、SubBytes、ShiftRows、MixColumns 四个操作都写到循环里，省去函数的调用时间。



Precomputation

将全部密钥都做完密钥扩展后复制到 GPU。

No.1

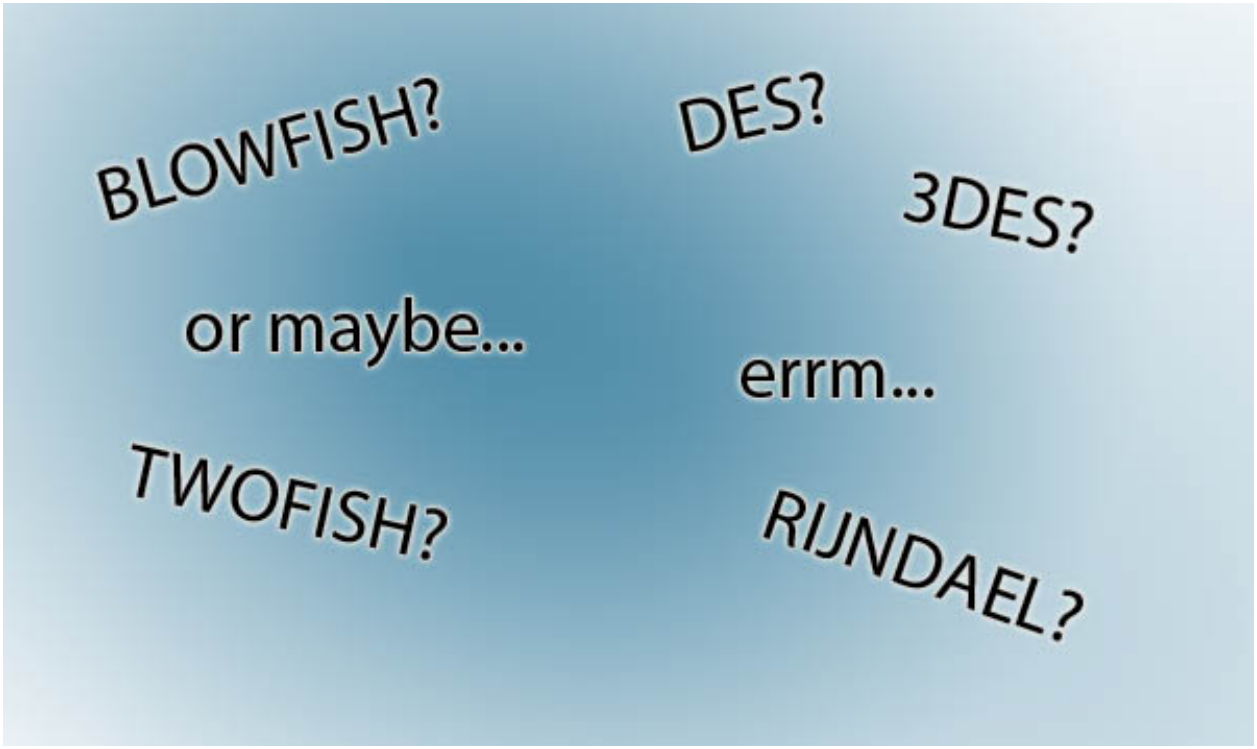
- 查找国内外有关 GPU 并行加速运算的文献，比较一下我们的优劣和不足。

No.2

- 继续优化 AES 的并行加速

No.3

- 尝试其它分组加密算法



No.4

- 搭建一个台式机的测试环境

End

Thank You!