# OpenCL优化思路（一）

为了充分利用寄存器资源，核函数中将尽量加载更多数据。例如，对于uchar类型，4通道图像，一个128位的寄存器可以处理4个像素，为了充分利用寄存器，可以将数据转成float4或int4类型，然后分解成4个uchar4来处理。参考以下代码，该程序完成的功能彩色4通道图像反相，注意程序中核函数的实现及主程序中global\_size的设置。

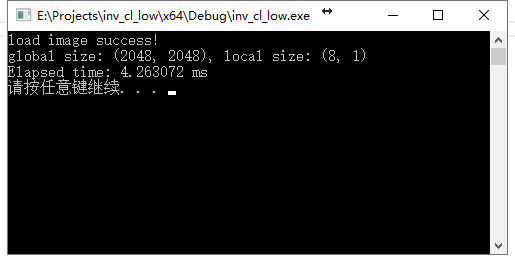
## 低效实现

### 1.1 核函数

|  |
| --- |
| \_\_kernel void inv(\_\_global uchar4\* srcImg, \_\_global uchar4\* dstImg, int height, int width)  {  int2 point = (int2) (get\_global\_id(0), get\_global\_id(1));  if (point.x < width && point.y < height)  {  uchar4 pixel = srcImg[point.y \* width + point.x];  uchar4 val;  val.x = 255 - pixel.x;  val.y = 255 - pixel.y;  val.z = 255 - pixel.z;  val.w = 255 - pixel.w;  dstImg[point.y \* width + point.x] = val;  }  } |

### 1.2 主函数

|  |
| --- |
| // inv\_cl\_low.cpp : Defines the entry point for the console application.  //  #include "stdafx.h"  #include <opencv2/highgui.hpp>  #include <opencv2/core.hpp>  #include <opencv2/imgproc.hpp>  #include <CL/cl.hpp>  #include <fstream>  #include <string>  int main(int argc, char \*argv[])  {  std::string filename = "lena512.jpg";  int normSize = 2048;  int local\_size\_x = 8;  int local\_size\_y = 1;  if (argc > 1)  {  filename = std::string(argv[1]);  }  if (argc > 2)  {  normSize = std::atoi(argv[2]);  }  if (argc > 3)  {  local\_size\_x = std::atoi(argv[3]);  local\_size\_y = std::atoi(argv[3]);  }  if (argc > 4)  {  local\_size\_y = std::atoi(argv[4]);  }  cl\_int err = 0;  cv::Mat image = cv::imread(filename.c\_str());  if (!image.data)  {  printf("read image failed!\n");  return -1;  }  printf("load image success!\n");  cv::resize(image, image, cv::Size(normSize, normSize));  cv::cvtColor(image, image, CV\_BGR2BGRA);  std::vector<cl::Platform> platforms;  err = cl::Platform::get(&platforms);  if (err<0)  {  printf("get platforms failed!\n");  return -1;  }  cl::Platform platform = platforms[1];  std::vector<cl::Device> devices;  err |= platform.getDevices(CL\_DEVICE\_TYPE\_GPU, &devices);  if (err<0)  {  printf("get devices failed!\n");  return -1;  }  cl::Device device = devices.front();  cl::Context context(devices);  std::ifstream file("functions.cl");  std::string src(std::istreambuf\_iterator<char>(file), (std::istreambuf\_iterator<char>()));  cl::Program::Sources sources(1, std::make\_pair(src.c\_str(), src.length() + 1));  cl::Program program(context, sources);  err |= program.build("-cl-std=CL1.2");  if (err<0)  {  printf("build program failed!\n");  return -1;  }  cl::Kernel kernel(program, "inv");  int pix\_num = image.cols \* image.rows;  uchar \*res\_data = new uchar[pix\_num \* 4];  memset(res\_data, 0, pix\_num);  cl::Buffer inputBuf(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, sizeof(uchar) \* pix\_num \* 4, (void\*)image.data);  cl::Buffer outputBuf(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_USE\_HOST\_PTR, sizeof(uchar) \* pix\_num \* 4, (void\*)res\_data);  err |= kernel.setArg(0, inputBuf);  err |= kernel.setArg(1, outputBuf);  err |= kernel.setArg(2, image.rows);  err |= kernel.setArg(3, image.cols);  if (err<0)  {  printf("set kernel argment failed!\n");  return -1;  }  cl::Event event;  cl::CommandQueue queue(context, device, CL\_QUEUE\_PROFILING\_ENABLE);  int g\_x = (image.cols + local\_size\_x - 1) / local\_size\_x \* local\_size\_x;  int g\_y = (image.rows + local\_size\_y - 1) / local\_size\_y \* local\_size\_y;  int l\_x = local\_size\_x;  int l\_y = local\_size\_y;  printf("global size: (%d, %d), local size: (%d, %d)\n", g\_x, g\_y, l\_x, l\_y);  err = queue.enqueueNDRangeKernel(kernel, cl::NullRange, cl::NDRange(g\_x, g\_y), cl::NDRange(l\_x, l\_y), NULL, &event);  if (err<0)  {  printf("excute kernel failed!\n");  return -1;  }  queue.finish();  cl\_ulong start = event.getProfilingInfo<CL\_PROFILING\_COMMAND\_START>();  cl\_ulong end = event.getProfilingInfo<CL\_PROFILING\_COMMAND\_END>();  printf("Elapsed time: %f ms\n", (end - start) / 1000000.0);  queue.enqueueReadBuffer(outputBuf, CL\_TRUE, 0, sizeof(uchar) \* pix\_num \* 4, res\_data);  if (err<0)  {  printf("read buffer failed!\n");  return -1;  }  cv::Mat out(cv::Size(image.cols, image.rows), CV\_8UC4, res\_data);  cv::imwrite("lena\_inv.jpg", out);  system("pause");  return 0;  } |



从结果看，处理2048 \* 2048的图像，Local\_Size设置为(8,1)，程序处理时间需要4.263ms。

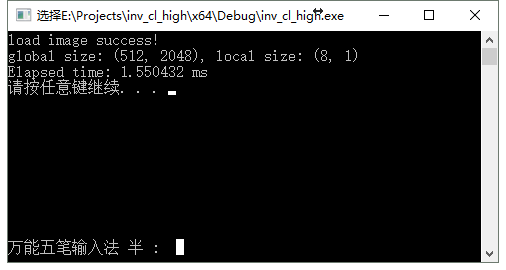
## 高效实现

### 2.1 核函数

|  |
| --- |
| \_\_kernel void inv(\_\_global int4\* srcImg, \_\_global int4\* dstImg, int height, int width)  {  int2 point = (int2) (get\_global\_id(0), get\_global\_id(1));  int4 src = srcImg[point.y \* width + point.x];  uchar4\* p\_src = (uchar4\*)&src;  uchar4 p\_dst[4];  p\_dst[0].x = 255 - p\_src[0].x;  p\_dst[0].y = 255 - p\_src[0].y;  p\_dst[0].z = 255 - p\_src[0].z;  p\_dst[0].w = 255 - p\_src[0].w;  p\_dst[1].x = 255 - p\_src[1].x;  p\_dst[1].y = 255 - p\_src[1].y;  p\_dst[1].z = 255 - p\_src[1].z;  p\_dst[1].w = 255 - p\_src[1].w;  p\_dst[2].x = 255 - p\_src[2].x;  p\_dst[2].y = 255 - p\_src[2].y;  p\_dst[2].z = 255 - p\_src[2].z;  p\_dst[2].w = 255 - p\_src[2].w;  p\_dst[3].x = 255 - p\_src[3].x;  p\_dst[3].y = 255 - p\_src[3].y;  p\_dst[3].z = 255 - p\_src[3].z;  p\_dst[3].w = 255 - p\_src[3].w;  dstImg[point.y \* width + point.x] = \*(int4\*)(p\_dst);  } |

### 2.2 主函数

|  |
| --- |
| // inv\_cl\_high.cpp : Defines the entry point for the console application.  //  #include "stdafx.h"  #include <opencv2/highgui.hpp>  #include <opencv2/core.hpp>  #include <opencv2/imgproc.hpp>  #include <CL/cl.hpp>  #include <fstream>  #include <string>  int main(int argc, char \*argv[])  {  std::string filename = "lena512.jpg";  int normSize = 2048;  int local\_size\_x = 8;  int local\_size\_y = 1;  if (argc > 1)  {  filename = std::string(argv[1]);  }  if (argc > 2)  {  normSize = std::atoi(argv[2]);  }  if (argc > 3)  {  local\_size\_x = std::atoi(argv[3]);  local\_size\_y = std::atoi(argv[3]);  }  if (argc > 4)  {  local\_size\_y = std::atoi(argv[4]);  }  cl\_int err = 0;  cv::Mat image = cv::imread(filename.c\_str());  if (!image.data)  {  printf("read image failed!\n");  return -1;  }  printf("load image success!\n");  cv::resize(image, image, cv::Size(normSize, normSize));  cv::cvtColor(image, image, CV\_BGR2BGRA);  std::vector<cl::Platform> platforms;  err = cl::Platform::get(&platforms);  if (err<0)  {  printf("get platforms failed!\n");  return -1;  }  cl::Platform platform = platforms[1];  std::vector<cl::Device> devices;  err |= platform.getDevices(CL\_DEVICE\_TYPE\_GPU, &devices);  if (err<0)  {  printf("get devices failed!\n");  return -1;  }  cl::Device device = devices.front();  cl::Context context(devices);  std::ifstream file("functions.cl");  std::string src(std::istreambuf\_iterator<char>(file), (std::istreambuf\_iterator<char>()));  cl::Program::Sources sources(1, std::make\_pair(src.c\_str(), src.length() + 1));  cl::Program program(context, sources);  err |= program.build("-cl-std=CL1.2");  if (err<0)  {  size\_t len;  printf("Error: Failed to build program executable!\n");  std::string log = program.getBuildInfo<CL\_PROGRAM\_BUILD\_LOG>(device);  printf("%s\n", log.c\_str());  return -1;  }  cl::Kernel kernel(program, "inv");  int pix\_num = image.cols \* image.rows;  uchar \*res\_data = new uchar[pix\_num \* 4];  memset(res\_data, 0, pix\_num);  cl::Buffer inputBuf(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, sizeof(uchar) \* pix\_num \* 4, (void\*)image.data);  cl::Buffer outputBuf(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, sizeof(uchar) \* pix\_num \* 4, (void\*)res\_data);  err |= kernel.setArg(0, inputBuf);  err |= kernel.setArg(1, outputBuf);  err |= kernel.setArg(2, image.rows);  err |= kernel.setArg(3, image.cols / 4);  if (err<0)  {  printf("set kernel argment failed!\n");  return -1;  }  cl::Event event;  cl::CommandQueue queue(context, device, CL\_QUEUE\_PROFILING\_ENABLE);  int g\_x = (image.cols / 4 + local\_size\_x - 1) / local\_size\_x \* local\_size\_x;  int g\_y = (image.rows + local\_size\_y - 1) / local\_size\_y \* local\_size\_y;  int l\_x = local\_size\_x;  int l\_y = local\_size\_y;  printf("global size: (%d, %d), local size: (%d, %d)\n", g\_x, g\_y, l\_x, l\_y);  err = queue.enqueueNDRangeKernel(kernel, cl::NullRange, cl::NDRange(g\_x, g\_y), cl::NDRange(l\_x, l\_y), NULL, &event);  if (err<0)  {  printf("excute kernel failed!\n");  return -1;  }  queue.finish();  cl\_ulong start = event.getProfilingInfo<CL\_PROFILING\_COMMAND\_START>();  cl\_ulong end = event.getProfilingInfo<CL\_PROFILING\_COMMAND\_END>();  printf("Elapsed time: %f ms\n", (end - start) / 1000000.0);  queue.enqueueReadBuffer(outputBuf, CL\_TRUE, 0, sizeof(uchar) \* pix\_num \* 4, res\_data);  if (err<0)  {  printf("read buffer failed!\n");  return -1;  }  cv::Mat out(cv::Size(image.cols, image.rows), CV\_8UC4, res\_data);  cv::imwrite("lena\_inv.jpg", out);  system("pause");  return 0;  } |



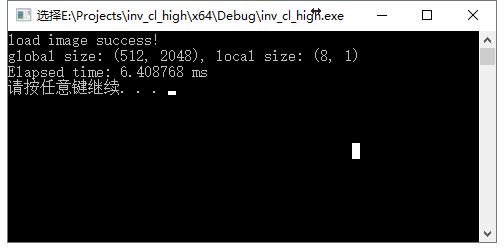
从结果上看，同样处理2048 \* 2048的图像，Local\_Size设置为(8,1)，程序处理的时间为1.550ms，速度为原来3倍多。

## 3 注意

### 3.1 避免多次访问全局内存

尽量避免多次访问全局内存，如以下核函数，采用了同样的思路进行优化，但速度反而比优化前慢很多：

|  |
| --- |
| \_\_kernel void inv(\_\_global float4\* srcImg, \_\_global float4\* dstImg, int height, int width)  {  int2 point = (int2) (get\_global\_id(0), get\_global\_id(1));  if (point.x \* point.y < height \* width)  {  \_\_global uchar4\* src = (\_\_global uchar4\*)(srcImg + point.y \* width + point.x);  \_\_global uchar4\* dst = (\_\_global uchar4\*)(dstImg + point.y \* width + point.x);  for (int i = 0;i < 4;i++)  {  dst[i].x = 255 - src[i].x;  dst[i].y = 255 - src[i].y;  dst[i].z = 255 - src[i].z;  dst[i].w = 255 - src[i].w;  }  }  } |



### 3.2 global\_size的设置

优化后的程序中，由于4个uchar4被看作一个int4类型，因此图像的宽相当于只有原来的1/4，而高度不变。另外，在程序中，图像大小最好与global\_size大小相同，若不同，将其扩展到global\_size大小，以避免在核函数中对越界的判断。