# 向Visual Studio的移植

## 一些头文件

Linux 使用C POSIX头文件，而Windows Visual Studio 使用自己的头文件，需要移植。

op\_lib\_core.h

include <sys/queue.h> // 需要移植 (成功)

sys/queue.h是LINUX/UNIX系统下面的一个标准头文件，用一系列的数据结构定义了一队列。 包括singly-lined list, list, simple queue (Singly-linked Tail queue), tail queue, circle queue五种。

src\core\op\_lib\_core.c: #include <sys/time.h> // 需要移植

src\externlib\op\_util.c: #include <sys/stat.h> // VS也有这个

src\externlib\op\_util.c: #include <sys/types.h> // VS也有这个

sys/stat.h本是linux系统上的头文件。但是在windows上有一套类似linux上的开发环境mingw，了解一下，实际上是用windows上的api封装出来的，确实有sys/stat.h。 但windows系统有自己的一套接口，如果你是用visual studio开发环境来写C++项目，会发现没有sys/stat.h。

#include <stdbool.h> // VS重新定义了 -LJ

#include <strings.h>

strings 头文件中包含了部分函数，没有在 string.h 中出现的。

包括：index, rindex, strcasecmp, strncasecmp 这四个函数：

（1）strcasecmp, strncasecmp重新定义了；

（2）VS中：index, rindex在string.h中定义：

char \* index(const char \*s, int c);

#include <string.h>

main(){

char \*s = "0123456789012345678901234567890";

char \*p;

p = index(s, '5');

printf("%s\n", p);

}

char \* rindex(const char \*s, int c);

#include <string.h>

main(){

char \*s = "0123456789012345678901234567890";

char \*p;

p = rindex(s, '5');

printf("%s\n", p);

}

## OP2\_core代码修改

（1）不用stdbool.h了

（2）c后缀，全部改为cpp

（3）op\_util.cpp

int remainder = (int)fmod(global\_size, mpi\_comm\_size); // 取余数

在VS对整数取余数，也可以用x%y

（4）op\_lib\_core.cpp中，添加#include <sys/time.h> // added by LiJian

错误调试

【1】 error C2057 and error C2466, 需要使用动态数组

  （1） C++语言中的动态数组编程：

int size=50;

   int \*p=new int[size]; 是正确的

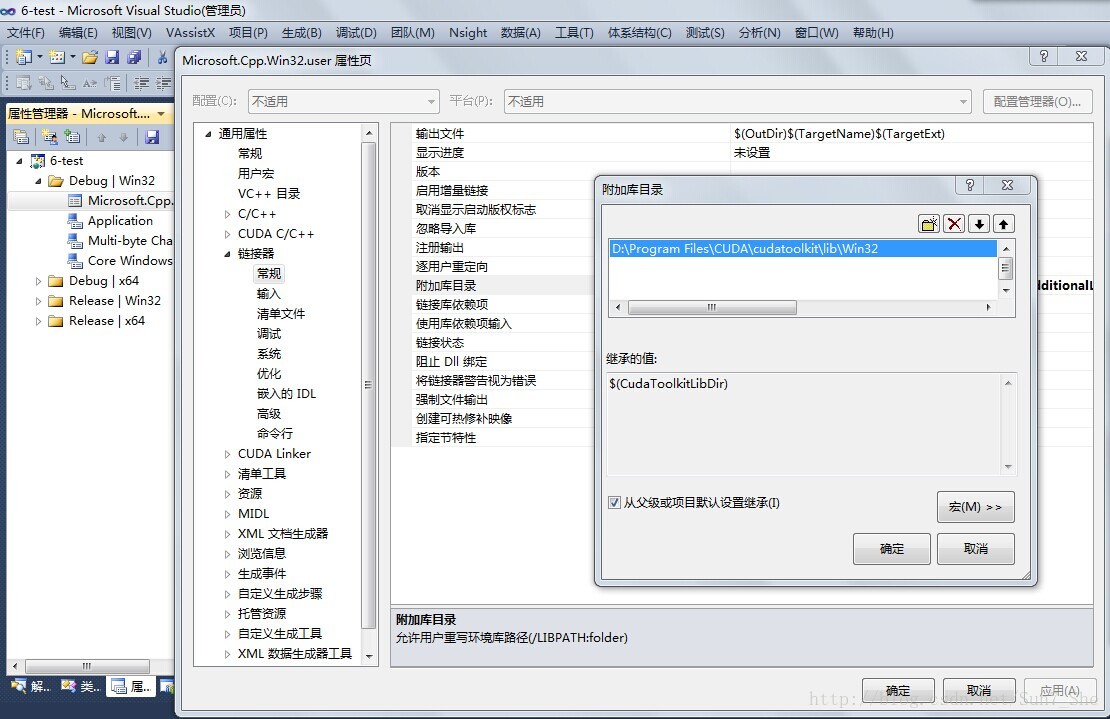
（2）C语言的动态数组

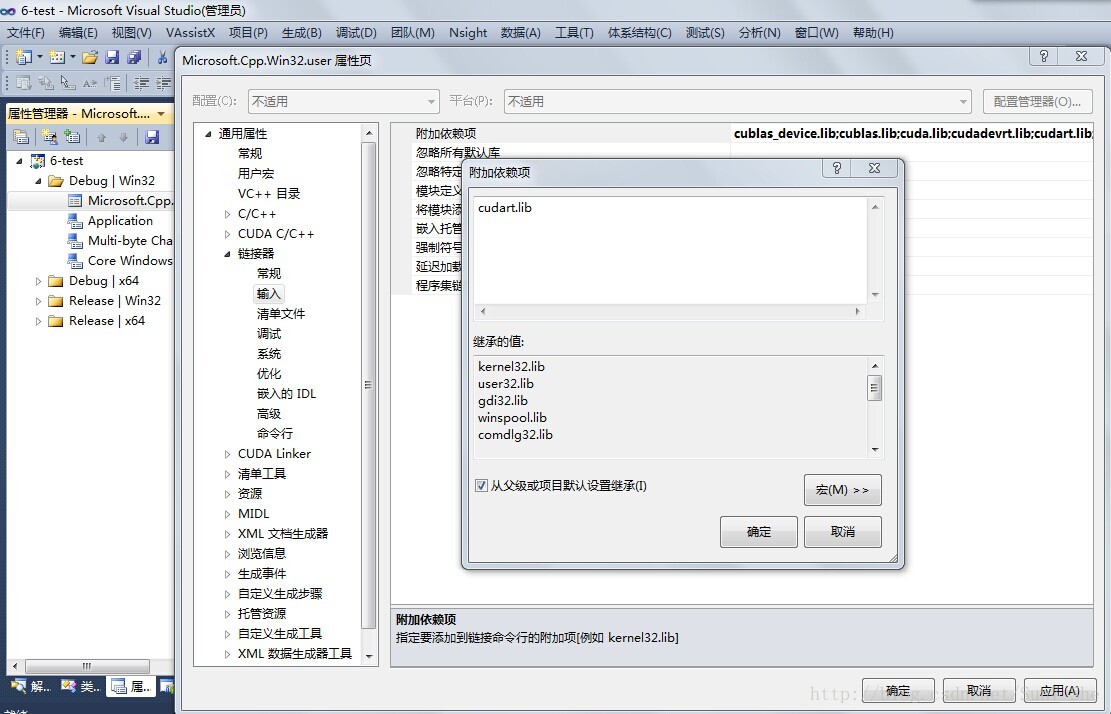
int size=10;

int \*p;

p=(int \*)calloc(size,sizeof(int));

【2】error LNK2001: 无法解析的外部符号 cudaMemcpy





[3] cu文件编译的错误

<https://www.cnblogs.com/rainbow70626/p/8409021.html>

[4] <<< >>> 错误：应输入表达式

该错误可忽略，应为是CUDA语法在CPP中显示错误，但对编译没有影响。

# OP2\_C的代码编译逻辑关系

参考op2/c下面的makefile文件

core: op\_lib\_core.c op\_util.c

hdf5: op\_hdf5.c + op\_util.o

seq: op.seq.c + op\_lib\_core.o

openmp4: op\_openmp4\_decl.c + op\_lib\_core.o

openmp: op\_openmp\_decl.c + op\_lib\_core.o

mpi\_seq: 生成libop2\_mpi.lib

# 附录

OpenMP早期是用来实现跨平台的多线程并发编程的一套标准（2.0）。到了OpenMP4.0加入了对SIMD指令的支持，以实现跨平台的向量化支持。

OpenMP4: 那么如何使用OpenMP来实现SIMD指令优化呢（向量化）呢？简单说只要在代码的循环逻辑前加入#pragma omp simd预处理指令就可以，不需要任何依赖库。

#pragma omp simd指令应用于代码中的循环逻辑，可以让多个迭代的循环利用simd指令实现并发执行。

#if 1

void bgra2rgb(const char \*src,char\*dst,int w,int h)

{

#pragma omp simd

for(int y=0;y<h;++y)

{

for(int x=0;x<w;++x)

{

dst[(y\*w+x)\*3 ] = src[(y\*w+x)\*4 + 2];

dst[(y\*w+x)\*3+1] = src[(y\*w+x)\*4 + 1];

dst[(y\*w+x)\*3+2] = src[(y\*w+x)\*4 + 0];

}

}

}

int main()

{

char bgra\_mat[480\*640\*4];

char rgb\_mat[480\*640\*3];

bgra2rgb(bgra\_mat,rgb\_mat,480,640);

}

#endif

编译：

$ gcc -O3 -fopt-info -fopenmp -mavx2 -o test\_simd test\_simd.c

test\_simd.c:13:3: note: loop vectorized

test\_simd.c:13:3: note: loop versioned for vectorization because of possible aliasing

上面编译命令执行时输出test\_simd.c:13:3: note: loop vectorized,就显示line 13的循环代码已经实现了循环向量化.下面详细解释几个特别的编译选项的意义：

-fopenmp 打开OpenMP预处理指令支持开关，使用此选项，代码中的#pragma omp simd预处理指令才有效。

-mavx2 指定使用intel AVX2指令集。如果目标CPU不支持AVX，也可以根据目标CPU的类型改为低版本的-msse4.1 -msse4.2 -msse4 -mavx

-fopt-info 显示优化过程的输出，该选项只是用于输出显示，指示哪些代码已经被优化了，可以不用，就没有上面的输出显示。

-O3 3级优化选项