

使用visual studio 调试MPI程序

使用VS调试MPI程序有两种方法，一种是通过使用Microsoft visual studio专业版的MPI cluster debugger功能，通过它可以大幅简化调试流程。考虑大多数人无法获得专业版VS，因此我们采用另一种方法。

MPI_Barrier

```
1 MPI_Barrier(MPI_COMM_WORLD);
```

使用MPI Barrier可以达到同步所有进程的目的

使用MPI_Barrier调试MPI程序

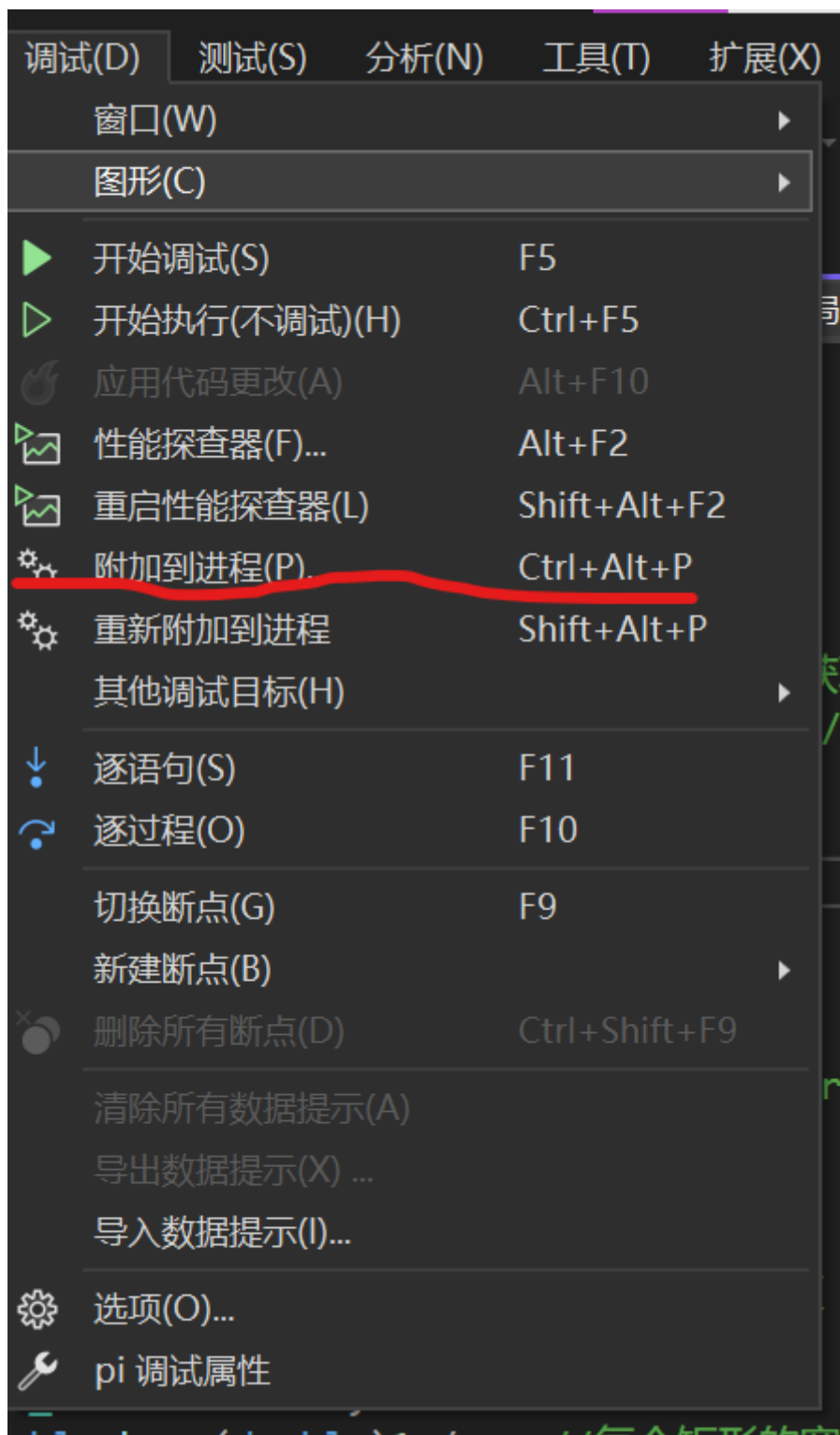
1. 在main函数开头添加以下包含MPI_Barrier的代码段

```
1  int rank, size, temp;
2  MPI_Init( &argc, &argv );
3  MPI_Comm_size( MPI_COMM_WORLD, &size );
4  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
5
6  if (rank == 0)
7  {
8      std::cin >> temp;
9  }
10
11 MPI_Barrier(MPI_COMM_WORLD); // All threads will wait here until you give
    thread 0 an input
```

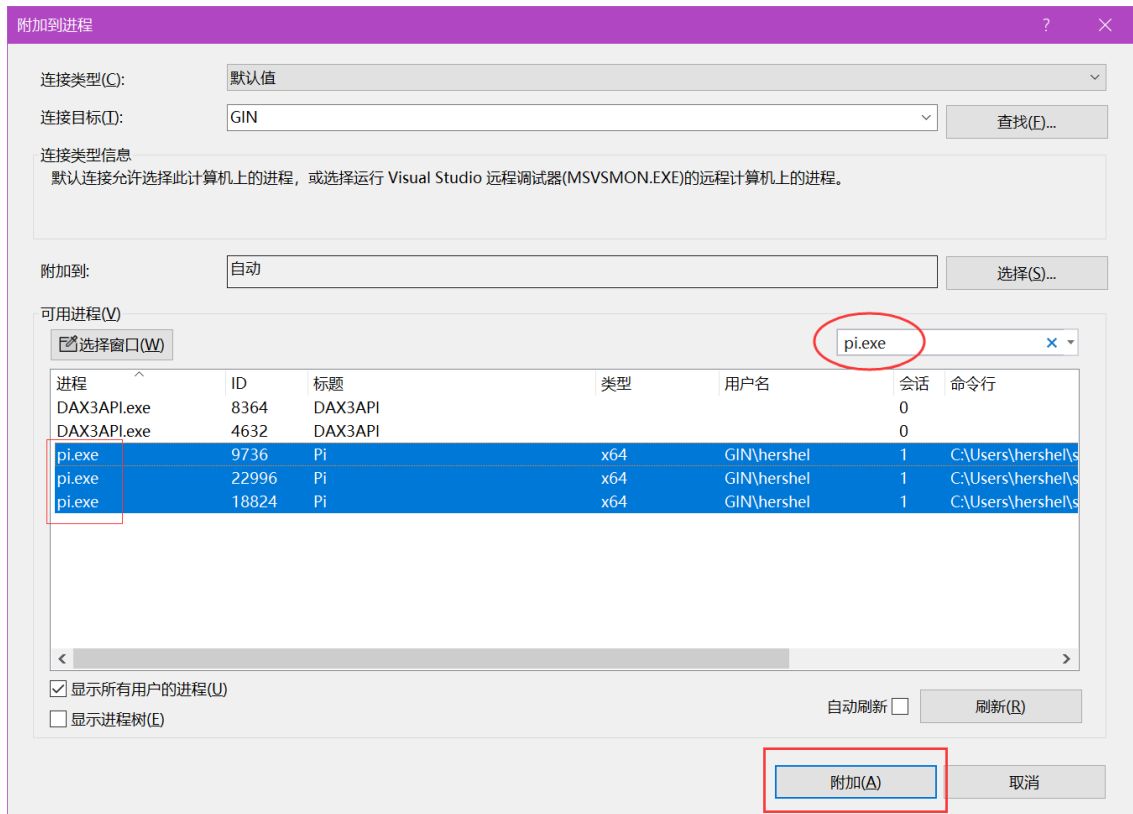
2. 在以上代码段之后添加断点，随后运行程序

```
15 MPI_Barrier(MPI_COMM_WORLD); // All threads will wait here until you give thread 0 an input
16
17 int tag = 666;
```

3. VS打开菜单栏的调试栏，选择附加到进程



4. 搜索对应进程名（项目名称），本次运行启动了三个进程，因此通过按住Shift选择全部的3个进程，并点击附加。



5. 向控制台窗口输入任意数字从而开始debug

More details

- [调试多个进程 - Visual Studio \(Windows\) | Microsoft Learn](#)
- [Debugging an MPI application with Microsoft Visual Studio | The Supercomputing Blog](#)

使用GDB调试MPI程序(Linux)

1. 在代码中设置一个无限循环来阻塞线程。

```
int main(int argc, char *argv[]) {
    int a=1;
    while(a==1);
    int count;
    /* Local prime counting */
}
```

2. 用“-g”(调试模式)编译代码并运行。

```
[root@cu01 2018081307023_李想]# mpic++ -g -o optimizer1 optimizer1.cpp
[root@cu01 2018081307023_李想]# mpirun -np 2 ./optimizer1 1000000
```

3. 建立新的连接, 获取线程pid:

```
[root@cu01 2018081307023_李想]# ps aux | grep optimizer1
[root@cu01 2018081307023_李想]# ps aux | grep optimizer1
root    25235  0.0  0.0  31360 1444 pts/2    S+   14:31   0:00 mpirun -np 2 ./optimiz
er1 1000000
root    25237 99.3  0.0  72932 4204 ?        Rs   14:31   1:08 ./optimizer1 1000000
root    25238 99.3  0.0  72932 4204 ?        Rs   14:31   1:08 ./optimizer1 1000000
root    25558  0.0  0.0  112652  960 pts/3    S+   14:33   0:00 grep --color=auto opti
mizer1
[root@cu01 2018081307023_李想]#
```

4. 用gdb连接pid。

```
[root@cu01 2018081307023_李懋]# gdb -q optimizer1 25237
[root@cu01 2018081307023_李懋]# gdb -q optimizer1 25238
```

有3个连接，一个用于代码运行，一个连接到pid 25237，一个连接到pid 25238。

```
1 我的服务器账号 x 2 我的服务器账号 x 3 我的服务器账号 x +
[root@cu01 2018081307023_李懋]# gdb -q optimizer1 25238
Reading symbols from /home/mpi_stu/2021_春/2018081307023_李懋/optimizer1...
Attaching to program: /home/mpi_stu/2021_春/2018081307023_李懋/optimizer1,
8
```

5. 设置值a=0 以跳出阻塞。在线程25238和25237中，a的值是非共享的，您需要将它们都设置为0。

```
(gdb) print a
$1 = 1
(gdb) set var a=0
(gdb) l
```

6. 然后就可以调试了。简单的调试方法如下：

- 使用l/list查看源代码。

```
(gdb) set var a=0
(gdb) l
10
11     #define MIN(a, b) ((a)<(b)?(a):(b))
12
13     int main(int argc, char *argv[]) {
14         int a=1;
15         while(a==1);
16         int count;           /* Local prime count 局部质数 */
17         double elapsed_time; /* Parallel execution time 并行执行时间 */
18         int first;           /* Index of first multiple 初倍数 */
19         int global_count;    /* Global prime count 全局质数 */
(gdb) l
20         int id;              /* Process ID number 进程ID */
21         int index;           /* Index of current prime 当前素数的索引 */
22         int high_value;      /* Highest value on this proc 此过程中的最大值 */
23         int low_value;       /* Lowest value on this proc 此过程中最小值 */
24         char *marked;        /* Portion of 2,...,'n' 部分 2,...,'n' */
25         int n;               /* Sieving from 2, ..., 'n' 筛除 2,..., 'n' */
26         int p;               /* Number of processes 进程数量 */
27         int proc0_size;      /* Size of proc 0's subarray proc0的子数组的大小 */
28         int prime;           /* Current prime 当前素数 */
29         int size;            /* Elements in 'marked' 标记的元素 */
(gdb)
```

- 使用b+行号添加断点。

```
32
33      // 初始化
34      // MPI程序启动时“自动”建立两个通信器:
35      // MPI_COMM_WORLD:包含程序中所有MPI进程
36      // MPI_COMM_SELF: 有单个进程独自构成, 仅包含自己
37      MPI_Init(&argc, &argv);
38
39      // MPI_COMM_RANK 得到本进程的进程号, 进程号取值范围为 0, ...
40      MPI_Comm_rank(MPI_COMM_WORLD, &id);
41
(gdb) b 40
Breakpoint 2 at 0x401e40: file optimizer1.cpp, line 40.
(gdb)
```

- 使用s/step或n/next逐步运行代码(这两个命令略有不同, “s”将跳转到函数中, 而“n”不会)

```
(gdb) step
37      MPI_Init(&argc, &argv);
(gdb) step
40      MPI_Comm_rank(MPI_COMM_WORLD, &id);
(gdb)
```

- 使用c/continue将代码运行到断点

```
(gdb) c
Continuing.

Breakpoint 1, main (argc=2, argv=0x7ffec42a5058) at optimizer1.cpp:40
40      MPI_Comm_rank(MPI_COMM_WORLD, &id);
(gdb) s
43      MPI_Comm_size(MPI_COMM_WORLD, &p);
```

- 以下是其他常见的gdb命令。

命令	缩写	描述
delete	d	删除断点
print	p	打印变量的值
run	r	运行程序
quit	q	退出gdb

- 通常, 线程需要一起调试, 否则可能会发生阻塞。有关其他调试技巧, 请参见参考资料。

参考:

gdb教程:

<https://www.cs.cmu.edu/~gilpin/tutorial/>

[Using gdb and ddd with MPI](#)

示例程序(π 计算)

思路

原理根据如下图所进行计算，就是通过定积分定义来进行计算。

$$\pi = \int_0^1 \frac{4}{x^2 + 1} dx \approx h \sum_{i=1}^n f(x_i)$$

➤ 采用等步长中矩形公式，其中 n 为积分区间数， $h = 1/n$ 为步长， $x_i = (i + 0.5)h$ 为积分区间的中点

代码

```
1  #include<iostream>
2  #include"mpi.h"
3  int main(int argc, char* argv[])
4  {
5      int myid, np, i, j;
6      MPI_Init(&argc, &argv); //启动并行程序
7      MPI_Comm_size(MPI_COMM_WORLD, &np); //获取进程总数
8      MPI_Comm_rank(MPI_COMM_WORLD, &myid); //获取当前进程号
9      //Just for debugger.if not,comment it.
10     int temp;
11     if (myid == 0)
12     {
13         std::cin >> temp;
14     }
15
16     MPI_Barrier(MPI_COMM_WORLD); // All threads will wait here until you
    give thread 0 an input
17 }
18
19 int tag = 666;
20 double pi = 0.0;
21 double fval; //fval代表取xi所对应的函数值 4/(1+x^2) 即每个矩形的高度
22 int n = 100000000;
23 MPI_Status status;
24 double h = (double)1 / n; //每个矩形的宽度
25 double local = 0.0; //每个进程计算的面积和
26 double start, end;
27 double xi;
28 if(myid==0)printf("np:%d\n", np);
29 start = MPI_Wtime(); //记录开始时间
30 for (i = myid; i < n; i += np) //利用np个进程同时计算各部分矩形面积
31 {
32     xi = (i + 0.5) * h;
33     fval = 4.0 / (1.0 + xi * xi); //得到f(xi)
34     local += fval;
35 }
36 local = local * h; //得到该进程所计算的面积
37
38 //进程号!=0的进程计算的结果发送到进程0上面
39 if (myid != 0)
40 {
41     MPI_Send(&local, 1, MPI_DOUBLE, 0, myid, MPI_COMM_WORLD);
42 }
```

```
43     if (myid == 0) //进程号为0就累加每个进程的计算结果
44     {
45         pi = local; //得到进程0的值 后面接收就会覆盖这个值
46         for (j = 1; j < np; j++)
47         {
48             MPI_Recv(&local, 1, MPI_DOUBLE, j, j, MPI_COMM_WORLD, &status);
//把其他进程的结果发送到local中
49             pi += local; //得到所有的面积和
50         }
51     }
52     end = MPI_wtime(); //结束计算时间
53     if (myid == 0)
54     {
55         printf("PI = %.15f\n", pi);
56         printf("Time = %lf\n", end - start);
57     }
58     MPI_Finalize();
59     return 0;
60 }
```