МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1
по курсу «Параллельная обработка данных»

Message Passing Interface (MPI)

Выполнил: А.В. Ефимов
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Знакомство с технологией MPI. Реализация метода Якоби.
Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Вариант: обмен граничными слоями через bsend, контроль сходимости allgather;

Программное и аппаратное обеспечение

```
##### CUDA Info

/opt/cuda/extras/demo_suite/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce MX150"
  CUDA Driver Version / Runtime Version          11.4 / 11.4
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 2003 MBytes (2099904512
bytes)
  ( 3) Multiprocessors, (128) CUDA Cores/MP:     384 CUDA Cores
  GPU Max Clock rate:                            1532 MHz (1.53 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 524288 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048
layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >
```

```
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.4, CUDA Runtime
Version = 11.4, NumDevs = 1, Device0 = NVIDIA GeForce MX150
Result = PASS

##### CPU Info

Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Address sizes:                   39 bits physical, 48 bits virtual
Byte Order:                      Little Endian
CPU(s):                          8
On-line CPU(s) list:             0-7
Vendor ID:                       GenuineIntel
Model name:                      Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
CPU family:                      6
Model:                           142
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
Stepping:                        10
CPU max MHz:                     3400.0000
CPU min MHz:                     400.0000
BogoMIPS:                        3601.00
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic
sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht
tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts
rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64
monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2
x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm
abm 3dnowprefetch cpuid_fault epb invpcid_single pti ibrs ibpb stibp
tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt
xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window
hwp_epp
Virtualization:                  VT-x
L1d cache:                       128 KiB (4 instances)
L1i cache:                       128 KiB (4 instances)
L2 cache:                        1 MiB (4 instances)
L3 cache:                        6 MiB (1 instance)
NUMA node(s):                    1
NUMA node0 CPU(s):               0-7
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:             Mitigation; PTE Inversion; VMX conditional
cache flushes, SMT vulnerable
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted,
no microcode; SMT vulnerable
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and
__user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB
conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:             Vulnerable: No microcode
Vulnerability Tsx async abort:   Not affected

##### RAM Info
               total          used          free        shared  buff/cache
available
Mem:            7.6Gi         1.7Gi         4.6Gi         633Mi         1.4Gi
5.1Gi
Swap:             0B           0B           0B
```

Метод решения

Вся область считается блоками, каждый процесс работает над своим блоком, причем длина каждой стороны блока увеличивается чтобы вместить краевые элементы – будь они из поставленных граничных условий или просто из других блоков.

Перед подсчетом сетки, краевые значения краевых блоков (т.е. блоков, обрабатываемые слотами, находящихся не в середине сетки слотов) инициализируются поставленными граничными условиями и меняются во время работы всей программы. Значения, стоящие в середине сетки, приравниваются поставленному начальному значению u_0.

Далее в цикле запускается весь процесс подсчета: каждый слот считает новые значения в своем блоке и определяет максимальное изменение значений, после чего все слоты обмениваются максимальными изменениями каждый в своем блоке (в этот момент также идет синхронизация слотов), из которых выбирается наибольшее изменение. Если оно в каком-то пределе достаточно маленькое, то процесс останавливается.

Если процесс не остановился, то блоки обмениваются своими краевыми значениями, которые нужны, чтобы слоты могли продолжиться изолированно считать новые значения в своем блоке.

Описание программы

Пример инициализации блока краевого слота:

```
if (proc_z == 0) {
    for (int j = 0; j < pad_block_size[y_dir]; ++j) {
        for (int i = 0; i < pad_block_size[x_dir]; ++i) {
            old_grid[calc_1d(i, j, 0)] = u[down_s];
            new_grid[calc_1d(i, j, 0)] = u[down_s];
        }
    }
}
```

Поиск максимального элементы во всех блоках:

```
MPI_Allgather(&max_diff, 1, MPI_DOUBLE, block_maxes, 1,
MPI_DOUBLE, MPI_COMM_WORLD);
for (int i = 0; i < proc_count; ++i) {
    max_diff = std::max(max_diff, block_maxes[i]);
}

if (max_diff < eps) {
    break;
}
```

Пример обмена между блоками:

```
if (proc_x < proc_size[x_dir] - 1) {
    for (int z = 1; z < pad_block_size[z_dir] - 1; ++z) {
        for (int y = 1; y < pad_block_size[y_dir] - 1; ++y) {
            edge_buff[calc_edge(y-1, z-1)] =
new_grid[calc_1d(pad_block_size[x_dir] - 2, y, z)];
        }
    }
    MPI_Bsend(edge_buff,
              edge_buff_size,
              MPI_DOUBLE,
              calc_rank(proc_x + 1, proc_y, proc_z),
              id,
              MPI_COMM_WORLD);
}
```

```
if (proc_x < proc_size[x_dir] - 1) {
    MPI_Recv(edge_buff, edge_buff_size, MPI_DOUBLE,
calc_rank(proc_x + 1, proc_y, proc_z), calc_rank(proc_x + 1,
proc_y, proc_z), MPI_COMM_WORLD, &status);
    for (int z = 1; z < pad_block_size[z_dir] - 1; ++z) {
        for (int y = 1; y < pad_block_size[y_dir] - 1; ++y) {
            new_grid[calc_1d(pad_block_size[x_dir] - 1, y, z)] =
edge_buff[calc_edge(y-1, z-1)];
        }
    }
}
```

Результаты

Скорость на CPU (в первом столбце – длина стороны блока, все блоки – кубические):

4	0.360906ms
8	9.37472ms
16	177.614ms
32	4895.84ms
64	141197ms

Далее представлена скорость на MPI, причем, из-за ограничения в 4 слота, сетка слотов увеличивалась по оси Z. Так как Каждый новый слот работает отдельно и увеличивает общую сетку, для компенсации размер блока по оси Z делился на количество слотов. Например, если запускается 4 слота с длиной куба 64, то блок имел размеры $64 \times 64 \times \frac{64}{4}$.

Один слот:

4	0.318497ms
8	6.59553ms
16	171.115ms
32	4877.48ms
64	139609ms

Два слота:

4	0.244499ms
8	5.27685ms
16	94.0098ms
32	2485.02ms
64	70925.6ms

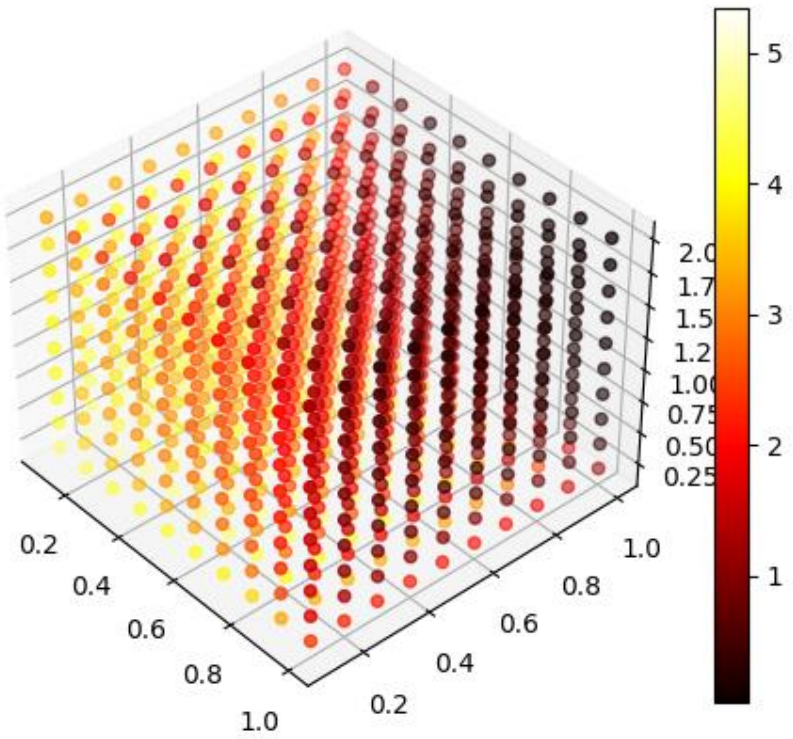
Три слота:

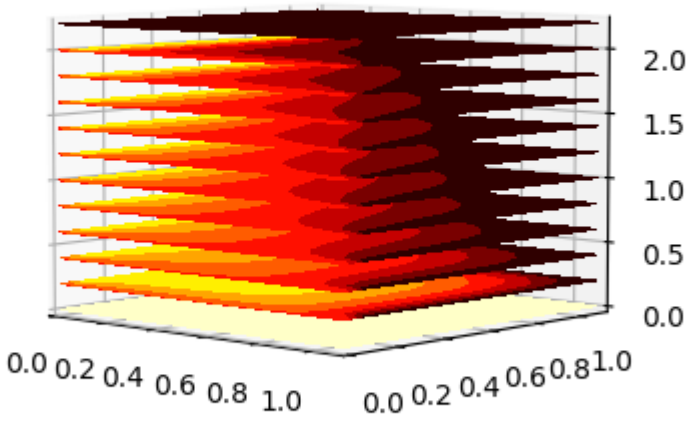
4	0.302748ms
8	2.70431ms
16	62.9896ms
32	3473.13ms
64	106768ms

Четыре слота:

4	0.515593ms
8	4.68929ms
16	69.6758ms
32	2672.35ms
64	90780.7ms

Результат для второго примера на кубе с длиной 10:





Выводы

- MPI запускает не потоки, а целые процессы. В то время, как потоки не параллелятся, процессы могут параллелиться, так как они запускаются каждый на своем процессоре.
- MPI может работать на более, чем одной вычислительной машине, например на кластере. Это особенно полезно при работе с большими данными, где вычислительной способности одной машины будет мало.