Multicore & GPU Programming : An Integrated Approach
Errata Document, Version 1.03

Gerassimos Barlas

February 13, 2017

1 Introduction

- Page 16, second paragraph, line 4: "increase only" should be "only increase".

2 Multicore and Parallel Program Design

- Page 30, Equation 2.4 should be:

$$totalComm = 8{\cdot}IMGX{\cdot}IMGY - 3{\cdot}2{\cdot}(IMGX-2) - 3{\cdot}2{\cdot}(IMGY-2) - 4{\cdot}5 = 8 \cdot IMGX \cdot IMGY - 6 \cdot (IMGX + IMGY) + 4$$

- Page 38, Equation 2.7 should be:

$$comp_{1D} = k \cdot N \cdot t_{comp} = \frac{N^2}{P} \cdot t_{comp}$$

3 Shared-memory programming : Threads

- Page 78, fourth line : "They only thing" should be "The only thing"
- Page 110, Section 3.7.1.2. A number of changes need to be made to Listing 3.21 and the associated text. To avoid confusion, the whole section in its edited form is given below:

  If the addition or removal of resources from the shared buffer takes a considerable amount of time (e.g. requires copying objects instead of references), using the second design approach can improve performance. Getting and releasing a permit from the monitor means that the run methods will be a bit longer than the miniscule ones of the previous design.

  The idea is that producers and consumers will use a pair of functions to first acquire exclusive access to a buffer's location, and second release the location back to the monitor to utilize.

  There is one major difference though: the acquisition and release of buffer elements potentially out-of-order, means that we can no longer treat the

buffer as a circular queue. For example, it would be possible to have a producer release an element prior to another producer that acquired one at an earlier time, thus allowing a consumer to get an item that is not there. For this reason, two separate queues have to be maintained, one for the free buffer elements (`emptySpotsQ`) and one for the occupied ones (`itemQ`).

Listing 1: A monitor-based solution to the producers-consumers problem, where the buffer elements are directly manipulated by the producer and consumer threads, under the supervision of the monitor. For the sake of brevity, only the differences with Listing 3.20 are shown.

```
1  // File : monitor2ProdCons.cpp
2  . . .
3  template<typename T>
4  class Monitor {
5  private:
6      QMutex l;
7      QWaitCondition full, empty;
8      queue<T *> emptySpotsQ;
9      queue<T *> itemQ;
10     T *buffer;
11 public:
12     T* canPut();
13     T* canGet();
14     void donePutting(T *x);
15     void doneGetting(T *x);
16     Monitor(int n = BUFFSIZE);
17     ~Monitor();
18 };
19 //-----------------------------------------
20
21 template<typename T>
22 Monitor<T>::Monitor(int n) {
23     buffer = new T[n];
24     for(int i=0;i<n;i++)
25         emptySpotsQ.push(&buffer[i]);
26 }
27 //-----------------------------------------
28
29 template<typename T>
30 T* Monitor<T>::canPut() {
31     QMutexLocker ml(&l);
32     while (emptySpotsQ.size() == 0)
33         full.wait(&l);
34     T *aux = emptySpotsQ.front();
35     emptySpotsQ.pop();
36     return aux;
37 }
38 //-----------------------------------------
39
40 template<typename T>
41 T* Monitor<T>::canGet() {
42     QMutexLocker ml(&l);
43     while (itemQ.size() == 0)
44         empty.wait(&l);
45     T* temp = itemQ.front();
46     itemQ.pop();
47     return temp;
48 }
49 //-----------------------------------------
50
51 template<typename T>
52 void Monitor<T>::donePutting(T *x) {
53     QMutexLocker ml(&l);
54     itemQ.push(x);
55     empty.wakeOne();
56 }
57 //-----------------------------------------
58
```

```
59  template<typename T>
60  void Monitor<T>::doneGetting(T *x) {
61      QMutexLocker ml(&l);
62      emptySpotsQ.push(x);
63      full.wakeOne();
64  }
65  //*************************************************************
66  . . .
67  template<typename T>
68  void Producer<T>::run() {
69      while (numProducts.tryAcquire()) {
70          T item = (*produce)();
71          T* aux = mon->canPut();
72          *aux = item;
73          mon->donePutting(aux);
74      }
75  }
76  //*************************************************************
77  . . .
78  template<typename T> void Consumer<T>::run() {
79      while (numProducts.tryAcquire()) {
80          T* aux = mon->canGet();
81          T item = *aux; // take the item out
82          mon->doneGetting(aux);
83          (*consume)(item);
84      }
85  }
```

The key points of the solution in Listing 3.21 are:

- The `Monitor` class provides two pairs of methods:
  - `canPut` and `donePutting` for `Producer` threads
  - `canGet` and `doneGetting` for `Consumer` threads

  The bodies of these methods essentially contain the two halves (unevenly divided) of the `put` and `get` methods respectively of Listing 3.20.
- The `canPut` and `canGet` methods return pointers to buffer locations that can be used for storage or retrieval of resources. The `in` and `out` and `count` variables are no longer necessary as they would introduce a race condition. Instead the typical FIFO queue operations `push`, `front`, and `pop` are used to hold and extract the buffer elements from queues `emptySpotsQ` and `itemQ`.
- All the addresses of the initially empty buffer elements are placed in the `emptySpotsQ` queue via the `Monitor` constructor (lines 24,25). The buffer is still allocated and freed as an array for convenience.
- The `Producer` and `Consumer` threads can take their time to store or extract a resource after the `canPut` and `canGet` methods return. The `Monitor` is able to serve other threads at that time.
- The `donePutting`/`doneGetting` methods require the address of the buffer element that was placed/removed from the buffer, as illustrated in lines 73 and 82 of the `run` methods. Once the corresponding queues are updated, a waiting `Consumer`/`Producer` is alerted via the `empty`/`full` wait conditions.

4 Shared-memory programming : OpenMP

- Page 171, second paragraph, line 2: $\int(x^2 + 2 \cdot sin(x)) = \frac{1}{3}(x^3 - 6 \cdot cos(x))$ should be $\int(x^2 + 2 \cdot sin(x))\ dx = \frac{1}{3}(x^3 - 6 \cdot cos(x))$
- Page 184, Listing 4.11 : For the code fragment to have the exact outcome as Listing 4.10, the following line should be added after the loop:

```
x[ N - 1 ] = x[ N - 1 ] + c[ N - 1 ];
```

- Page 207,fifth paragraph, 2nd line: "The list should contains data" should be "The list should contain data".
- Page 237, Exercise 3. The code fragment should read:

```
for (int i = 1; i < K-1; i++)
  for (int j = 0; j < M; j++)
      a[i][j] = a[i-1][j] + a[i+1][j];
```

5 Distributed memory programming

- Page 241 : "MPI communication primitive inside CUDA kernels" is actually a false statement. It should be corrected to "MPI communication primitives involving GPU memory".
- Page 242, "Data marshaling" paragraph: "big-indian and small-indian" should be "big-endian and small-endian".
- Page 246, Section 5.5.2, third paragraph : "separated by a semicolon" should be "separated by a colon", as it is visible in the following example.
- Page 248, second line from the bottom: "if a big-indian machine, like e.g. an i7 processor, were sending data to a little-indian machine" should be "if a little-indian machine, like e.g. an i7 processor, were sending data to a big-indian machine"
- Section 5.8, page 255: The first paragraph should read:

  In general, buffered sends as described in the previous section, are considered a bad practice because of the explicit need to perform a memory copy. A copy is not mandated by the normal send primitive. Performance can be enhanced if no copy ever takes place (as in the synchronous communication mode), or when we are allowed to continue execution without worrying about the progress of the communication. The latter is the domain of the "immediate", non-blocking functions, which strive to improve concurrency by overlapping communication and computation. The transition is as simple as using the `MPI_Isend` function in the place of `MPI_Send`.
- Page 258, 10th line from the bottom : "distinguishing" should be "distinguish".
- Page 262, Listing 5.9, lines 4-13. A simpler and faster variation of the `MSB` function is the following:

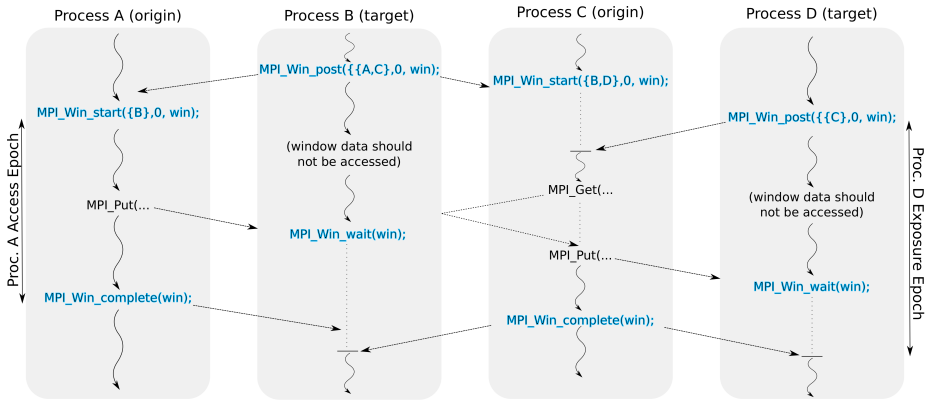

Figure 1: Correction for Figure 5.16

```
int MSB(int i) {
    int pos = 0;
    while (i != 0) {
        i >>= 1;
        pos++;
    }
    return pos-1;
}
```

- Page 272, last sentence of section 5.11.1 : "exact operation" should be "exact opposite operation".
- Page 272, signatures of functions `MPI_Gather` and `MPI_Gatherv`: the `sendbuf` and `sendcnt` are not ignored at the destination process, as the destination also contributes to the collected data.
- Page 278, line 48 in Listing 5.15 should be:

```
    locDiv = ceil (1.0 * divisions / N);
```

- Section 5.12, page 290 : "MPI provides two mechanisms that can be used to communicate structure between heterogeneous machines:" should be "MPI provides two mechanisms that can be used to communicate structures between heterogeneous machines:"
- Section 5.12, page 299, last two items in the list. Please consider the following clarifications:
  - **Derived datatypes** : if data are heterogeneous, and/or in non-contiguous but regularly spaced memory locations (e.g. parts of the rows of a matrix) , and are communicated often.
  - **Packing/Unpacking** : if data are heterogeneous, and/or in non-contiguous memory locations (e.g. referenced by pointers), and are not communicated often.
- Figure 5.16, page 311 : the `MPI_Win_complete` and `MPI_Win_wait` parameters should be fixed as shown in Figure 1

- Listing 5.19, pages 293-295: lines 42,43 and 78,79 can be safely removed because the program works only for K and L which are multiples of *procY* and *procX* respectively. The caption in page 295 should be augmented with the following: "The `K` and `L` matrix dimensions must be multiples of `procY` and `procX` respectively."

  The supplied source code file has an `assert` statement enforcing these attributes.

- Page 353, line 17 in the code listing should be:

```
ecb_crypt((char *)&k,(char *) ciph, len, DES_DECRYPT);
```

- Listing 5.25:
  - Line 23 should be: `int *data = new int[M];`
  - Line 48 should be: `MPI_Win_create (data, M * sizeof (int), sizeof (int), MPI_INFO_NULL, MPI_COMM_WORLD, &dataWin);`
  - Line 56 should be: `deliveredItems = (i == N - 1) ? (M - (N - 1) * maxItemsPerBucket) : maxItemsPerBucket;`

- Listing 5.27, line 27 should be: `MPI_Type_create_subarray (1, &sizes, &subsizes, &starts, MPI_ORDER_C, MPI_INT, &filetype);`

6 GPU Programming

- Unfortunately, through no fault of the author, the NVidia name has been changed to "Nvidia" thoughout the text of the chapter. Please excuse the annoyance.

- Section 6.3, page 400, first paragraph: "A GPU can contain multiple SMs, each running each own kernel." should be "A GPU can contain multiple SMs, each running its own kernel."

- Section 6.3, page 400, line 17: "the reason in performance" should be "the reason **is** performance".

- Section 6.3, page 402, first paragraph: "This means that there will be 12 SMs that will receive four blocks and six SMs that will receive three blocks." should be "This means that there will be 12 SMs that will receive four blocks and four SMs that will receive three blocks.

- Section 6.6, page 410, 3^{rd} paragraph should read: "In the Listings that follow, in order to keep track of what kind of memory each pointer references, we prefix the pointers to host memory with `h_` or `h`, and the pointers to device memory with `d_` or `d`."

- Page 415, Figure 6.8: The bandwidth between board memory and GPU is 200 GB/sec and not MB/sec.

- Page 423, 3rd text line from the bottom : "up up" should be "up".

- Page 423, 2nd text line from the bottom : "`unsigned char c[M]`" should be "`unsigned int c[M]`".

- Page 424, Listing 6.10, line 25 should be:

```
foo<<<1, 256, K*sizeof(int) + L*sizeof(double) + M*sizeof(↩
    unsigned int)>>>(da);
```

- Section 6.7, pages 441: modify line 49 of the `odd.cu` listing, as follows:

```
while (((localID | step) < blockDim.x) && ((localID & step) == 0)↩
    )
```

- Section 6.7.2, page 445, last line:"These are discussed in the next section." should be "These are discussed in the previous section."

- Page 448, Figure 6.14: The top row of numbers should be "0 1 2 3 32 33 34 35 64 65 66 67 96 97 98 99" instead of "0 1 2 3 32 33 34 35 64 67 34 35 96 97 98 99".

- Page 451, line 34 in Listing 6.21 : The comment should read "// Optimized version of localH[bankID + v * BLOCKSIZE]++"

- Page 458, Figure 6.17 caption : "Empty cells represent padding." should change to "Empty cells represent padding based on the assumption that the `__align__(16)` qualifier were used in the definition of `x`, `y`, etc.". In the lower left part of the figure, the $\lceil \frac{N \cdot sizeof(float)}{16} \rceil \cdot 16$ expression should be $\lceil \frac{N \cdot sizeof(int)}{16} \rceil \cdot 16$.

- Section 6.7.4, listing of page 458: The signature of the kernel should change from:

```
__global__ void distance2( float *x, float *y, unsigned char *↩
    colorR, unsigned char *colorG, unsigned char *colorB, int N↩
    )
```

  to

```
__global__ void distance2( int *x, int *y, float *dist, int N)
```

- Exercise 3, page 524: remove "subtraction" from the list of potential operators.

- Section 6.7.7, page 465, middle of the page: "This is a blocking function, that returns when all the pending commands in a stream are complete." should be "This is a non-blocking function, that returns immediately. The resources associated with the stream will be released when all its pending operations are complete. For synchronization purposes one should use the `cudaStreamSynchronize()` call:"

```
cudaError_t cudaStreamSynchronize (cudaStream_t stream) ; // ↩
    Stream identifier
```

- Section 6.7.7, page 467: line 34 of the listing should become:

```
    cudaFree(h_data[i]);
```

- Section 6.7.7, page 466 and page 467: The following two lines should be added before line 22 of the listing in the top of page 466, and line 29 in the listing of page 467.

```
cudaStreamSynchronize(str[0]);
cudaStreamSynchronize(str[1]);
```

The text in page 466 "The blocking calls of lines 22 and 23..." refers to the above two lines.

- Section 6.7.7.1, page 468: The following line should be inserted after line 21:

```
cudaStreamCreate (&str);
```

- Listing 6.40, page 515, line 70 should be:

```
MPI_Send (iobuf + pos, actualSize, MPI_UNSIGNED_CHAR, stat.↩
    MPI_SOURCE, TAG_DATA, MPI_COMM_WORLD);
```

7 The Thrust Template Library

- Page 533, 4th line : "`thrust:sequence` and `thrust:fill` functions" should be "`thrust::sequence` and `thrust::fill` functions"
- Section 7.4.2, page 540, "`thrust::is_sorted_until` : returns an iterator to the last position (exclusive) that the input vector is sorted." should be "`thrust::is_sorted_until` : returns an iterator to the first position in the input vector that is out-of-order. If the vector is sorted, the iterator points past the end of the vector."
- Exercise 8, page 572 :

```
return this.key < o.key;
```

should be

```
return this->key < o.key;
```

- Exercise 8, page 572 : The 10^8 requirement may be too big for some GPUs to handle, based on the available memory. Smaller array sizes can be used for testing as well.
- Exercise 9, page 572 : The proper section reference for the mathematical details of the Mandelbrot set is 3.8.2 (page 126), instead of 4.22.1.

8 Load Balancing

- Page 575, Section 8.1, line 3 : "this trend contines" should be "this trend continues".
- Page 580, line 10 in Listing 8.1 should be "`10 ...`" as following the array allocations, the image is actually read in the `R`,`G` and `B` arrays.
- Page 585, Figure 8.5 : the two shown meshes are regular but did not come out properly by the printing process (some vertical lines are missing).
- Page 602, second item in the itemized list: p_{cpu} should be 0.01sec/image.
- Page 628, Exercise 5 : The speed of the first node should be $p_0 = 4 \cdot 1.631 \cdot 10^{-07}$.

9 Appendix C

- Page 638, Section C.4 : the function calls in the sample code should be `omp_get_wtime` and not `mpi_get_wtime`.