**Exercise 5.5**

For the tiled matrix multiplication kernel in Fig. 5.6, draw the access patterns of threads in a warp of Lines 9 and 10 for a small 16×16 matrix size. Calculate the tx values and ty values for each thread in a warp and use these values in the M and N index calculations in Lines 9 and 10. Show that the threads indeed access consecutive M and N locations in global memory during each iteration.

**Solution:**

Assuming tile width of 16 as we are processing 16\*16 matrix and there is lack of boundary condition check in Fig. 5.6.

**Figure 5.5.1 Arrangement of threads in block(0,0)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (0,0) | (0,1) | (0,2) | (0,3) | (0,4) | (0,5) | (0,6) | (0,7) | (0,8) | (0,9) | (0,10) | (0,11) | (0,12) | (0,13) | (0,14) | (0,15) |
| (1,0) | (1,1) | (1,2) | (1,3) | (1,4) | (1,5) | (1,6) | (1,7) | (1,8) | (1,9) | (1,10) | (1,11) | (1,12) | (1,13) | (1,14) | (1,15) |
| (2,0) | (2,1) | (2,2) | (2,3) | (2,4) | (2,5) | (2,6) | (2,7) | (2,8) | (2,9) | (2,10) | (2,11) | (2,12) | (2,13) | (2,14) | (2,15) |
| (3,0) | (3,1) | (3,2) | (3,3) | (3,4) | (3,5) | (3,6) | (3,7) | (3,8) | (3,9) | (3,10) | (3,11) | (3,12) | (3,13) | (3,14) | (3,15) |
| (4,0) | (4,1) | (4,2) | (4,3) | (4,4) | (4,5) | (4,6) | (4,7) | (4,8) | (4,9) | (4,10) | (4,11) | (4,12) | (4,13) | (4,14) | (4,15) |
| (5,0) | (5,1) | (5,2) | (5,3) | (5,4) | (5,5) | (5,6) | (5,7) | (5,8) | (5,9) | (5,10) | (5,11) | (5,12) | (5,13) | (5,14) | (5,15) |
| (6,0) | (6,1) | (6,2) | (6,3) | (6,4) | (6,5) | (6,6) | (6,7) | (6,8) | (6,9) | (6,10) | (6,11) | (6,12) | (6,13) | (6,14) | (6,15) |
| (7,0) | (7,1) | (7,2) | (7,3) | (7,4) | (7,5) | (7,6) | (7,7) | (7,8) | (7,9) | (7,10) | (7,11) | (7,12) | (7,13) | (7,14) | (7,15) |
| (8,0) | (8,1) | (8,2) | (8,3) | (8,4) | (8,5) | (8,6) | (8,7) | (8,8) | (8,9) | (8,10) | (8,11) | (8,12) | (8,13) | (8,14) | (8,15) |
| (9,0) | (9,1) | (9,2) | (9,3) | (9,4) | (9,5) | (9,6) | (9,7) | (9,8) | (9,9) | (9,10) | (9,11) | (9,12) | (9,13) | (9,14) | (9,15) |
| (10,0) | (10,1) | (10,2) | (10,3) | (10,4) | (10,5) | (10,6) | (10,7) | (10,8) | (10,9) | (10,10) | (10,11) | (10,12) | (10,13) | (10,14) | (10,15) |
| (11,0) | (11,1) | (11,2) | (11,3) | (11,4) | (11,5) | (11,6) | (11,7) | (11,8) | (11,9) | (11,10) | (11,11) | (11,12) | (11,13) | (11,14) | (11,15) |
| (12,0) | (12,1) | (12,2) | (12,3) | (12,4) | (12,5) | (12,6) | (12,7) | (12,8) | (12,9) | (12,10) | (12,11) | (12,12) | (12,13) | (12,14) | (12,15) |
| (13,0) | (13,1) | (13,2) | (13,3) | (13,4) | (13,5) | (13,6) | (13,7) | (13,8) | (13,9) | (13,10) | (13,11) | (13,12) | (13,13) | (13,14) | (13,15) |
| (14,0) | (14,1) | (14,2) | (14,3) | (14,4) | (14,5) | (14,6) | (14,7) | (14,8) | (14,9) | (14,10) | (14,11) | (14,12) | (14,13) | (14,14) | (14,15) |
| (15,0) | (15,1) | (15,2) | (15,3) | (15,4) | (15,5) | (15,6) | (15,7) | (15,8) | (15,9) | (15,10) | (15,11) | (15,12) | (15,13) | (15,14) | (15,15) |

**Access pattern for threads in each warp:**

Warp 1:

Thread( 0, 0) Mds[ 0][ 0] = M[ 0] Nds[ 0][ 0] = N[ 0]

Thread( 0, 1) Mds[ 0][ 1] = M[ 1] Nds[ 0][ 1] = N[ 1]

Thread( 0, 2) Mds[ 0][ 2] = M[ 2] Nds[ 0][ 2] = N[ 2]

Thread( 0, 3) Mds[ 0][ 3] = M[ 3] Nds[ 0][ 3] = N[ 3]

Thread( 0, 4) Mds[ 0][ 4] = M[ 4] Nds[ 0][ 4] = N[ 4]

Thread( 0, 5) Mds[ 0][ 5] = M[ 5] Nds[ 0][ 5] = N[ 5]

Thread( 0, 6) Mds[ 0][ 6] = M[ 6] Nds[ 0][ 6] = N[ 6]

Thread( 0, 7) Mds[ 0][ 7] = M[ 7] Nds[ 0][ 7] = N[ 7]

Thread( 0, 8) Mds[ 0][ 8] = M[ 8] Nds[ 0][ 8] = N[ 8]

Thread( 0, 9) Mds[ 0][ 9] = M[ 9] Nds[ 0][ 9] = N[ 9]

Thread( 0,10) Mds[ 0][10] = M[ 10] Nds[ 0][10] = N[ 10]

Thread( 0,11) Mds[ 0][11] = M[ 11] Nds[ 0][11] = N[ 11]

Thread( 0,12) Mds[ 0][12] = M[ 12] Nds[ 0][12] = N[ 12]

Thread( 0,13) Mds[ 0][13] = M[ 13] Nds[ 0][13] = N[ 13]

Thread( 0,14) Mds[ 0][14] = M[ 14] Nds[ 0][14] = N[ 14]

Thread( 0,15) Mds[ 0][15] = M[ 15] Nds[ 0][15] = N[ 15]

Thread( 1, 0) Mds[ 1][ 0] = M[ 16] Nds[ 1][ 0] = N[ 16]

Thread( 1, 1) Mds[ 1][ 1] = M[ 17] Nds[ 1][ 1] = N[ 17]

Thread( 1, 2) Mds[ 1][ 2] = M[ 18] Nds[ 1][ 2] = N[ 18]

Thread( 1, 3) Mds[ 1][ 3] = M[ 19] Nds[ 1][ 3] = N[ 19]

Thread( 1, 4) Mds[ 1][ 4] = M[ 20] Nds[ 1][ 4] = N[ 20]

Thread( 1, 5) Mds[ 1][ 5] = M[ 21] Nds[ 1][ 5] = N[ 21]

Thread( 1, 6) Mds[ 1][ 6] = M[ 22] Nds[ 1][ 6] = N[ 22]

Thread( 1, 7) Mds[ 1][ 7] = M[ 23] Nds[ 1][ 7] = N[ 23]

Thread( 1, 8) Mds[ 1][ 8] = M[ 24] Nds[ 1][ 8] = N[ 24]

Thread( 1, 9) Mds[ 1][ 9] = M[ 25] Nds[ 1][ 9] = N[ 25]

Thread( 1,10) Mds[ 1][10] = M[ 26] Nds[ 1][10] = N[ 26]

Thread( 1,11) Mds[ 1][11] = M[ 27] Nds[ 1][11] = N[ 27]

Thread( 1,12) Mds[ 1][12] = M[ 28] Nds[ 1][12] = N[ 28]

Thread( 1,13) Mds[ 1][13] = M[ 29] Nds[ 1][13] = N[ 29]

Thread( 1,14) Mds[ 1][14] = M[ 30] Nds[ 1][14] = N[ 30]

Thread( 1,15) Mds[ 1][15] = M[ 31] Nds[ 1][15] = N[ 31]

Warp 2:

Thread( 2, 0) Mds[ 2][ 0] = M[ 32] Nds[ 2][ 0] = N[ 32]

Thread( 2, 1) Mds[ 2][ 1] = M[ 33] Nds[ 2][ 1] = N[ 33]

Thread( 2, 2) Mds[ 2][ 2] = M[ 34] Nds[ 2][ 2] = N[ 34]

Thread( 2, 3) Mds[ 2][ 3] = M[ 35] Nds[ 2][ 3] = N[ 35]

Thread( 2, 4) Mds[ 2][ 4] = M[ 36] Nds[ 2][ 4] = N[ 36]

Thread( 2, 5) Mds[ 2][ 5] = M[ 37] Nds[ 2][ 5] = N[ 37]

Thread( 2, 6) Mds[ 2][ 6] = M[ 38] Nds[ 2][ 6] = N[ 38]

Thread( 2, 7) Mds[ 2][ 7] = M[ 39] Nds[ 2][ 7] = N[ 39]

Thread( 2, 8) Mds[ 2][ 8] = M[ 40] Nds[ 2][ 8] = N[ 40]

Thread( 2, 9) Mds[ 2][ 9] = M[ 41] Nds[ 2][ 9] = N[ 41]

Thread( 2,10) Mds[ 2][10] = M[ 42] Nds[ 2][10] = N[ 42]

Thread( 2,11) Mds[ 2][11] = M[ 43] Nds[ 2][11] = N[ 43]

Thread( 2,12) Mds[ 2][12] = M[ 44] Nds[ 2][12] = N[ 44]

Thread( 2,13) Mds[ 2][13] = M[ 45] Nds[ 2][13] = N[ 45]

Thread( 2,14) Mds[ 2][14] = M[ 46] Nds[ 2][14] = N[ 46]

Thread( 2,15) Mds[ 2][15] = M[ 47] Nds[ 2][15] = N[ 47]

Thread( 3, 0) Mds[ 3][ 0] = M[ 48] Nds[ 3][ 0] = N[ 48]

Thread( 3, 1) Mds[ 3][ 1] = M[ 49] Nds[ 3][ 1] = N[ 49]

Thread( 3, 2) Mds[ 3][ 2] = M[ 50] Nds[ 3][ 2] = N[ 50]

Thread( 3, 3) Mds[ 3][ 3] = M[ 51] Nds[ 3][ 3] = N[ 51]

Thread( 3, 4) Mds[ 3][ 4] = M[ 52] Nds[ 3][ 4] = N[ 52]

Thread( 3, 5) Mds[ 3][ 5] = M[ 53] Nds[ 3][ 5] = N[ 53]

Thread( 3, 6) Mds[ 3][ 6] = M[ 54] Nds[ 3][ 6] = N[ 54]

Thread( 3, 7) Mds[ 3][ 7] = M[ 55] Nds[ 3][ 7] = N[ 55]

Thread( 3, 8) Mds[ 3][ 8] = M[ 56] Nds[ 3][ 8] = N[ 56]

Thread( 3, 9) Mds[ 3][ 9] = M[ 57] Nds[ 3][ 9] = N[ 57]

Thread( 3,10) Mds[ 3][10] = M[ 58] Nds[ 3][10] = N[ 58]

Thread( 3,11) Mds[ 3][11] = M[ 59] Nds[ 3][11] = N[ 59]

Thread( 3,12) Mds[ 3][12] = M[ 60] Nds[ 3][12] = N[ 60]

Thread( 3,13) Mds[ 3][13] = M[ 61] Nds[ 3][13] = N[ 61]

Thread( 3,14) Mds[ 3][14] = M[ 62] Nds[ 3][14] = N[ 62]

Thread( 3,15) Mds[ 3][15] = M[ 63] Nds[ 3][15] = N[ 63]

Warp 3:

Thread( 4, 0) Mds[ 4][ 0] = M[ 64] Nds[ 4][ 0] = N[ 64]

Thread( 4, 1) Mds[ 4][ 1] = M[ 65] Nds[ 4][ 1] = N[ 65]

Thread( 4, 2) Mds[ 4][ 2] = M[ 66] Nds[ 4][ 2] = N[ 66]

Thread( 4, 3) Mds[ 4][ 3] = M[ 67] Nds[ 4][ 3] = N[ 67]

Thread( 4, 4) Mds[ 4][ 4] = M[ 68] Nds[ 4][ 4] = N[ 68]

Thread( 4, 5) Mds[ 4][ 5] = M[ 69] Nds[ 4][ 5] = N[ 69]

Thread( 4, 6) Mds[ 4][ 6] = M[ 70] Nds[ 4][ 6] = N[ 70]

Thread( 4, 7) Mds[ 4][ 7] = M[ 71] Nds[ 4][ 7] = N[ 71]

Thread( 4, 8) Mds[ 4][ 8] = M[ 72] Nds[ 4][ 8] = N[ 72]

Thread( 4, 9) Mds[ 4][ 9] = M[ 73] Nds[ 4][ 9] = N[ 73]

Thread( 4,10) Mds[ 4][10] = M[ 74] Nds[ 4][10] = N[ 74]

Thread( 4,11) Mds[ 4][11] = M[ 75] Nds[ 4][11] = N[ 75]

Thread( 4,12) Mds[ 4][12] = M[ 76] Nds[ 4][12] = N[ 76]

Thread( 4,13) Mds[ 4][13] = M[ 77] Nds[ 4][13] = N[ 77]

Thread( 4,14) Mds[ 4][14] = M[ 78] Nds[ 4][14] = N[ 78]

Thread( 4,15) Mds[ 4][15] = M[ 79] Nds[ 4][15] = N[ 79]

Thread( 5, 0) Mds[ 5][ 0] = M[ 80] Nds[ 5][ 0] = N[ 80]

Thread( 5, 1) Mds[ 5][ 1] = M[ 81] Nds[ 5][ 1] = N[ 81]

Thread( 5, 2) Mds[ 5][ 2] = M[ 82] Nds[ 5][ 2] = N[ 82]

Thread( 5, 3) Mds[ 5][ 3] = M[ 83] Nds[ 5][ 3] = N[ 83]

Thread( 5, 4) Mds[ 5][ 4] = M[ 84] Nds[ 5][ 4] = N[ 84]

Thread( 5, 5) Mds[ 5][ 5] = M[ 85] Nds[ 5][ 5] = N[ 85]

Thread( 5, 6) Mds[ 5][ 6] = M[ 86] Nds[ 5][ 6] = N[ 86]

Thread( 5, 7) Mds[ 5][ 7] = M[ 87] Nds[ 5][ 7] = N[ 87]

Thread( 5, 8) Mds[ 5][ 8] = M[ 88] Nds[ 5][ 8] = N[ 88]

Thread( 5, 9) Mds[ 5][ 9] = M[ 89] Nds[ 5][ 9] = N[ 89]

Thread( 5,10) Mds[ 5][10] = M[ 90] Nds[ 5][10] = N[ 90]

Thread( 5,11) Mds[ 5][11] = M[ 91] Nds[ 5][11] = N[ 91]

Thread( 5,12) Mds[ 5][12] = M[ 92] Nds[ 5][12] = N[ 92]

Thread( 5,13) Mds[ 5][13] = M[ 93] Nds[ 5][13] = N[ 93]

Thread( 5,14) Mds[ 5][14] = M[ 94] Nds[ 5][14] = N[ 94]

Thread( 5,15) Mds[ 5][15] = M[ 95] Nds[ 5][15] = N[ 95]

Warp 4:

Thread( 6, 0) Mds[ 6][ 0] = M[ 96] Nds[ 6][ 0] = N[ 96]

Thread( 6, 1) Mds[ 6][ 1] = M[ 97] Nds[ 6][ 1] = N[ 97]

Thread( 6, 2) Mds[ 6][ 2] = M[ 98] Nds[ 6][ 2] = N[ 98]

Thread( 6, 3) Mds[ 6][ 3] = M[ 99] Nds[ 6][ 3] = N[ 99]

Thread( 6, 4) Mds[ 6][ 4] = M[100] Nds[ 6][ 4] = N[100]

Thread( 6, 5) Mds[ 6][ 5] = M[101] Nds[ 6][ 5] = N[101]

Thread( 6, 6) Mds[ 6][ 6] = M[102] Nds[ 6][ 6] = N[102]

Thread( 6, 7) Mds[ 6][ 7] = M[103] Nds[ 6][ 7] = N[103]

Thread( 6, 8) Mds[ 6][ 8] = M[104] Nds[ 6][ 8] = N[104]

Thread( 6, 9) Mds[ 6][ 9] = M[105] Nds[ 6][ 9] = N[105]

Thread( 6,10) Mds[ 6][10] = M[106] Nds[ 6][10] = N[106]

Thread( 6,11) Mds[ 6][11] = M[107] Nds[ 6][11] = N[107]

Thread( 6,12) Mds[ 6][12] = M[108] Nds[ 6][12] = N[108]

Thread( 6,13) Mds[ 6][13] = M[109] Nds[ 6][13] = N[109]

Thread( 6,14) Mds[ 6][14] = M[110] Nds[ 6][14] = N[110]

Thread( 6,15) Mds[ 6][15] = M[111] Nds[ 6][15] = N[111]

Thread( 7, 0) Mds[ 7][ 0] = M[112] Nds[ 7][ 0] = N[112]

Thread( 7, 1) Mds[ 7][ 1] = M[113] Nds[ 7][ 1] = N[113]

Thread( 7, 2) Mds[ 7][ 2] = M[114] Nds[ 7][ 2] = N[114]

Thread( 7, 3) Mds[ 7][ 3] = M[115] Nds[ 7][ 3] = N[115]

Thread( 7, 4) Mds[ 7][ 4] = M[116] Nds[ 7][ 4] = N[116]

Thread( 7, 5) Mds[ 7][ 5] = M[117] Nds[ 7][ 5] = N[117]

Thread( 7, 6) Mds[ 7][ 6] = M[118] Nds[ 7][ 6] = N[118]

Thread( 7, 7) Mds[ 7][ 7] = M[119] Nds[ 7][ 7] = N[119]

Thread( 7, 8) Mds[ 7][ 8] = M[120] Nds[ 7][ 8] = N[120]

Thread( 7, 9) Mds[ 7][ 9] = M[121] Nds[ 7][ 9] = N[121]

Thread( 7,10) Mds[ 7][10] = M[122] Nds[ 7][10] = N[122]

Thread( 7,11) Mds[ 7][11] = M[123] Nds[ 7][11] = N[123]

Thread( 7,12) Mds[ 7][12] = M[124] Nds[ 7][12] = N[124]

Thread( 7,13) Mds[ 7][13] = M[125] Nds[ 7][13] = N[125]

Thread( 7,14) Mds[ 7][14] = M[126] Nds[ 7][14] = N[126]

Thread( 7,15) Mds[ 7][15] = M[127] Nds[ 7][15] = N[127]

Warp 5:

Thread( 8, 0) Mds[ 8][ 0] = M[128] Nds[ 8][ 0] = N[128]

Thread( 8, 1) Mds[ 8][ 1] = M[129] Nds[ 8][ 1] = N[129]

Thread( 8, 2) Mds[ 8][ 2] = M[130] Nds[ 8][ 2] = N[130]

Thread( 8, 3) Mds[ 8][ 3] = M[131] Nds[ 8][ 3] = N[131]

Thread( 8, 4) Mds[ 8][ 4] = M[132] Nds[ 8][ 4] = N[132]

Thread( 8, 5) Mds[ 8][ 5] = M[133] Nds[ 8][ 5] = N[133]

Thread( 8, 6) Mds[ 8][ 6] = M[134] Nds[ 8][ 6] = N[134]

Thread( 8, 7) Mds[ 8][ 7] = M[135] Nds[ 8][ 7] = N[135]

Thread( 8, 8) Mds[ 8][ 8] = M[136] Nds[ 8][ 8] = N[136]

Thread( 8, 9) Mds[ 8][ 9] = M[137] Nds[ 8][ 9] = N[137]

Thread( 8,10) Mds[ 8][10] = M[138] Nds[ 8][10] = N[138]

Thread( 8,11) Mds[ 8][11] = M[139] Nds[ 8][11] = N[139]

Thread( 8,12) Mds[ 8][12] = M[140] Nds[ 8][12] = N[140]

Thread( 8,13) Mds[ 8][13] = M[141] Nds[ 8][13] = N[141]

Thread( 8,14) Mds[ 8][14] = M[142] Nds[ 8][14] = N[142]

Thread( 8,15) Mds[ 8][15] = M[143] Nds[ 8][15] = N[143]

Thread( 9, 0) Mds[ 9][ 0] = M[144] Nds[ 9][ 0] = N[144]

Thread( 9, 1) Mds[ 9][ 1] = M[145] Nds[ 9][ 1] = N[145]

Thread( 9, 2) Mds[ 9][ 2] = M[146] Nds[ 9][ 2] = N[146]

Thread( 9, 3) Mds[ 9][ 3] = M[147] Nds[ 9][ 3] = N[147]

Thread( 9, 4) Mds[ 9][ 4] = M[148] Nds[ 9][ 4] = N[148]

Thread( 9, 5) Mds[ 9][ 5] = M[149] Nds[ 9][ 5] = N[149]

Thread( 9, 6) Mds[ 9][ 6] = M[150] Nds[ 9][ 6] = N[150]

Thread( 9, 7) Mds[ 9][ 7] = M[151] Nds[ 9][ 7] = N[151]

Thread( 9, 8) Mds[ 9][ 8] = M[152] Nds[ 9][ 8] = N[152]

Thread( 9, 9) Mds[ 9][ 9] = M[153] Nds[ 9][ 9] = N[153]

Thread( 9,10) Mds[ 9][10] = M[154] Nds[ 9][10] = N[154]

Thread( 9,11) Mds[ 9][11] = M[155] Nds[ 9][11] = N[155]

Thread( 9,12) Mds[ 9][12] = M[156] Nds[ 9][12] = N[156]

Thread( 9,13) Mds[ 9][13] = M[157] Nds[ 9][13] = N[157]

Thread( 9,14) Mds[ 9][14] = M[158] Nds[ 9][14] = N[158]

Thread( 9,15) Mds[ 9][15] = M[159] Nds[ 9][15] = N[159]

Warp 6:

Thread(10, 0) Mds[10][ 0] = M[160] Nds[10][ 0] = N[160]

Thread(10, 1) Mds[10][ 1] = M[161] Nds[10][ 1] = N[161]

Thread(10, 2) Mds[10][ 2] = M[162] Nds[10][ 2] = N[162]

Thread(10, 3) Mds[10][ 3] = M[163] Nds[10][ 3] = N[163]

Thread(10, 4) Mds[10][ 4] = M[164] Nds[10][ 4] = N[164]

Thread(10, 5) Mds[10][ 5] = M[165] Nds[10][ 5] = N[165]

Thread(10, 6) Mds[10][ 6] = M[166] Nds[10][ 6] = N[166]

Thread(10, 7) Mds[10][ 7] = M[167] Nds[10][ 7] = N[167]

Thread(10, 8) Mds[10][ 8] = M[168] Nds[10][ 8] = N[168]

Thread(10, 9) Mds[10][ 9] = M[169] Nds[10][ 9] = N[169]

Thread(10,10) Mds[10][10] = M[170] Nds[10][10] = N[170]

Thread(10,11) Mds[10][11] = M[171] Nds[10][11] = N[171]

Thread(10,12) Mds[10][12] = M[172] Nds[10][12] = N[172]

Thread(10,13) Mds[10][13] = M[173] Nds[10][13] = N[173]

Thread(10,14) Mds[10][14] = M[174] Nds[10][14] = N[174]

Thread(10,15) Mds[10][15] = M[175] Nds[10][15] = N[175]

Thread(11, 0) Mds[11][ 0] = M[176] Nds[11][ 0] = N[176]

Thread(11, 1) Mds[11][ 1] = M[177] Nds[11][ 1] = N[177]

Thread(11, 2) Mds[11][ 2] = M[178] Nds[11][ 2] = N[178]

Thread(11, 3) Mds[11][ 3] = M[179] Nds[11][ 3] = N[179]

Thread(11, 4) Mds[11][ 4] = M[180] Nds[11][ 4] = N[180]

Thread(11, 5) Mds[11][ 5] = M[181] Nds[11][ 5] = N[181]

Thread(11, 6) Mds[11][ 6] = M[182] Nds[11][ 6] = N[182]

Thread(11, 7) Mds[11][ 7] = M[183] Nds[11][ 7] = N[183]

Thread(11, 8) Mds[11][ 8] = M[184] Nds[11][ 8] = N[184]

Thread(11, 9) Mds[11][ 9] = M[185] Nds[11][ 9] = N[185]

Thread(11,10) Mds[11][10] = M[186] Nds[11][10] = N[186]

Thread(11,11) Mds[11][11] = M[187] Nds[11][11] = N[187]

Thread(11,12) Mds[11][12] = M[188] Nds[11][12] = N[188]

Thread(11,13) Mds[11][13] = M[189] Nds[11][13] = N[189]

Thread(11,14) Mds[11][14] = M[190] Nds[11][14] = N[190]

Thread(11,15) Mds[11][15] = M[191] Nds[11][15] = N[191]

Warp 7:

Thread(12, 0) Mds[12][ 0] = M[192] Nds[12][ 0] = N[192]

Thread(12, 1) Mds[12][ 1] = M[193] Nds[12][ 1] = N[193]

Thread(12, 2) Mds[12][ 2] = M[194] Nds[12][ 2] = N[194]

Thread(12, 3) Mds[12][ 3] = M[195] Nds[12][ 3] = N[195]

Thread(12, 4) Mds[12][ 4] = M[196] Nds[12][ 4] = N[196]

Thread(12, 5) Mds[12][ 5] = M[197] Nds[12][ 5] = N[197]

Thread(12, 6) Mds[12][ 6] = M[198] Nds[12][ 6] = N[198]

Thread(12, 7) Mds[12][ 7] = M[199] Nds[12][ 7] = N[199]

Thread(12, 8) Mds[12][ 8] = M[200] Nds[12][ 8] = N[200]

Thread(12, 9) Mds[12][ 9] = M[201] Nds[12][ 9] = N[201]

Thread(12,10) Mds[12][10] = M[202] Nds[12][10] = N[202]

Thread(12,11) Mds[12][11] = M[203] Nds[12][11] = N[203]

Thread(12,12) Mds[12][12] = M[204] Nds[12][12] = N[204]

Thread(12,13) Mds[12][13] = M[205] Nds[12][13] = N[205]

Thread(12,14) Mds[12][14] = M[206] Nds[12][14] = N[206]

Thread(12,15) Mds[12][15] = M[207] Nds[12][15] = N[207]

Thread(13, 0) Mds[13][ 0] = M[208] Nds[13][ 0] = N[208]

Thread(13, 1) Mds[13][ 1] = M[209] Nds[13][ 1] = N[209]

Thread(13, 2) Mds[13][ 2] = M[210] Nds[13][ 2] = N[210]

Thread(13, 3) Mds[13][ 3] = M[211] Nds[13][ 3] = N[211]

Thread(13, 4) Mds[13][ 4] = M[212] Nds[13][ 4] = N[212]

Thread(13, 5) Mds[13][ 5] = M[213] Nds[13][ 5] = N[213]

Thread(13, 6) Mds[13][ 6] = M[214] Nds[13][ 6] = N[214]

Thread(13, 7) Mds[13][ 7] = M[215] Nds[13][ 7] = N[215]

Thread(13, 8) Mds[13][ 8] = M[216] Nds[13][ 8] = N[216]

Thread(13, 9) Mds[13][ 9] = M[217] Nds[13][ 9] = N[217]

Thread(13,10) Mds[13][10] = M[218] Nds[13][10] = N[218]

Thread(13,11) Mds[13][11] = M[219] Nds[13][11] = N[219]

Thread(13,12) Mds[13][12] = M[220] Nds[13][12] = N[220]

Thread(13,13) Mds[13][13] = M[221] Nds[13][13] = N[221]

Thread(13,14) Mds[13][14] = M[222] Nds[13][14] = N[222]

Thread(13,15) Mds[13][15] = M[223] Nds[13][15] = N[223]

Warp 8:

Thread(14, 0) Mds[14][ 0] = M[224] Nds[14][ 0] = N[224]

Thread(14, 1) Mds[14][ 1] = M[225] Nds[14][ 1] = N[225]

Thread(14, 2) Mds[14][ 2] = M[226] Nds[14][ 2] = N[226]

Thread(14, 3) Mds[14][ 3] = M[227] Nds[14][ 3] = N[227]

Thread(14, 4) Mds[14][ 4] = M[228] Nds[14][ 4] = N[228]

Thread(14, 5) Mds[14][ 5] = M[229] Nds[14][ 5] = N[229]

Thread(14, 6) Mds[14][ 6] = M[230] Nds[14][ 6] = N[230]

Thread(14, 7) Mds[14][ 7] = M[231] Nds[14][ 7] = N[231]

Thread(14, 8) Mds[14][ 8] = M[232] Nds[14][ 8] = N[232]

Thread(14, 9) Mds[14][ 9] = M[233] Nds[14][ 9] = N[233]

Thread(14,10) Mds[14][10] = M[234] Nds[14][10] = N[234]

Thread(14,11) Mds[14][11] = M[235] Nds[14][11] = N[235]

Thread(14,12) Mds[14][12] = M[236] Nds[14][12] = N[236]

Thread(14,13) Mds[14][13] = M[237] Nds[14][13] = N[237]

Thread(14,14) Mds[14][14] = M[238] Nds[14][14] = N[238]

Thread(14,15) Mds[14][15] = M[239] Nds[14][15] = N[239]

Thread(15, 0) Mds[15][ 0] = M[240] Nds[15][ 0] = N[240]

Thread(15, 1) Mds[15][ 1] = M[241] Nds[15][ 1] = N[241]

Thread(15, 2) Mds[15][ 2] = M[242] Nds[15][ 2] = N[242]

Thread(15, 3) Mds[15][ 3] = M[243] Nds[15][ 3] = N[243]

Thread(15, 4) Mds[15][ 4] = M[244] Nds[15][ 4] = N[244]

Thread(15, 5) Mds[15][ 5] = M[245] Nds[15][ 5] = N[245]

Thread(15, 6) Mds[15][ 6] = M[246] Nds[15][ 6] = N[246]

Thread(15, 7) Mds[15][ 7] = M[247] Nds[15][ 7] = N[247]

Thread(15, 8) Mds[15][ 8] = M[248] Nds[15][ 8] = N[248]

Thread(15, 9) Mds[15][ 9] = M[249] Nds[15][ 9] = N[249]

Thread(15,10) Mds[15][10] = M[250] Nds[15][10] = N[250]

Thread(15,11) Mds[15][11] = M[251] Nds[15][11] = N[251]

Thread(15,12) Mds[15][12] = M[252] Nds[15][12] = N[252]

Thread(15,13) Mds[15][13] = M[253] Nds[15][13] = N[253]

Thread(15,14) Mds[15][14] = M[254] Nds[15][14] = N[254]

Thread(15,15) Mds[15][15] = M[255] Nds[15][15] = N[255]

**Exercise 5.12**

In an attempt to improve performance, a bright young engineer changed the reduction kernel into the following. (A) Do you believe that the performance will improve? Why or why not? (B) Should the engineer receive a reward or a lecture? Why?

\_\_shared\_\_ float partialSum[];

unsigned int tid=threadIdx.x;

for (unsigned int stride=n>>1; stride >= 32; stride >>= 1) {

\_\_syncthreads();

if (tid < stride)

shared[tid] += shared[tid + stride];

}

\_\_syncthreads();

if (tid < 32) {

// unroll last 5 predicated steps

shared[tid] += shared[tid + 16];

shared[tid] += shared[tid + 8];

shared[tid] += shared[tid + 4];

shared[tid] += shared[tid + 2];

shared[tid] += shared[tid + 1];

}

**Solution:**

The kernel should have used partialSum[] instead of shared[]. On overlooking this issue, there are some additional issues with the kernel which have been given below.

The kernel does not perform sum reduction properly. For input vectors of size less than or equal to 32, the control skips the for loop and enters the last if statement. Within the if statement the sum of values are not assinged to any particular index position. The following example is the value of different index positions of partialSum[] for an input vector of length 8. As we can see none of them has the sum of the entire input vector.

Enter the number of elements to be summed up: 8

Thread: 0 partialSum[0]: 7.000000

Thread: 1 partialSum[1]: 11.000000

Thread: 2 partialSum[2]: 15.000000

Thread: 3 partialSum[3]: 19.000000

Thread: 4 partialSum[4]: 15.000000

Thread: 5 partialSum[5]: 18.000000

Thread: 6 partialSum[6]: 13.000000

Thread: 7 partialSum[7]: 7.000000

For an input vector of size greater than 32, the control enters the for loop and collects the sum of values for indices less than 32. However, we face an issue similar to small input vector sizes as there is no logic in the kernel to sum up the values and assign it to a particular index. The following example shows the value of different index positions of partialSum[] within the for loop for an input vector for length 64.

Enter the number of elements to be summed up: 64

Thread: 0 partialSum[0]: 32.000000 stride: 32

Thread: 1 partialSum[1]: 34.000000 stride: 32

Thread: 2 partialSum[2]: 36.000000 stride: 32

Thread: 3 partialSum[3]: 38.000000 stride: 32

Thread: 4 partialSum[4]: 40.000000 stride: 32

Thread: 5 partialSum[5]: 42.000000 stride: 32

Thread: 6 partialSum[6]: 44.000000 stride: 32

Thread: 7 partialSum[7]: 46.000000 stride: 32

Thread: 8 partialSum[8]: 48.000000 stride: 32

Thread: 9 partialSum[9]: 50.000000 stride: 32

Thread: 10 partialSum[10]: 52.000000 stride: 32

Thread: 11 partialSum[11]: 54.000000 stride: 32

Thread: 12 partialSum[12]: 56.000000 stride: 32

Thread: 13 partialSum[13]: 58.000000 stride: 32

Thread: 14 partialSum[14]: 60.000000 stride: 32

Thread: 15 partialSum[15]: 62.000000 stride: 32

Thread: 16 partialSum[16]: 64.000000 stride: 32

Thread: 17 partialSum[17]: 66.000000 stride: 32

Thread: 18 partialSum[18]: 68.000000 stride: 32

Thread: 19 partialSum[19]: 70.000000 stride: 32

Thread: 20 partialSum[20]: 72.000000 stride: 32

Thread: 21 partialSum[21]: 74.000000 stride: 32

Thread: 22 partialSum[22]: 76.000000 stride: 32

Thread: 23 partialSum[23]: 78.000000 stride: 32

Thread: 24 partialSum[24]: 80.000000 stride: 32

Thread: 25 partialSum[25]: 82.000000 stride: 32

Thread: 26 partialSum[26]: 84.000000 stride: 32

Thread: 27 partialSum[27]: 86.000000 stride: 32

Thread: 28 partialSum[28]: 88.000000 stride: 32

Thread: 29 partialSum[29]: 90.000000 stride: 32

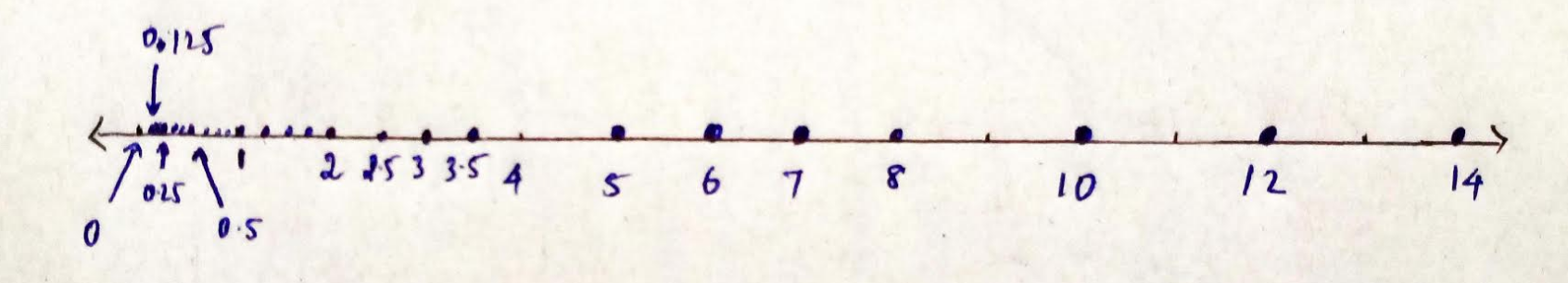
Thread: 30 partialSum[30]: 92.000000 stride: 32

Thread: 31 partialSum[31]: 94.000000 stride: 32

**Exercise 6.2**

Draw the equivalent of Fig. 6.5 for another 6-bit format (1-bit sign, 2-bit mantissa, 3-bit exponent). Use your result to explain what each additional exponent bit does to the set of representable numbers on the number line.

**Solution:**



The additional exponent bit doubles the number of major intervals of representable numbers compared to Fig. 6.5. The intervals to the left of zero are not represented in the daigram.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exponent** | **Mantissa** | **E10** | **e** | **Value** |
| 000 | 00 | 0 | -3 | 0.125 |
|  | 01 |  |  | 0.15625 |
|  | 10 |  |  | 0.1875 |
|  | 11 |  |  | 0.21875 |
| 001 | 00 | 1 | -2 | 0.25 |
|  | 01 |  |  | 0.3125 |
|  | 10 |  |  | 0.375 |
|  | 11 |  |  | 0.4375 |
| 010 | 00 | 2 | -1 | 0.5 |
|  | 01 |  |  | 0.625 |
|  | 10 |  |  | 0.75 |
|  | 11 |  |  | 0.875 |
| 011 | 00 | 3 | 0 | 1 |
|  | 01 |  |  | 1.25 |
|  | 10 |  |  | 1.5 |
|  | 11 |  |  | 1.75 |
| 100 | 00 | 4 | 1 | 2 |
|  | 01 |  |  | 2.5 |
|  | 10 |  |  | 3 |
|  | 11 |  |  | 3.5 |
| 101 | 00 | 5 | 2 | 4 |
|  | 01 |  |  | 5 |
|  | 10 |  |  | 6 |
|  | 11 |  |  | 7 |
| 110 | 00 | 6 | 3 | 8 |
|  | 01 |  |  | 10 |
|  | 10 |  |  | 12 |
|  | 11 |  |  | 14 |

**Exercise 6.3**

Assume that in a new processor design, due to technical difficulty, the floating-point arithmetic unit that performs addition can only do “round to zero” (rounding by truncating the value toward 0). The hardware maintains sufficient number of bits that the only error introduced is due to rounding. What is the maximal ulp error value for add operations on this machine?

**Solution:**  0D ULP

It does not introduce any error

**Exercise 6.4**

A graduate student wrote a CUDA kernel to reduce a large floating-point array to the sum of all its elements. The array will always be sorted with the smallest values to the largest values. To avoid branch divergence, he decided to implement the algorithm of Fig. 5.16. Explain why this can reduce the accuracy of his results.

**Solution:**

The algorithm adds each element with another element that is blockDim.x/2 units apart. As the elements are in ascending order such an algorithm would result in adding a value with small exponent with a value with large exponent. This entails the mantissa of the element with small exponent being right shifted till the exponents of both the elements are equal. Such an approach leads to truncation of bits of the sum of elements thus reducing the accuracy.