Dueling GPUs

Peter Brooker

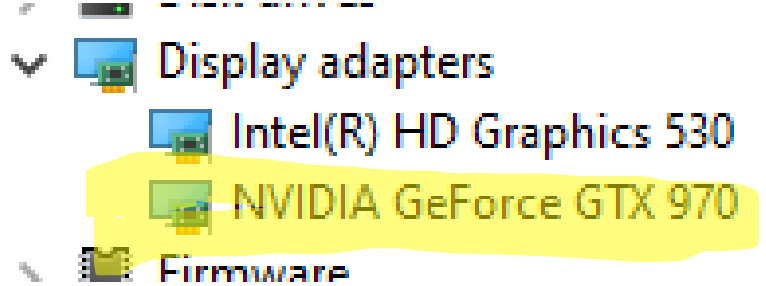
March 2024

Outline

- Dueling GPU code description
- Sample Game Output
- Kernel Used by Code
- Implementation Details
- Potential Code improvements

Dueling GPU description

Dueling GPUs description



- Assignment was for Dueling GPUs
- Unfortunately, Alienware Minitower has only 1 CUDA GPU
- Code was created to “Virtually” access 2nd GPU
  - Memory Allocation explicitly for d0_… and d1_…
  - If 2nd GPU present, code uses cudaSetDevice(1)

```
if (nDevices > 1)
{
    cudaSetDevice(1);
}
```

- Uses concept of GPU weighted board vector multiplied by each 1 of 10 solution vectors
- Dot product SUM used to identify 4 in a row as well as next guess
- If GPU#1 is first, “X” placed in Top Left
- Else, GPU#2 “O” placed randomly

Sample Game Output

Microsoft Visual Studio Debug Console

```
****************************************
* 2 GPUs compete in 4x4 Tic-Tack-Toe *
****************************************


Number of CUDA capable devices = 1

2nd GPU starts

 - | - | - | O
---|---|---|---
 - | - | - | -
---|---|---|---  End of First Turn
 - | - | - | -
---|---|---|---
 - | - | - | -



Now 1st GPU
h_boardOfNums:                0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0
Vector for SV multiply :      1     1     1  -100     1     1     1     1     1     1     1     1     1     1     1     1


 - | - | - | O
---|---|---|---
 - | - | - | X
---|---|---|---  End of Turn
 - | - | - | -
---|---|---|---
 - | - | - | -
```

```
Now 2nd GPU.
h_boardOfNums:                0     0     0     1     0     0     0    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :      1     1     1    10     1     1     1  -100     1     1     1     1     1     1     1     1


 -  |  -  |  O  |  O
----|-----|-----|----
 -  |  -  |  -  |  X
----|-----|-----|----   End of Turn
 -  |  -  |  -  |  -
----|-----|-----|----
 -  |  -  |  -  |  -


Now 1st GPU
h_boardOfNums:                0     0     1     1     0     0     0    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :      1     1  -100  -100     1     1     1    10     1     1     1     1     1     1     1     1


 -  |  -  |  O  |  O
----|-----|-----|----
 -  |  -  |  X  |  X
----|-----|-----|----   End of Turn
 -  |  -  |  -  |  -
----|-----|-----|----
 -  |  -  |  -  |  -
```

```
Now 2nd GPU.
h_boardOfNums:                0     0     1     1     0     0    -1    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :      1     1    10    10     1     1  -100  -100     1     1     1     1     1     1     1     1


 -  | O  | O  | O
----|----|----|----
 -  |  - |  X |  X
----|----|----|----  End of Turn
 -  |  - |  - |  -
----|----|----|----
 -  |  - |  - |  -


Now 1st GPU
h_boardOfNums:                0     1     1     1     0     0    -1    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :      1  -100  -100  -100     1     1    10    10     1     1     1     1     1     1     1     1


 -  | O  | O  | O
----|----|----|----
 -  |  X |  X |  X
----|----|----|----  End of Turn
 -  |  - |  - |  -
----|----|----|----
 -  |  - |  - |  -
```

```
Now 2nd GPU.
h_boardOfNums:                0     1     1     1     0    -1    -1    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :      1    10    10    10     1  -100  -100  -100     1     1     1     1     1     1     1     1


 O | O | O | O
---|---|---|---
 - | X | X | X
---|---|---|---  Four in a row for 2nd GPU
 - | - | - | -
---|---|---|---
 - | - | - | -



4 in a row for 2nd GPU
**********end of game**********
D:\Work\CUDA\CudaGPUBattle\x64\Debug\CudaGPUBattle.exe (process 4768) exited with code 0.
Press any key to close this window . . .
```

Kernel Used by Code:

Cuda Device Kernel for SV array multiplication

```
// same kernel for whether it is on GPU#1 or GPU#2
// the kernel call will pass in different array pointers
// also, the cudaMemcpy() will use different source arrays
__global__ void SVKernel(int* d_forDotProduct,  int* d_SV01, int* d_SV02, int* d_SV03, int* d_SV04, int* d_SV05,
                                                int* d_SV06, int* d_SV07, int* d_SV08, int* d_SV09, int* d_SV10,
                                                int* d_dot01, int* d_dot02, int* d_dot03, int* d_dot04, int* d_dot05,
                                                int* d_dot06, int* d_dot07, int* d_dot08, int* d_dot09, int* d_dot10 )
{
    int i = threadIdx.x;
    d_dot01[i] = d_forDotProduct[i] * d_SV01[i];
    d_dot02[i] = d_forDotProduct[i] * d_SV02[i];
    d_dot03[i] = d_forDotProduct[i] * d_SV03[i];
    d_dot04[i] = d_forDotProduct[i] * d_SV04[i];
    d_dot05[i] = d_forDotProduct[i] * d_SV05[i];
    d_dot06[i] = d_forDotProduct[i] * d_SV06[i];
    d_dot07[i] = d_forDotProduct[i] * d_SV07[i];
    d_dot08[i] = d_forDotProduct[i] * d_SV08[i];
    d_dot09[i] = d_forDotProduct[i] * d_SV09[i];
    d_dot10[i] = d_forDotProduct[i] * d_SV10[i];
}
```

- Just performs multiplication with each one of 10 solution vectors
- Sum of multiplication is the vector “dot” product.

Implementation Details:

Solution Vectors (SV) defined in code

- Represents board as single 16 element, scalar array.
- A 4 x 4 board has 10 possible solutions
  - 4 horizontal rows
  - 4 vertical columns
  - 2 diagonals
- Each solution can be represented as 16x1 array of 0s & 1s
  - Top row: {1,1,1,1, 0,0,0,0, 0,0,0,0, 0,0,0,0}
  - 1st column: {1,0,0,0, 1,0,0,0, 1,0,0,0, 1,0,0,0}
  - Diagonal: {1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1}

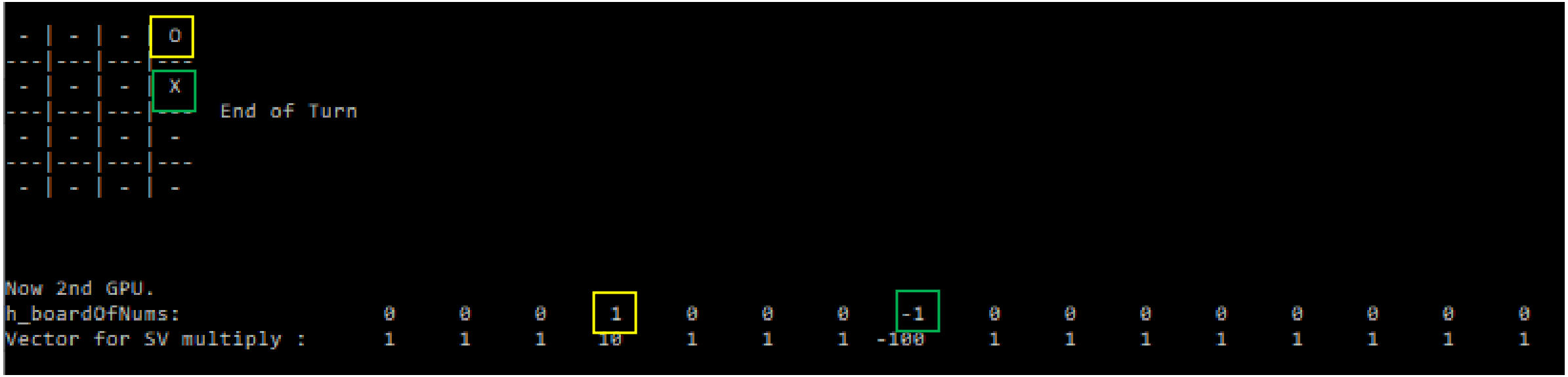
10 Solution Vectors (SV) defined in code

- Will multiply SVs by board weighted for each GPU
- Dot product sum will be used for next guess
- Sum also tests if board is at solution

```
// set up solution vectors for 10 possible solution. hz row, vt col, diagonal.
int h_SV01[16] = {  1,1,1,1, 0,0,0,0, 0,0,0,0, 0,0,0,0 }; // top row
int h_SV02[16] = {  0,0,0,0, 1,1,1,1, 0,0,0,0, 0,0,0,0 }; // 2nd row
int h_SV03[16] = {  0,0,0,0, 0,0,0,0, 1,1,1,1, 0,0,0,0 }; // 3rd row
int h_SV04[16] = {  0,0,0,0, 0,0,0,0, 0,0,0,0, 1,1,1,1 }; // bot row
int h_SV05[16] = {  1,0,0,0, 1,0,0,0, 1,0,0,0, 1,0,0,0 }; // 1st col
int h_SV06[16] = {  0,1,0,0, 0,1,0,0, 0,1,0,0, 0,1,0,0 }; // 2nd col
int h_SV07[16] = {  0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0 }; // 3rd col
int h_SV08[16] = {  0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,1 }; // 4th col
int h_SV09[16] = {  1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 }; // NW-SE diag
int h_SV10[16] = {  0,0,0,1, 0,0,1,0, 0,1,0,0, 1,0,0,0 }; // SW-NE diag
```

..h_boardOfNums array

- GPU#1 “X”: value = -1 in array
- GPU#2 “O”: value = 1 in array
- Open Tile: value = 0 in array



GPU weighted board vector

```
 -  |  -  |  -  |  O
----|-----|-----|----
 -  |  -  |  -  |  X
----|-----|-----|----   End of Turn
 -  |  -  |  -  |  -
----|-----|-----|----
 -  |  -  |  -  |  -



Now 2nd GPU.
h_boardOfNums:              0     0     0     1     0     0     0    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :    1     1     1    10     1     1     1  -100     1     1     1     1     1     1     1     1
```

- Above is for GPU#2 (“O”) which has value 1 in h_boardOfNums
- GPU#2 Weighted vector =10 if tile is “O”, -100 if “X”, 1 if open

Code for GPU#2 weighted vector

```
cout << "Vector for SV multiply : ";
int h_forDotProduct[16];
for (int j = 0; j < 16; ++j)
{
    if (h_boardOfNums[j] == -1)   //this is GPU0...1st GPU
    {
        h_forDotProduct[j] = -100;
    }
    else if (h_boardOfNums[j] == 1)  // this is GPU1...2nd GPU
    {
        h_forDotProduct[j] = 10;
    }
    else if (h_boardOfNums[j] == 0)
    {
        h_forDotProduct[j] = 1;
    }
```

```
Now 2nd GPU.
h_boardOfNums:              0     0     0     1     0     0     0    -1     0     0     0     0     0     0     0     0
Vector for SV multiply :    1     1     1    10     1     1     1  -100     1     1     1     1     1     1     1     1
```

Next Guess Identification

- On host side, create sums of all d_dotXX arrays returned from Kernel
  - XX = 01,02,…,10
  - Sum is the “dot product” of GPU weighted board array with each of the solution vectors
- Identify the max sum
- If max sum = 10 + 10 + 10 +1 =31, next guess is in the open space and game is over

Potential Code Improvements

Potential Code Improvements

- Guessing
- Proposed Improvement:
  - Place all code in functions after main
  - In main, only highlevel functions are called
  - Function names would be very descriptive so main would read like a flow chart
  - Why not done? Returning arrays defined in functions confusing due to heap vs stack persistence issues.

Thank You!