

# Dataparallel Programming on GPUs

Daniel Rembold, Maximillian Scholz

Technische Universität Hamburg Harburg

*daniel.rembold@tuhh.de , maximillian.scholz@tuhh.de*

July 6, 2014

# Inhaltsverzeichnis

1. Aufgabenstellung
2. Einfache Implementierung
3. NVIDIA-Template
4. Optimierung
5. Laufzeit- und Performancemessung
6. Fazit

# Aufgabenstellung

- ▶ Möglichst effiziente Implementierung einer Matrixmultiplikation in row-major und column-major Format
- ▶ Quadratische Matrizen der Dimension  $32 * k, k \in \{1, 2, 4, \dots, 128\}$

# Naiver Algorithmus

- ▶ C-Code für die CPU
- ▶ Sequentielle Ausführung

## Listing 1: Matrixmultiplication in C

```
int i,j,k;

for( i = 0, i<Ndim; i++){
    for( j = 0, j<Mdim; j++){
        for( k = 0; k<Pdim; k++){
            C[i*Ndim+j] += A[i*Ndim+k] * B[k*Pdim+j];
        }
    }
}
```

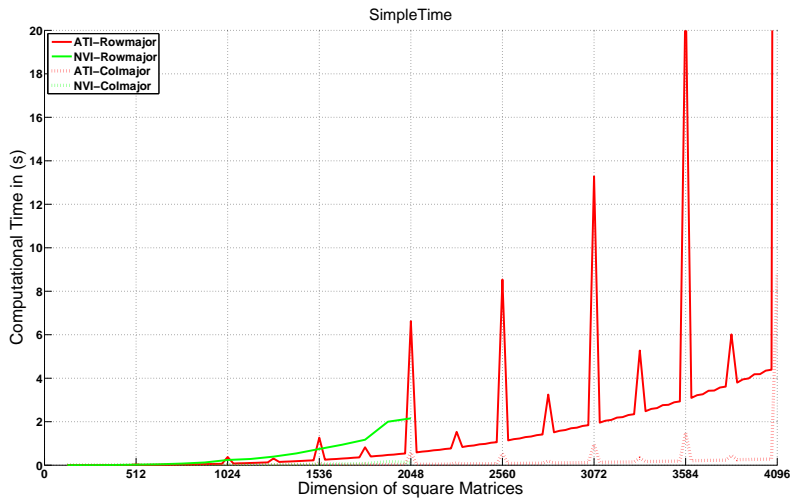
# Einfache Implementierung

## Einfache Implementierung in OpenCL

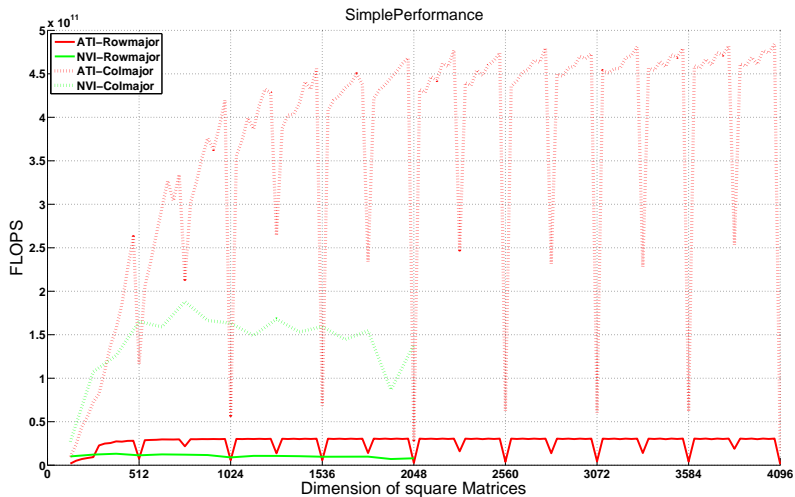
### Listing 2: Einfachster Code in OpenCL

```
int i,j,k;
i = global_id(0);
j = global_id(1);
for(k=0; k<Pdim; k++){
    C[i*Ndim+j] += A[i*Ndim+k] * B[k*Pdim+j];
}
```

# Laufzeit



# Performance



# NVIDIA-Template

- ▶ Zerlegung der Matrix in Blöcken
- ▶ Verwendung von lokalem Speicher

## Listing 3: NVIDIA-Snippet

```
int bx = get_group_id(0); // Blockindex x
int by = get_group_id(1); // Blockindex y
int tx = get_local_id(0); // Threadindex x
int ty = get_local_id(1); // Threadindex y
```



## Listing 4: NVIDIA-Snippet(2)

```
__local T As[16][16]; __local T Bs[16][16]; T Csub =
    0.0f;
for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep
    , b += bStep){
    As[ty][tx] = A[a + ty + hA * tx];
    Bs[ty][tx] = B[b + ty + hA * tx];
    barrier(CLK_LOCAL_MEM_FENCE);
    for (int k = 0; k < 16; ++k) Csub += As[ty][k] * Bs[k][
        tx];
    }
    barrier(CLK_LOCAL_MEM_FENCE);
}
int c = 16 * ( by + hA * bx):
C[c + hA * tx + ty] = Csub;
```

## Optimierung

- ▶ Anzahl der Threadblocks werden verringert, Anzahl der Blocks bleiben
- ▶ Halb so viele Threads, jedoch doppelt so viel Arbeit pro Thread

### Listing 5: Optimized Code in OpenCL

```
T Csub[2] = {0,0};
__local T As[BLOCK_SIZE][BLOCK_SIZE];
__local T Bs[BLOCK_SIZE][BLOCK_SIZE];
for (int a = aBegin, b = bBegin; a <= aEnd; a += aStep,
     b+=bStep){
AS(ty, tx) = A[a + wA * ty + tx];
BS(ty, tx) = B[b + wB * ty + tx];
AS(ty+16, tx) = A[a + wA * (ty+16) + tx];
BS(ty+16, tx) = B[b + wB * (ty+16) + tx];
barrier(CLK_LOCAL_MEM_FENCE); // sync threads
}
```

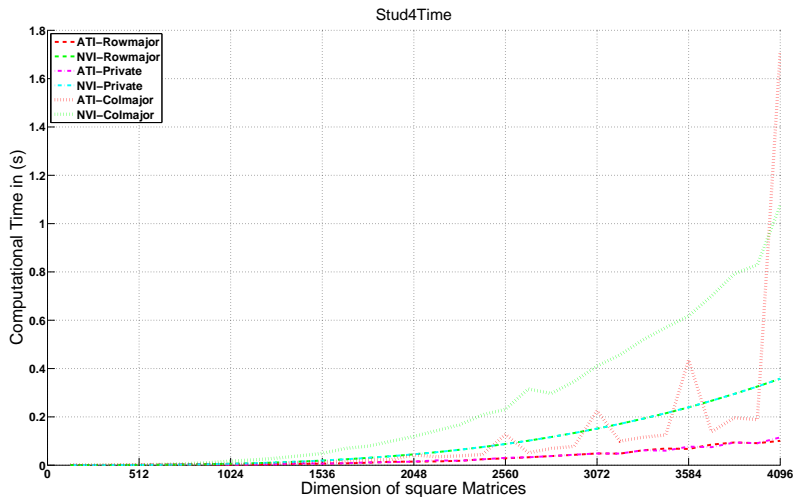
# Optimierung

## Listing 6: innere Schleife und Output

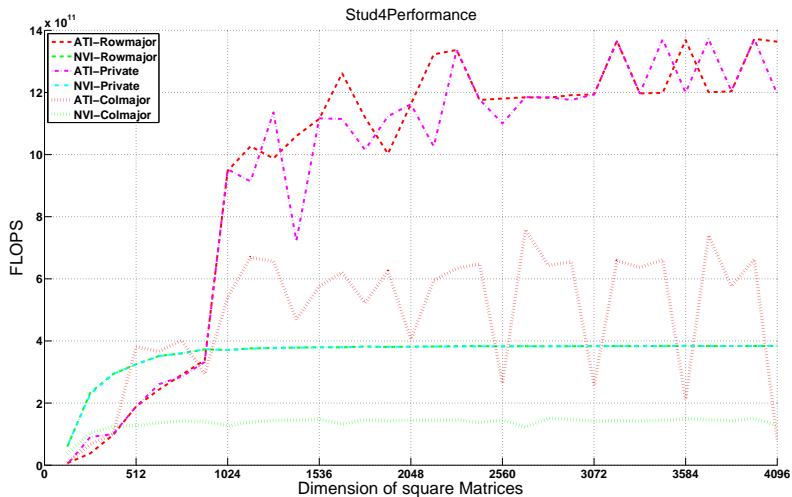
```
for (int k = 0; k < BLOCK_SIZE; ++k){  
  Csub[0] += AS(ty, k) * BS(k, tx);  
  Csub[1] += AS(ty+16, k) * BS(k, tx);  
}  
barrier(CLK_LOCAL_MEM_FENCE); // sync threads  
}  
int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;  
C[c + wB * ty + tx] = Csub[0];  
C[c + wB * (ty+16) + tx] = Csub[1];
```

- ▶ Daten aus dem lokalen Speicher werden wiederverwendet
- ▶ Weniger Threads pro Block ermöglichen Überlappung von Speicherzugriff mit Arithmetik

# Laufzeitmessung auf ATI und NVIDIA



# Performancemessung auf ATI und NVIDIA



# Begründung der Ergebnisse und mögl. Verbesserungen

- ▶ Vektorarithmetik (float4)
- ▶ Image-Objekte

# Quellen



University of Bristol

Optimizing OpenCL performance

[http://www.cs.bris.ac.uk/home/simonm/workshops/OpenCL\\_lecture3.pdf](http://www.cs.bris.ac.uk/home/simonm/workshops/OpenCL_lecture3.pdf)



NVIDIA

OpenCL SDK Code Samples

<https://developer.nvidia.com/opencl>



Vasily Volkov (UC Berkeley , September 22, 2010)

Better Performance at Lower Occupancy

<http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf>

Vielen Dank für eure Aufmerksamkeit!