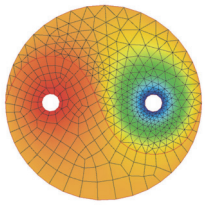
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 16, 2016, No. 3



DATA MANAGEMENT IN CUDA PROGRAMMING FOR HIGH BANDWIDTH MEMORY IN GPU ACCELERATORS

GRZEGORZ KORPAŁA

Institute of Metal Forming, Technische Universität Bergakademie Freiberg
Bernhard v. Cotta Str. 4, 09599 Freiberg, Germany
Corresponding author: Grzegorz.Korpala@imf.tu-freiberg.de

Abstract

The new High Bandwidth Memory 2 (HBM 2) built into the Tesla P100 enables speedier calculations without much effort. HBM 2 by P100 has a max. bandwidth of 720 GB/s which is lower than the bandwidth of the GPU cache and Shared Memory (SMem) of Kepler-GPU which are almost 2,500 GB/s in size (Woolley, 2013). For Kepler-GPU architecture it is popular to shift data to the SMem and decrease the computation time by reduction of access number to VRAM. In new GPUs like Maxwell and Pascal with much higher bandwidth it is questionable if use of SMem in this architecture gives large increase of performance. This publication will explain how data management between Video-RAM (VRAM) and the GPU processor must look like in order to be able to utilize the full calculation power of the GPU (depending of GPU architecture) by simple models for a three-dimensional calculation.

Key words: GPU computation shared memory, memory management

1. INTRODUCTION

The technological development of new materials nowadays begins with the understanding of structural processes and their effect on technology design. In the relevant literature, structure models can be found for hot forming which provide a very good image of reality based on physical phenomena. Among these calculation methods for material models are the phase field models or the cellular automaton (CA) concept. Constant improvements of the models lead to continuously improving precision for universal applications which unfortunately impacts the calculation times negatively. The exact adjustment of coefficients requires more and more time. In addition, a combination of FEM systems and these material models puts a serious strain on the computing power and necessitates the use of HPC systems. A concept for the use of the CPU for FEM calculations and the GPU for structure models would significantly reduce the calculation times. Many publications have introduced structure models which can be well implemented in the GPU language and this solution was even suggested explicitly. Unfortunately, this concept was not pursued any further within the applicable literature or solutions are stuck in the development phase. Programming such calculation systems is based on programming languages such as Nvidia-CUDA® or OpenCl which not always have the same program structure as CPU-based solutions. There are frequent mistakes made during the direct implementation of the CPU codes to GPU codes of other models or use of function libraries which will actually increase calculation times. The main problems are usually caused by misunderstandings in hardware/software set-up. These problems always appear when material scientists who have no experience in such programming tasks seek to design a simple program. A constant shift of large amounts of data between RAM-VRAM and VRAM-GPU is not desirable and results in a lower calculation per-

 ISSN 1641-8581

formance of the GPU. Even though the data management on the level of RAM-VRAM is well-known and can be automated using the ToolKits, VRAM-GPU data management is little known. Strategies for CPU-GPU data management can already be found in the literature (Sanders & Kandrot, 2010; Korpala & Kawalla, 2015; Woolley, 2013). The hardware specification can be found in (Korpala & Kawalla 2015).

2. CALCULATION ALGORITHM

The HBM 2 integrated in the new GPU accelerators is equipped with a wide memory interface which is 4096 Bit. It allows massive access to the data saved in HBM 2. For details about Tesla P100 and HBM 2 see in white paper of P100 (Nvidia, 2016). The data can be called up using several threads (VRAM processing - VRAM in figure 1). However, until the new Pascal architecture, this type of memory was not in use. An acceleration of the calculation speed was for Kepler and Maxwell-architecture possible, through the preceding import of the data in the SMem and computation (VRAM-S. Memory-Processing-VRAM in figure 1).

grids. In addition, the some CA models require 6, 26 or even 124 neighbour cells for the central cell for the calculation, depending on the method of calculation. In figure 3, there are three examples of the 26 positions of the calculation block in the calculation volume.

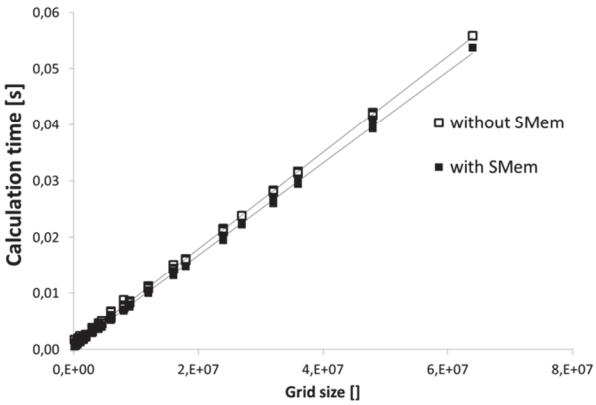


Fig. 2. *Calculation time for two different data management strategies (without and with SMem)*

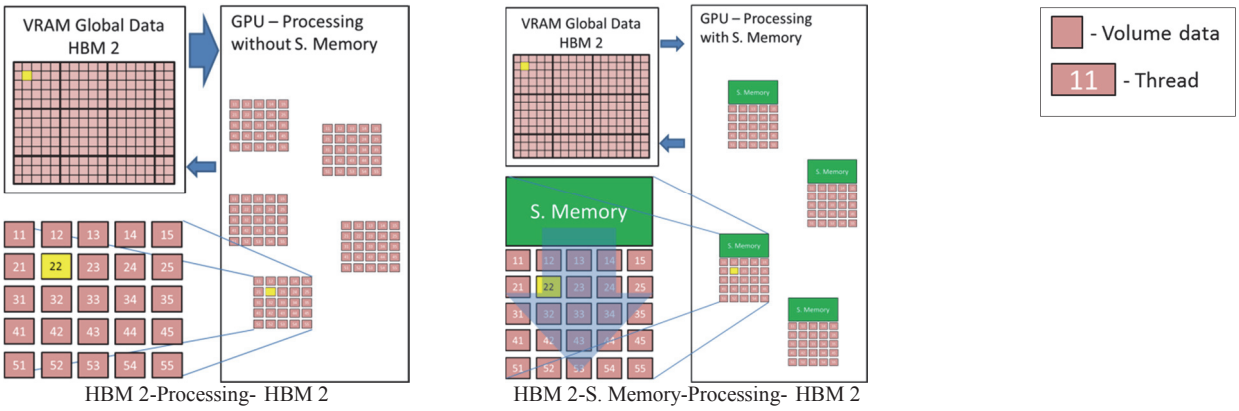
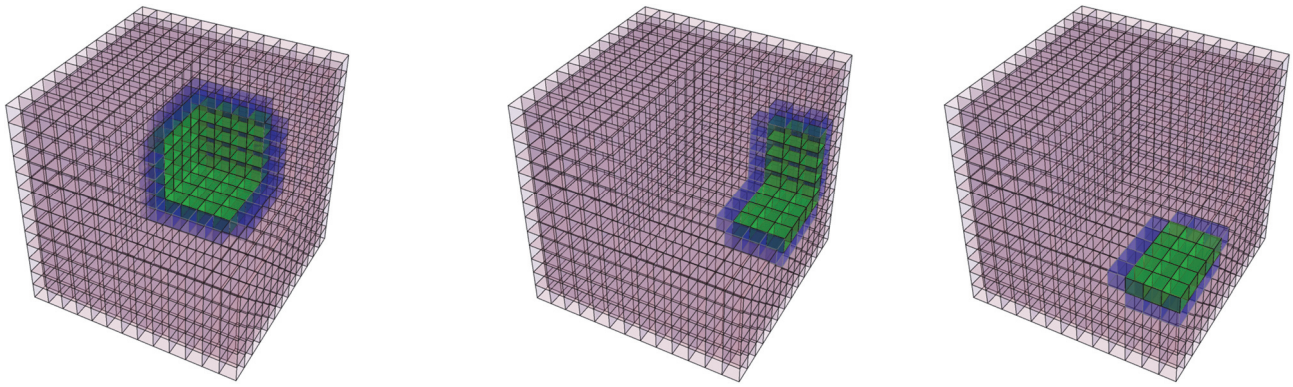


Fig. 1. *Diagram of the data computation in the GPU-accelerators using different memories*

The potential acceleration for a simple calculation of sum of the 6 neighbours (left, right, in front, behind, above and below) can be found in figure 2 and results in a degrease of calculation time of up to 12% by large computation grids on Kepler GPU.

The data import into SMem may happen in different ways. The simplest method is to have only the first thread copy the significant values in the calculation block. In the second method, all threads copy the values significant for the corresponding thread ID. Of course the import becomes complicated for 3D grids since the thread block sizes as well as the SMem size must remain constant while the grids for the structure simulation can vary. That means that there is no data to be imported on the edges of the

The size of the table used for the calculation in SMem is as in figure 3, longer, deeper and wider by 2, which enables simple access to the corresponding features. It should be noted that the maximum number of such tables with double-precision values (64 bit) depends on the capacity of the GPU architecture. All other data are called up through direct access to VRAM. That means that such code design includes planning beforehand. To save in SMem, the parameters used mostly in the calculation should be selected.



In the centre with all neighbours | **At the edge with the missing neighbours on one side** | **At the edge with the missing neighbours on two sides**

Fig. 3. *Calculation volume and block position in the calculation, purple - calculation volume, green - cells, which are loaded directly through the threads, blue - neighbour cells which are loaded through the central thread.*

Indexing threads in the calculation block is realised as is customary with three parameters. (threadIdx.x, threadIdx.y, threadIdx.z). However, the indexing according to nomenclature of CUDA is only written with one parameter which leads to problems with many programmers. In addition, the SMem tables are shifted by 1 in each direction. Indexing for 3D looks as follows (Wolfram, 2015):

```
threadIdx.*,blockDim.*,blockIdx.*;          Integrated values in CUDA
width, height, depth;                       width, height and depth of the calculation volume

thx = threadIdx.x + 1;                      Corresponding position in SMem
thy = threadIdx.y + 1;
thz = threadIdx.z + 1;
x = blockIdx.x * blockDim.x +               Corresponding x, y positions in the calculation volume
threadIdx.x;
y = blockIdx.y * blockDim.y +
threadIdx.y;

zi = threadIdx.z;                           Auxiliary values for the calculation in the depth of the calculation volume
str = blockDim.z;
mpos = x + (y + zi * height) * width;
lim = depth - zi;
```

In addition, the calculation of the entire calculation volume is always realised using 3D blocks which result in a 2D grid. These 2D block grids are displayed in figure 4 in layers with the same brightness.

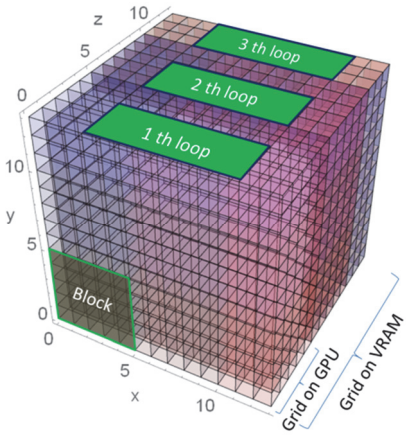


Fig. 4. *3D blocks stacked in 2D grid (loops calculate the entire calculation volume).*

The calculation volume in the depth is calculated by a loop with an addition "for" function:

```
for (ii = 0; ii <= lim; ii += str)          Loop for the calculation in the depth
{

index = ii * width * height +               Index for the VRAM
mpos;                                       z position in the calculation volume
z = index / width / height;

  /*Loading of SMem*/                       Importing in SMem

 __syncthreads();

  /*Calculation*/                           Actual calculation

}
```

3. ALGORITHM FOR THE IMPORTING DATA IN SMEM

GPUs allows for parallel and massive access of multiple threads to the data available in the VRAM, the concept is based on the function "If" within which that particular thread calls up the data from the VRAM. First, all data of the cells is loaded which have been assigned threads. The first "If"

condition restricts importing since the data are in a one-dimensional indexing of the calculation volume. The conditions also have to be entered for which the threads located on the block edge import the data adjacent to that block. For more complex calculation instructions with multiple neighbours, additional information is required which prolongs the import algorithm.

```
if (x < width && y < height && \                Conditions for
      z < depth && thx <= blx && \              correct data im-
            thy <= bly && thz <= blz)           port
{
  SMem[thx][thy][thz]=VRAM[index];

                                                Data import for
    if("Condition")                             green area
    {
 SMem[xSM] [ySM] [zSM] =
VRAM[iVRAM];
    }                                           Data import for
                                                blue area
}

                                                (s. table 1)
```

Importing the corresponding data "VRAM[iVRAM]" to certain addresses on the edges of the SMem "SMem[xSM] [ySM] [zSM]" must occur according to table 1.

As can be seen in figure 5, the minimum of calculation time for Kepler GPU is achieved with the highest volume of access to the neighbouring data. The performance does not reach the maximum level for small blocks. Depending on where most of the data is saved in the hardware, the acceleration will vary. The benefit of using of SMem in Kepler begins by 6 time access and size of blocks of more than 4 cells.

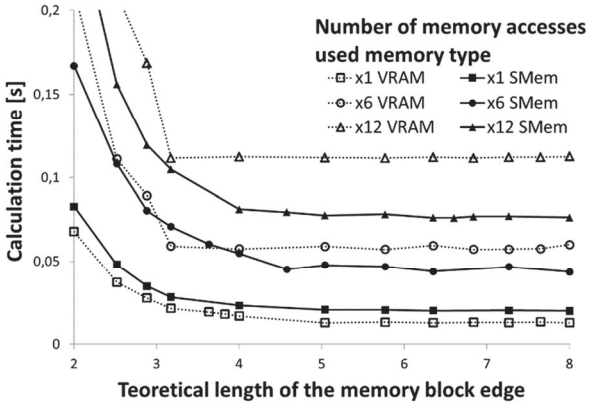


Fig. 5. *Calculation time for 64,000,000 elements in 3D (Kepler GPU)*

Table 1. *Conditions for "If" condition of data import in SMem*

Condition	iVRAM	xSM	ySM	zSM
thx == blx && xpos < width - 1	index + 1	thx + 1	thy	thz
thy == bly && ypos < height - 1	index + width	thx	thy + 1	thz
thz == blz && zpos < depth - 1	Index + (width * height)	thx	thy	thz + 1
thx == 1 && xpos > 0	index - 1	thx - 1	thy	thz
thy == 1 && ypos > 0	index - width	thx	thy - 1	thz
thz == 1 && zpos > 0	index - (width * height)	thx	thy	thz - 1

Data which are is SMem can then simply be used for any calculation.

4. RESULTS

The performance test was conducted on volume with size of 400x400x400 cells for different GPUs architectures, block-memory size and with using only VRAM or with using import data to SMem. The theoretical block size can be found from the third root of the number of processes in the block. Various lengths, widths and depths of the block are considered and the fastest variant is taken for a theoretical length. Each calculation time is a mean time from 100 samples. In algorithm we make a sum of 6 neighbours. This calculation we make 1, 6 or 12 to check when we achieve the profit from the use of SMem.

The Maxwell and Pascal architecture has higher memory bandwidth then Kepler and the using of SMem gives no satisfying results. By some combination of SMem size the calculation takes more time with SMem then without. The test performance shows that SMem by Maxwell and Pascal gives no benefit by calculation time.

The use of small blocks by compilation increases the calculation time independently from the GPU-architecture. The impact of this tendency depends on the architecture. Of course this impact by the new GPUs is smaller than by the old one.

It was not clear why no effect on the calculation time could be found. The code was compiled once more and tested with the single precision (SP) data type. The results of calculation time with the sum of

6 neighbours and with six times access to each cell is illustrated in figure 8. Here the benefit of using SMem is large for all GPUs.

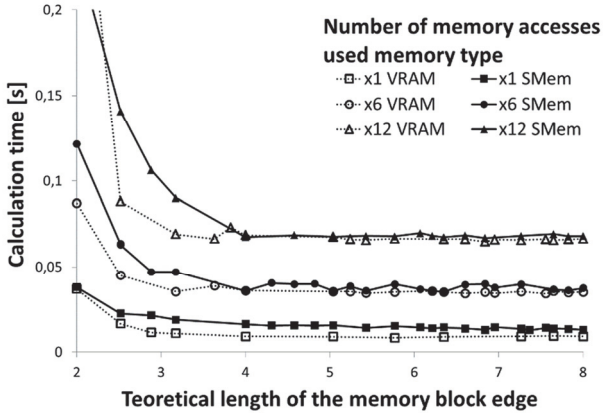


Fig. 6. Calculation time for 64,000,000 elements in 3D (Maxwell GPU)

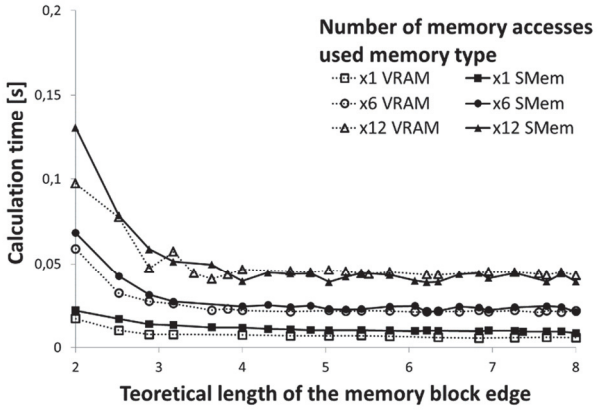


Fig. 7. Calculation time for 64,000,000 elements in 3D (Pascal GPU HBM 2)

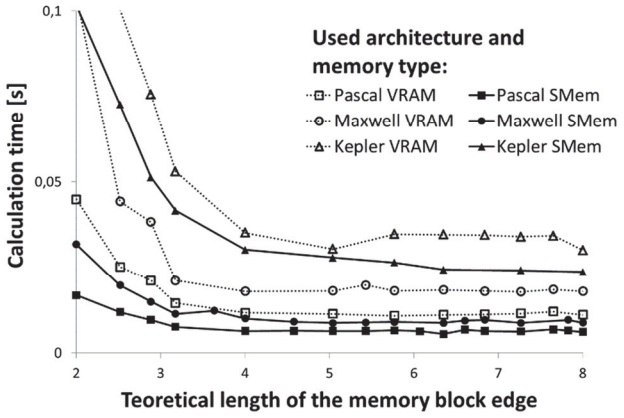


Fig. 8. Calculation time for 64,000,000 elements in 3D with single precision data type and different architectures

5. SUMMARY

The implementation of the GPU calculation can be realized using different strategies. The application of SMem by Kepler architecture is generally recommended. However, careful planning of data amounts is required for the programming of complex structure models. With the development of the new memory type like GDDRX and HBM 2, which has a larger bandwidth, in general shorter calculation times cannot be expected or the benefit of using SMem is too small according to programing effort. For material scientists who expect low computing times, the use for HBM 2-based accelerators without SMem is an attractive solution. As an alternative, users may also turn to the existing solutions of GPU manufacturers. However, the limited optimisation options of the manufacturers should be mentioned here.

Most commercial functions were prepared universally for a wide spectrum of GPU cards which can result in a reduction in performance (by Kepler-GPU). Further advancement of accelerators will continue to head in the direction of higher VRAM capacities as well as faster memory speeds. The reason for the lower improvement of calculation time was the additional effort by translation of 32 bit SMem to the 64 bit data which are used by calculation. Despite small SMem impact on calculations time by double precision instructions the calculation is much greater than using the CPU. That is why the application may be of significant support for certain fields within materials science. Processing large data from in-situ tests, crystallographic analyses and even inverse analyses will be carried out on desktop computers in the future (Ferenc et al., 2011).

ACKNOWLEDGEMENT

The author would like to thank Nvidia Corporation for Titan X (Maxwell) donation.

REFERENCES

Ferenc, M. J., Ferenc, I., Róbert, M., István, L., 2011, Simulation of reaction–diffusion processes in three dimensions using CUDA. Analytical Platforms for Providing and Handling Massive Chemical Data, 76-85.

Korpala, G., Kawalla, R., 2015, Optimization and application of GPU calculations in material science. *Computer Methods in Materials Science*, 15 (1), 185-191.

Nvidia. 2016, Whitepaper NVIDIA Tesla P100. Available online at: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecturewhitepaper. pdf, accessed: 15.11.2016

Sanders, J., Kandrot, E., 2010, An Introduction to General-Purpose GPU Programming. In J. Sanders, & E. Kandrot, CUDA by Example.

Wolfram, S., 2015, Wolfram Language Dokumentation Center Mathematica 10.3. Wolfram.

Woolley, C., 2013, GPU Optimization Fundamentals. Available online at: http:https://www.olcf.ornl.gov/wp-content/uploads/2013/02/ GPU_Opt_Fund-CW1.pdf. accessed: 15.11.2016.

ZARZĄDZANIE PAMIĘCIĄ W PROGRAMOWANIU CUDA DLA OSIĄGNIECIA WYSOKIEJ PRZEPUSTOWOŚCI PAMIĘCI W AKCELERATORACH GPU

Streszczenie

Nowa pamięć wysokiej przepustowości (HBM 2) wykorzystywana w karcie Tesla P100 umożliwia znaczne przyspieszenie obliczeń. Pamięć HBM 2 zastosowana w modelu P100 pozwala na transfer danych z przepustowością 720 GB/s, co jest ciągle mniejszą wartością niż prędkości oferowane przez pamięć podręczną i współdzieloną (SMem) procesorów GPU należących do architektury Kepler, których wartości osiągają poziom 2,500 GB/s (Woolley, 2013). Popularnym podejściem stosowanym w celu skrócenia czasu obliczeń w architekturze Kepler jest zastosowanie przesunięcia danych do pamięci SMem w celu zredukowania ilości dostępów do VRAM. W nowych procesorów graficznych takich jak Maxwell i Pascal oferujących znacznie wyższą przepustowość pamięci wątpliwości poddaje się sens wykorzystania SMem do osiągnięcia wzrostu wydajności. W publikacji wyjaśniono sposób zarządzania pamięcią Video-RAM (VRAM) i procesora w celu pełnego wykorzystania mocy obliczeniowej GPU (w zależności od architektury) na podstawie prostych modeli i twój-wymiarowych obliczeń.

Received: October 7, 2016
Received in a revised form: October 27, 2016
Accepted: November 9, 2016