Exploring the Benefits of OpenCL Shared Virtual Memory: A Comparative Analysis on Integrated and External GPUs

Filip Krużel and Mateusz Nytko
Department of Computer Science
Cracow University of Technology
Warszawska 24, 31-155 Kraków, Poland
Email: filip.kruzel@pk.edu.pl

KEYWORDS

GPGPU, Intel Xe-LP, Intel ARC, Shared Virtual Memory, OpenCL, Nvidia RTX.

ABSTRACT

In this article, we test the feature of using a Shared Virtual Memory in OpenCL on Intel Iris Xe and Nvidia GeForce RTX 3060 GPUs. The first of these architectures is integrated into the CPU, so by definition, it uses the same RAM as the CPU. The second one uses a PCI-Express connection to transfer data between computer memory and separate GPU memory. OpenCL Shared Virtual Memory (SVM) feature allows zero-copy mechanisms to use the same address space for the CPU and Accelerator. In this work, the authors test the differences between classical implementations of the FEM numerical integration algorithm on GPU with explicit data sending between CPU and Accelerator and the SVM implementation with the transfer hidden from the user. Research should answer whether the advantages of using a weaker integrated card with faster data transfer will overcome the external graphics card's connection bottleneck.

INTRODUCTION

Two main trends exist in the evolution of computer architectures used in scientific computing. The first concerns the growing number of processors' computing cores and wide register units. The second, however, coincides with using specialised accelerators to accelerate the most demanding parts of the code. In the development of modern computing accelerators, most architectures are based on graphics processors, in which there is a SIMD method of computing using thousands of threads running simultaneously. Due to this trend, using graphics cards to accelerate code fragments has become a standard. In home applications, we can observe another trend related to the miniaturisation of specialised equipment used in modern smartphones, tablets, or laptops. Increasing the processing power of this type of equipment is associated with the need to minimise the energy consumed while maintaining high performance. At the junction of these two application types, we can observe interesting designs that combine many specialised cores with a smaller number of general-purpose cores. Such an integrated architecture can save energy using standard cores and maintain high computing power during more complex tasks. In addition to energy efficiency, another advantage of using such architectures is that they are equipped with new technologies for transferring data between different types of cores and memory. When programming accelerators equipped with their memory, there is, unfortunately, a problem with the preparation and transfer of data between the so-called host (most often understood as computer RAM) and accelerator memory. Although high-speed interfaces connect modern GPUs, this usually represents a bottleneck in overall program performance. Hence, there is an opportunity to use heterogeneous architectures for scientific computing to use high-speed inter-core and memory buses efficiently. With the introduction of various systems with integrated graphics and general-purpose cores on a single chip, the need to support RAM access is emerging. The first exciting system with complete unification of special-purpose and standard cores was the IBM Cell processor used in the PlayStation 3. The architecture allows fast data transfers from memory to SPUs and efficient data exchange between standard IBM Power cores. These capabilities, combined with energy efficiency, have resulted in the widespread use of this architecture in scientific computing. In our previous work, we explored the possibility of using this architecture for specific parts of calculations using the Finite Element Method (Krużel and Banaś, 2013).

The problem of data exchange between different types of computing cores plays a vital role in all heterogeneous architectures. As a result, several different strategies have emerged at the software and hardware layers. In June 2012, some significant players in heterogeneous computing, including AMD and ARM, formed the Heterogeneous System Architecture Foundation to develop technology standards for supporting computing on different types of cores (HSA Foundation, 2013). The collaboration resulted in the developing of the Heterogeneous Unified Memory Model (hUMA), first introduced in the AMD Accelerated Unit processor (Kyriazis, 2012). Almost simultaneously with the development of the hUMA model, Khronos Group intro-

Communications of the ECMS, Volume 37, Issue 1,
Proceedings, ©ECMS Enrico Vicario, Romeo Bandinelli,
Virginia Fani, Michele Mastroianni (Editors) 2023
ISBN: 978-3-937436-80-7/978-3-937436-79-1 (CD) ISSN 2522-2414

duced the OpenCL 2.0 specification with Shared Virtual Memory capabilities, which unifies CPU and GPU memory on the developer side, and in HSA systems can leverage the hardware capabilities of hUMA (Howes, 2014). With OpenCL 2.0, this can be used as shared memory for CPU and GPU cores (Graczyk, 2013). In our previous work, we tested the possibilities of use of the hUMA and SVM on the AMD Accelerated Processing Unit A10-7850 codenamed "Kaveri" with the encouraging result of achieving a 30% speed-up compared to the model with the sending data between two separate memory regions (Krużel and Banaś, 2015).

With the introduction of OpenCL 2.0 capabilities to Nvidia GPUs in 2017, there is also the possibility of testing SVM features using externally connected GPUs (Hindriksen, 2017).

The graphics card market is dominated by two leading players, Nvidia and AMD. Their graphics cards and GPU-based accelerators are characterised by a very similar structure based on arrays of stream processors grouped into larger units called multiprocessors. The growing market for external accelerators forced Intel, which is supreme in producing general-purpose processors, to search for a solution in this area. Based on the broad vector registers that characterise the Cell/BE, Intel began to build the Larabee graphics card architecture. Due to this architecture, the company tried to overcome the main barrier to the wider use of accelerator programming techniques, which was a complex model and programming method (Seiler et al., 2008). At the same time, Intel was working on the Single-Chip Computer and Teraflops Research Chip projects, which feature a huge multicore structure. Based on these projects, the Intel MIC (Many Integrated Core) architecture was developed, which was used in Intel Xeon Phi coprocessors, codenamed Knights Corner (KNC). The MIC architecture has been touted as one that combines the power of graphics accelerators with the programming ease of processors. The next generation of the MIC architecture, Knights Landing, was offered as a separate PCI-Express card and a standalone processor. Intel Xeon Phi was a significant part of the most powerful computer systems in the world, and in June 2015, the use of this type of accelerator reached 34% of all systems (Strohmaier et al., 2020). Although the Intel Xeon Phi architecture has officially been discontinued (Intel, 2018), its evolution has led to the creation of Intel Xe graphics cards, which were announced in November 2019 (Intel, 2019). The new architecture was mentioned to avoid repeating Xeon Phi errors, provide a unified programming model (oneAPI), and provide exceptional performance (Cutress, 2019). Because Intel Xe-LP GPUs are integrated into the Intel Core processors, and the latter solution is often sold as a part of gaming laptops equipped with the second, more powerful GPU, the authors decided to test the possibility of using OpenCL Shared Virtual Memory mechanism on this type of machine. For tests, we used an ASUS TUF DASH F15 laptop computer equipped with an Intel i7 11370 CPU with the integrated Intel Iris GPU and an additional GPU, Nvidia GeForce RTX 3060 Laptop. For the algorithm tested, we used our auto-tuned FEM numerical integration, which we have already tested on modern graphics processors (Banaś et al., 2020), CPUs (Krużel, 2019), hybrid systems (Krużel and Banaś, 2015), as well as Intel Xeon Phi (Banaś and Krużel, 2014), and Intel Xe-LP (Krużel and Nytko, 2022). Extending previous research to include a new architecture may indicate whether integrated GPUs can compete with the external GPU when the bottleneck connected with the data transfer is eliminated (or hidden from the user).

NUMERICAL INTEGRATION

Finite element method procedures turned out to be one of the most challenging engineering tasks related to the use of accelerators. One of the essential parts of the finite element method is numerical integration, used to prepare elementary stiffness matrices for the solving system. Most research on using accelerator computing power has focused on using GPUs to speed up the solution of the final system of linear equations (Geveler et al., 2013; Buatois et al., 2009). Often, this procedure is optimised first, as it is the most time-consuming part of FEM. However, once the process has been optimised, the earlier steps of the computation, such as numerical integration or assembling the entire matrix, also significantly affect the execution time (Mamza et al., 2012).

Numerical integration in the finite element method is correlated with the applied geometry and the approximation type of the given elementary shape functions. Therefore, an appropriate geometric transformation must be applied to the mesh geometry used for the calculations. By denoting the physical coordinates in the mesh as $\boldsymbol{x}$, the transformation from the reference element with coordinates $\boldsymbol{\xi}$ is marked as $\boldsymbol{x}(\boldsymbol{\xi})$. Typically, it is obtained as a linear, multilinear, square, cubic, or other transformation of the underlying geometric functions and the set of degrees of freedom. The use of the Jacobian matrix $J = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}$ is required for the transformation of the coordinates from the reference element to the real element. This significantly distinguishes this algorithm from other matrix integration and multiplication algorithms. The numerical quadrature converts the analytical integral to the sum of the integration points in the reference domain. Of the various possible quadratures, we will focus on the most popular Gaussian quadrature (Lyness, 1969). The coordinates in the reference element are marked as $\boldsymbol{\xi}^Q$, and the weights as $\boldsymbol{w}^Q$ where $Q = 1, ..., N_Q$ (N_Q - number of Gauss points depending on the type of element and the degree of approximation applied). In the final numerical integration formula used in our calculations, we use the determinant of the Jacobian matrix $det\boldsymbol{J} = det(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}})$ and get:

$$\int_{\Omega_e} \sum_i \sum_j \boldsymbol{C}^{i;j} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_j} d\Omega \approx$$
$$\approx \sum_{Q=1}^{N_Q} \left(\sum_i \sum_j \boldsymbol{C}^{i;j} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_j} \det \boldsymbol{J} \right) \Bigg|_{\boldsymbol{\xi}^Q} \boldsymbol{w}^Q \quad (1)$$

To standardise and describe the algorithm from the point of view of mathematical calculations, some modifications were made to the above formula, which allows us to create a general numerical integration algorithm for finite elements of the same type and degree of approximation (Alg. 1).

The algorithm's structure allows us to modify the loops' order and use different memory types to store different arrays. This property allowed us to develop a system for automatically tuning the algorithm for different types of accelerators. For the memory transfer test in the numerical integration algorithm, we used the artificial convection-diffusion-reaction problem with a matrix of problem coefficients $\boldsymbol{C}$ different from 0. To complicate the calculations, we have used prismatic elements that can reproduce even the most complex geometry of the computational area Ω (Kallinderis, 1995).

Algorithm 1: Generalised numerical integration algorithm for elements of the same type and degree of approximation

- determine the algorithm parameters – N_{EL}, N_Q, N_S;
- load tables $\boldsymbol{\xi}^Q$ i $\boldsymbol{w}^Q$ with numerical integration data;
- load the values of all shape functions and their derivatives relative to local coordinates at all integration points in the reference element;

for $e = 1$ *to* N_{EL} **do**
- load problem coefficients common for all integration points (Array $\boldsymbol{C}^e$);
- load the necessary data about the element geometry (Array $\boldsymbol{G}^e$);
- initialize element stiffness matrix $\boldsymbol{A}^e$ and element right-hand side vector $\boldsymbol{b}^e$;

for $i_Q = 1$ *to* N_Q **do**
- calculate the data needed for Jacobian transformations ($\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}$, $\frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{x}}$, **vol**);
- calculate the derivatives of the shape function relative to global coordinates using the Jacobian matrix;
- calculate the coefficients $\boldsymbol{C}[i_Q]$ i $\boldsymbol{D}[i_Q]$ at the integration point;

for $i_S = 1$ *to* N_S **do**

for $j_S = 1$ *to* N_S **do**

for $i_D = 0$ *to* N_D **do**

for $j_D = 0$ *to* N_D **do**

$\boldsymbol{A}^e[i_S][j_S] += \boldsymbol{C}[i_Q[[i_D][j_D] \times \boldsymbol{\phi}[i_Q][i_S][i_D] \times \boldsymbol{\phi}[i_Q][j_S][j_D] \times \mathbf{vol}$

end

end

if $i_S = j_S$ **and** $i_D = j_D$ **then**

$\boldsymbol{b}^e[i_S] += \boldsymbol{D}[i_Q][i_D] \times \boldsymbol{\phi}[i_Q][i_S][i_D] \times \mathbf{vol}$

end

end

end

end

- write the entire matrix $\boldsymbol{A}^e$ and vector $\boldsymbol{b}^e$;

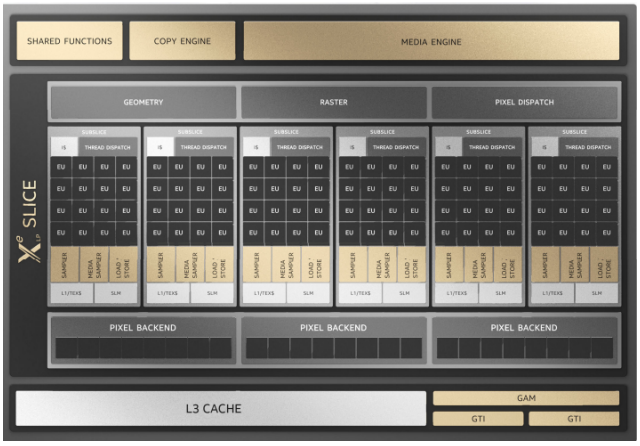
end

USED GPUs

To perform the tests, the authors used the mobile version of the Nvidia GeForce RTX 3060 and the Iris Xe-LP integrated with the Intel i7 11370H processor.

Intel Iris Xe-LP

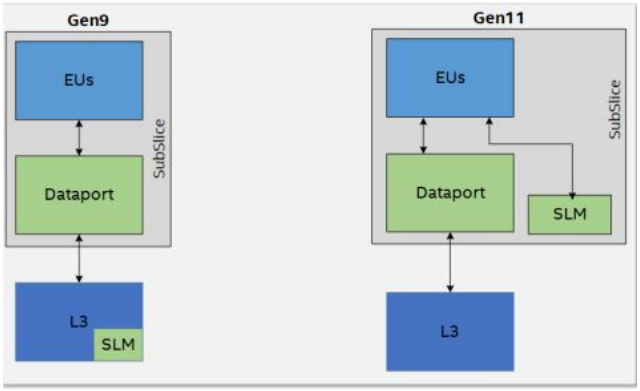
Compared to the previous generation of Intel processors, in Iris Xe-LP architecture, there has been a significant change in the processing power of the graphics unit - now it can even reach 2.2 teraflops (Intel, 2020). The number of execution units (EU) - basic computing units has also increased, which in the previous generation amounted to 64 units, and currently 96 (Fig. 1).



Figures. 1: Intel Iris XE-LP Architecture (Intel, 2020)

The heterogeneous graphics architecture of Intel processors allows physical DRAM to be shared. Compared to PCI-E-based chipsets, sharing physical memory eliminates the communication bottleneck between the computing unit and memory. The great advantage of this solution is the lack of a copy buffer between units, which results in increased computing efficiency while reducing the amount of memory and energy needed to operate. The memory management subsystem in Intel 11th-generation processors has been optimised to minimise latency, and the memory controller scheduling algorithms have been improved, positively affecting the overall memory bandwidth. Theoretically, the memory bandwidth of the embedded Intel Xe GPU can reach 38,4 GB/s. In addition, compared to the pre-

vious generation of processors, the L3 cache memory has been increased - up to 3072 kB. The Iris XE-LP graphics unit is divided into six blocks containing 16 Execution Units (EU). Shared Local Memory (SLM) with a size of 64kB is available for computing units. In the previous generation of processors, this memory was connected to the L3 cache outside the block, which caused delays due to the need to use a data port (Fig. 2). In the generation used, SLM is available directly for Execution Units, significantly reducing delays (Intel, 2020).



Figures. 2: Intel Iris XE-LP Shared Local Memory (Intel, 2020)

Nvidia GeForce RTX 3060

The GeForce RTX 3060 Laptop graphics card is based on the Ampere architecture. The graphics chip is divided into 3 Graphics Processing Clusters (GPC), which contain 5 Texture Processing Units (TPC). Each TPC has 2 clusters with two blocks containing Streaming Multiprocessors (SM) - basic computing units. Each streaming multiprocessor includes 128 kB of L1/shared memory, allocated according to computational needs (Nvidia, 2021).

The Nvidia GeForce RTX 3060 graphics card contains 3072 kB L2 cache memory and 6GB of RAM. The data from the computers' RAM is transferred through the PCI-Express x4 interface, which is theoretically capable of 7,88 GB/s bandwidth.

DEVELOPMENT TOOLS

ModFEM

The primary tool used in the research was the modular software platform for engineering calculations using the finite element method called ModFEM (Michalik et al., 2013). Its modular structure enables the modification of individual parts of the finite element calculation, such as approximation, mesh handling, and solving solvers. The program consists of several levels with separate modules. The main user-managed module is the Problems module, which defines the weak FEM formulation and the other modules the user uses. An extension of the approximation module with appropriate accelerator support was developed during the investigation. The problem modules of the researched tasks were also modified to prepare the data structures properly and test the numerical integration algorithm in the OpenCL environment.

OpenCL Shared Virtual Memory

OpenCL programming is an exceptionally versatile method among the many programming methods used in accelerators. It supports many modern architectures, including GPUs, coprocessors (e.g. Xeon Phi), or CellBE. OpenCL has been gradually developed since 2009, and its 2.0 version was introduced in 2014, bringing out several essential improvements. OpenCL memory model defines several types of memory regions available for programmers. Due to the model's origin in GPU programming, the memory model is based on the physical organisation of typical GPU memory. Due to the portability of OpenCL, each of the memory objects can be mapped differently, depending on the available hardware resources. The variables defined inside the kernel belong to private memory and can be stored in scalar or vector registers. The other memory regions are assigned through specific qualifiers. OpenCL defines three types of memory – global, constant and shared (local). Global memory stores variables visible to all threads executing the kernel; constant memory is also available for all threads but is only accessible for reading. The fastest local memory stores the variables threads share in a single workgroup. Due to the portability of the created code, OpenCL contains procedures that allow adapting to different platforms and devices even without physical equivalents of specific memory types (Rul et al., 2010). Typical behaviour in OpenCL computing was copying the data necessary for calculations from the host memory to the device's global or constant memory for further use. It required proper preparation of the memory buffers on the host side from the programmer side and many hardware or system solutions to copy the data from the host to the accelerator. Even for the integrated solutions with CPU and Accelerator on one die (e.g. CellBE, APU), the memory was partitioned into the host and device parts, which prevents zero-copy operations and takes precious time. The procedure was complicated and time-consuming despite software and hardware solutions to pass data from the host to the device and back. Shared Virtual Memory (SVM) presented in OpenCL 2.0 solves both problems with programming the OpenCL memory model. From the programmer side, it reduces the necessary preparations for the data and the whole mapping and passing pointer operations that take a lot of code lines. From a performance point of view, the HSA-compatible hardware uses the full potential of hUMA, which means there is no copying between the CPU and the accelerator. If the device is not HSA compatible, the framework will handle data transmission in the best possible way. Shared Virtual memory defines a buffer that can be used directly by both a host and an accelerator without unnecessary mappings and copying of the data. OpenCL allows for Fine-Grain or Coarse-Grain memory buffers, depending on the hardware type. De-

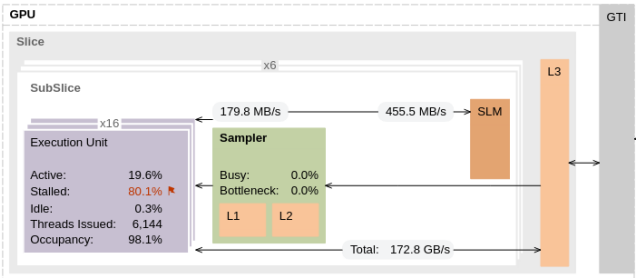
tailed information on SVM OpenCL features can be found in (Intel, 2014; Howes, 2014). Although the Intel Xe-LP card used in this paper is an integrated unit, it does not support Fine-Grain technology and operates as a non-atomic duplicate buffer (Coarse-Grain). Data sharing occurs at the granularity of regions of OCL memory objects. Updates between the host and device occur explicitly through the map and unmap calls. This behaviour can hide data transfer latency in other operations, such as calculations or data preparation (Das et al., 2015). The Nvidia card works in the same mode, but with the introduction of the Nvidia Pascal architecture, some Unified Memory Buffer mechanisms were added to omit the overhead connected with the PCI-Express bottleneck. Some mechanisms include a wide 49-bit virtual addressing and on-demand page migration. It addresses the entire system memory and allows GPU threads to migrate pages from anywhere in the system to GPU memory on demand with maximum efficiency (Sakharnykh, 2016). Also, the driver uses heuristics to maintain data localisation and prevent excessive page faults (Harris, 2017). This feature is very convenient for our Auto-Tuning mechanism because of its multiple repetitions to fit the investigated architecture.

Auto-Tuning

To obtain the best possible results for different types of architecture, the authors developed a system for automatically tuning the numerical integration code. For this purpose, several parameters that affect the algorithm's performance have been characterised. The parameter list in the auto-tuning system is mainly about how memory is handled in the OpenCL model, which is based on the graphics card architecture. Arrays passed as arguments to the numerical integration routine (e.g. geometrical data and problem coefficients) can be used directly in calculations or previously downloaded to local tables stored in registers or shared memory. Using shared memory as a temporary data read buffer, data are read continuously - a single thread reads another memory cell and writes to the shared buffer. The read data can be stored in a buffer and used later for calculations, or written to registers, freeing the buffer for further use. The total number of combinations of these parameters for any architecture is 40. The developed system consists of scripts and code fragments that are responsible for compiling the kernel with the appropriate options. An additional parameter defines the minimum number of threads that can be run on the accelerator and is set to 64 for both GPUs. Detailed information on the auto-tuning mechanism used can be found in (Banaś et al., 2020).

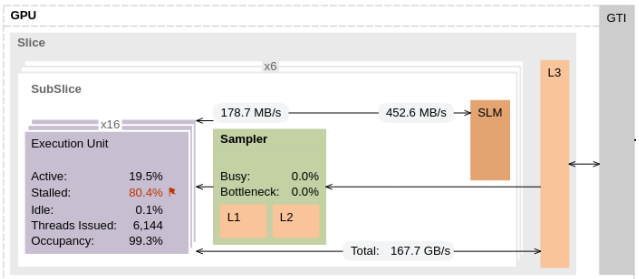
RESULTS

The best results obtained with Auto-Tuning are presented in table 1. As seen in both cases for OpenCL 1.2, the data transfer times are much higher than the computation time. The numerical integration algorithm for the convection-diffusion problem with a linear approximation is a memory-bound task (Banaś et al., 2018). The difference is much more visible in the case of the faster GeForce RTX 3060 card because of its higher computational power. After modification of the code for the use of Shared Memory Buffer, the speed up is extraordinary. In the case of the Iris Xe GPU, all data transfer is hidden under the computation time but can be separated by using internal time measurements in the kernel. As can be seen, data transfer is still more extensive than the calculations, but both values are much smaller than in OpenCL 1.2. To fully understand the behaviour of our code, we have used the Intel Vtune profiler (Fig. 3).



Figures. 3: Execution profile for OpenCL 1.2 version of the code

As can be seen, the memory-bound code has a lot of stalls connected to the necessity of synchronisation with the memory. Even with the 98% occupancy, the GPU is active only for 20% of the total computation time. In OpenCL 1.2 with the explicit memory sending regions, the observed memory bandwidth reaches the maximum at 18 GB/s out of 38 GB/s of the theoretical maximum, which is not bad but indicates that the amount of data sent was too small to use all the bandwidth fully. With the use of SVM, the total occupancy value increases slightly. Still, the average speed of obtaining the data from memory decreases because of spreading it over a more extended period, not only the write and read procedures (Fig. 4).



Figures. 4: Execution profile for Shared Virtual Memory version of the code

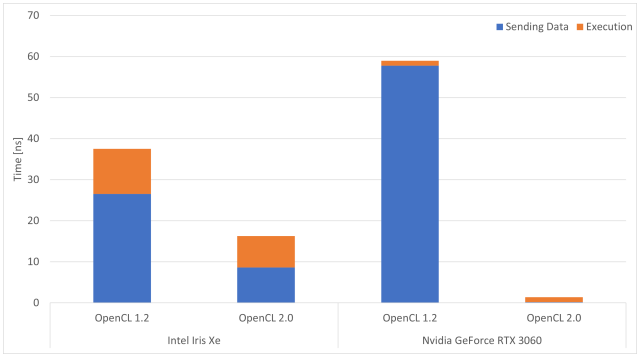
The behaviour of the GeForce RTX 3060 is much more predictable - the calculation time is the same in both versions of OpenCL, but the data transfer time is highly faster. This behaviour is quite different from the Intel Xe case. According to (Ravi, 2016), for the Coarse-grain SVM buffer, transfer times should be hidden in the map/unmap regions of the code. This can be

observed in the Nvidia case, where almost no overhead time is connected to data transfer. But in the Intel case, the latency is associated with the synchronisation points in code(clFinish). This may indicate that Intels' implementation of SVM uses some of the mechanisms which characterised Fine-grain SVM access.

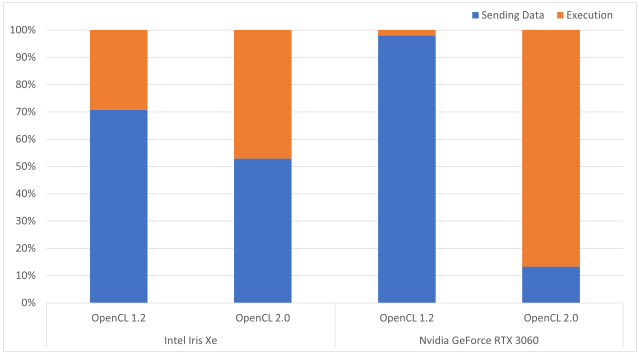
Table 1: Results (in *ns*)

	Intel Iris Xe		Nvidia GeForce RTX 3060	
	OpenCL 1.2	*OpenCL 2.0*	*OpenCL 1.2*	*OpenCL 2.0*
Sending Data	26.52	8.61	57.77	0.19
Execution	10.99	7.66	1.20	1.22

The differences in execution time are visible in figure 5. In the OpenCL 1.2 case, the total time spent for the whole calculation process with the data transfer is higher for the external GPU, despite its faster processing speed. This calls into question the validity of using external graphics cards for scientific and technical computing with relatively large data sizes in relation to the computation. SVM technology significantly reduces the data transfer time and gains an impressive speedup compared to the embedded Intel solution.



Figures. 5: Comparison of execution results with separate data transfer analysis



Figures. 6: Percentage of data transfer and calculations

Figure 6 shows the ratio of calculation to computation in each case. As can be observed in the Intel Iris Xe, the time spent on data transfer is reduced by 20% with OpenCL 2.0. In the Nvidia case, we see that without SVM, the time spent on data transfer is 98% of all time. With OpenCL 2.0 techniques to hide the transfer in code map/unmap regions, we can reduce this time to 13% of the total time.

CONCLUSIONS

In summary, we can conclude that SVM can significantly speed up the total computation time by reducing the data exchange between the host and the accelerator. In addition, the simplified method of transferring data to the accelerator is convenient for developers and encourages more extensive use of GPU computing power in the newly developed software. All these observations align with the result we obtained in (Krużel and Banaś, 2015), where we tested fully HSA-compatible hardware using the AMD Accelerated Processing Unit as an example. Although the architectures tested in this paper can only use the most straightforward implementation of shared virtual memory, we can see a significant speedup. The data obtained from the Intel Profiler shows that we have to test this feature on more powerful GPUs with more demanding tasks to see if it would significantly reduce the data transfer time in various applications. Despite that, the extensive memory addressing mode in modern external GPUs can overcome the bottleneck connected with the additional link via the PCI_Express port and using the separate memory for the card. In our future work, we will test this feature on more powerful cards to fully utilize the available bandwidth and see if the difference between the external GPUs and the internal solutions is more visible.

REFERENCES

Banaś, K. and Krużel, F. (2014). OpenCL performance portability for Xeon Phi coprocessor and NVIDIA GPUs: A case study of finite element numerical integration. In *Euro-Par 2014: Parallel Processing Workshops*, volume 8806 of *Lecture Notes in Computer Science*, pages 158–169. Springer International Publishing.

Banaś, K., Krużel, F., and Bielański, J. (2020). Optimal kernel design for finite element numerical integration on GPUs. *Computing in Science and Engineering*, Volume 22(Issue 6):61–74.

Banaś, K., Krużel, F., Bielański, J., and Chłoń, K. (2018). A comparison of performance tuning process for different generations of NVIDIA GPUs and an example scientific computing algorithm. In Wyrzykowski, R., Dongarra, J., Deelman, E., and Karczewski, K., editors, *Parallel Processing and Applied Mathematics*, pages 232–242, Cham. Springer International Publishing.

Buatois, L., Caumon, G., and Levy, B. (2009). Concurrent number cruncher: A GPU implementation of a general sparse linear solver. *Int. J. Parallel Emerg. Distrib. Syst.*, 24(3):205–223.

Cutress, I. (2019). Intel's xe for hpc: Ponte vecchio with chiplets, emib, and foveros on 7nm, coming 2021. *AnandTech*.

Das, D., Raghavendra, P., Gupta, M., and Tye, T.

(2015). Implementing cross-device atomics in heterogeneous processors.

Geveler, M., Ribbrock, D., Göddeke, D., Zajac, P., and Turek, S. (2013). Towards a complete FEM-based simulation toolkit on GPUs: Unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses. *Computers & Fluids*, 80:327 – 332. Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011.

Graczyk, R. (2013). Intel Iris Pro 5200 test; Crysis 3 na integrze? Accessed on 10th Febuary 2015.

Harris, M. (2017). Unified memory for cuda beginners. *Nvidia Developer*.

Hindriksen, V. (2017). NVidia (finally) opens door to OpenCL 2.0. *Linkedin Pulse*.

Howes, L.; Munshi, A. (2014). *The OpenCL Specification*. Khronos OpenCL Working Group. version 2.0, revision 26.

HSA Foundation (2013). *http://www.hsafoundation.com*.

Intel (2014). *OpenCL 2.0 Shared Virtual Memory Overview*.

Intel (2019). Intel Unveils New GPU Architecture with High-Performance Computing and AI Acceleration, and oneAPI Software Stack with Unified and Scalable Abstraction for Heterogeneous Architectures. *Intel Newsroom*.

Intel (2020). *Intel Architecture Day 2020 Presentation Slides*. Whitepaper.

Intel (July 23, 2018). Product change notification 116378 - 00.

Kallinderis, Y. (1995). Adaptive hybrid prismatic-tetrahedral grids. *International Journal for Numerical Methods in Fluids*, 20:1023–1037.

Krużel, F. (2019). Vectorized implementation of the FEM numerical integration algorithm on a modern CPU. In *European Conference for Modelling and Simulation*, volume Volume 33, pages 414–420.

Krużel, F. and Banaś, K. (2013). Vectorized OpenCL implementation of numerical integration for higher order finite elements. *Computers and Mathematics with Applications*, 66(10):2030–2044.

Krużel, F. and Banaś, K. (2015). AMD APU systems as a platform for scientific computing. *Computer Methods in Materials Science*, 15(2):362–369.

Krużel, F. and Nytko, M. (2022). Intel iris xe-lp as a platform for scientific computing. In Ganzha, M., editor, *Communication Papers of the 17th Conference on Computer Science and Intelligence Systems, September 4-7, 2022, Sofia, Bulgaria*, number Vol. 32 in Annals of Computer Science and Information Systems, pages 121–128, Warszawa. PTI.

Kyriazis, G. (2012.). Heterogeneous system architecture: A technical review. Technical report, AMD. revision 1.0.

Lyness, J. N. (1969). Quadrature methods based on complex function values. *Mathematics of Computation*, 23(107):601–619.

Mamza, J., Makyla, P., Dziekoński, A., Lamecki, A., and Mrozowski, M. (2012). Multi-core and multiprocessor implementation of numerical integration in Finite Element Method. In *Microwave Radar and Wireless Communications (MIKON), 2012 19th International Conference on*, volume 2, pages 457 – 461.

Michalik, K., Banaś, K., Płaszewski, P., and Cybułka, P. (2013). ModFEM – a computational framework for parallel adaptive finite element simulations. *Computer Methods in Materials Science*, 13(1):3–8.

Nvidia (2021). *NVIDIA Ampere GA102 GPU Architecture: Ampere GA10x*. Whitepaper.

Ravi, K. R. (2016). Performance considerations for opencl on nvidia gpus. In *GPU Technology Conference, APril 4-7, 2016, Silicon Valley*. Nvidia.

Rul, S., Vandierendonck, H., D'Haene, J., and De Bosschere, K. (2010). An experimental study on performance portability of OpenCL kernels. In *Application Accelerators in High Performance Computing, 2010 Symposium, Papers*, page 3, Knoxville, TN, USA.

Sakharnykh, N. (2016). Beyond gpu memory limits with unified memory on pascal. *Nvidia Developer*.

Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., and Dubey, P. (2008). Larrabee: a many-core x86 architecture for visual computing. *SIGGRAPH 08: ACM SIGGRAPH 2008 papers*, pages 1–15.

Strohmaier, E., Dongarra, J., Simon, H., Meuer, M., and Meuer, H. (2020). *Top500 The List*.

FILIP KRUŻEL is an assistant professor at the Institute of Computer Science of the Cracow University of Technology. His scientific interests focus on the high-performance computing and the use of various types of accelerators and non-standard hardware architectures. His e-mail address is: *filip.kruzel@pk.edu.pl* and his Webpage can be found at *https://ii.pk.edu.pl/~fkruzel*.

MATEUSZ NYTKO is an research and teaching assistant at the Institute of Computer Science of the Cracow University of Technology. His research interests focus on multiprocessor architectures and high performance computing. His e-mail address is: *mateusz.nytko@pk.edu.pl*.