Communication Papers of the of the 17th Conference on Computer Science and Intelligence Systems pp. 121–128
DOI: 10.15439/2022F132
ISSN 2300-5963 ACSIS, Vol. 32

Intel Iris Xe-LP as a platform for scientific computing

Filip Krużel
Cracow University of Technology
ul. Warszawska 24, 31-155 Kraków, Poland
Email: filip.kruzel@pk.edu.pl

Mateusz Nytko
Cracow University of Technology
ul. Warszawska 24, 31-155 Kraków, Poland
Email: mateusz.nytko@pk.edu.pl

***Abstract*—In the present article, we describe the implementation of the finite element numerical integration algorithm for the Intel Iris Xe-LP Graphics Processing Unit. This GPU is a direct successor of a Xeon Phi accelerator architecture. Although it is used in integrated circuits and does not offer substantial performance, its test should be treated as a preview of the estimated performance for the Intel HPG Graphics Cards that are announced to be released in 2022. In the article, we use our previously developed auto-tuning Finite Element numerical integration OpenCL code on the Intel Iris Xe-LP GPU integrated into the Intel i7 11370H CPU and compare the results with the Nvidia GeForce RTX 3060 GPU. This article brings the answer to the question of whether the new Intel architecture can be a direct competitor to the more classic GPU architecture. It also allows showing if the new architecture can be used for the computation of complex engineering tasks.**

I. Introduction

DURING the evolution of computer architectures used in scientific computing, we can observe two main trends. The first is connected to the increasing number of computing cores and wide register units in CPUs. The second coincides with the use of specialized accelerators used to speed up the most demanding fragments of code. In the development of the modern computing accelerators, most of the architectures were based on GPUs, which occur with the SIMD manner of calculation with the use of the thousands of threads operating simultaneously. Since the presentation of the first Nvidia Tesla in 2007 [1], the architecture of the most of the accelerators used in high-performance systems is based on the Nvidia GPUs. According to the top500 list in November 2021, almost 30% of the supercomputers have Nvidia GPU-based accelerators (Fig. 1).

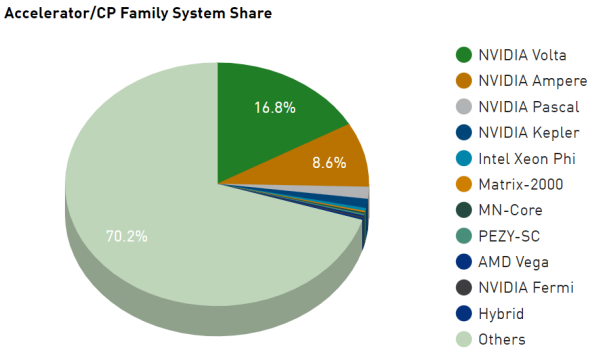

Fig. 1. Top500 Accelerators [2]

Despite these trends, there have also been attempts to find a middle ground between the high computational power provided by GPU-based accelerators and the relative ease of programming and versatility of CPUs. This type of accelerator combines both - the relatively large amount of computing cores with the extensive registers. As an example of such an architecture, the CELL Broadband Engine developed by IBM can be chosen. This architecture consists of one general processing core and eight smaller specialistic cores (Synergistic Processor Elements) equipped with wide 128-bit vector registers and AL units. These two types of accelerators, developed by Nvidia and IBM, inspired Intel, their main competitor, to search for its solution in the area of computing accelerators. Based on the wide vector registers which characterise the Cell/BE architecture, Intel started to create Larabee's graphics card architecture. With this architecture, the company attempted to overcome the main barrier to wider adoption of accelerators programming techniques, which was the complex model and programming method. The main advantages of the designed architecture were extensive (512-bit) vector registers, specialised texture units, coherent memory hierarchy and compatibility with x86 architecture [3]. At the same time, Intel was working on the Single-Chip Computer and Teraflops Research Chip projects characterised by a huge multi-core structure. Based on these designs, the Intel MIC (Many Integrated Core) architecture was developed and used in the Intel Xeon Phi co-processors codenamed Knights Corner(KNC) [4]. The MIC architecture was advertised as an architecture that could combine the power of GPU accelerators with the ease of programming that characterizes processors. The next generation of the MIC architecture, Knights Landing, was offered as a separate PCI-express card and a stand-alone CPU. Intel Xeon Phi was a significant part of the most powerful computer systems in the world and in June 2015 the usage of this type of accelerator reached 34% of all systems [2]. Even though Intel Xeon Phi architecture was officially discontinued [5], its evolution led to the Intel Xe graphics cards, which were announced in November 2019 [6]. The new architecture was mentioned

not to repeat the Xeon Phi's mistakes, provide a unified programming model (oneAPI), and outstanding performance [7]. Intel Xe has different variants of microarchitectures, from integrated/low power consumption (Xe-LP) to enthusiast/high-performance gaming (Xe-HPG), data centre/high performance (Xe-HP) and high-performance computing (Xe-HPC) [8]. Due to the continuous unavailability of more advanced solutions (Xe-HPG/HPC), the authors decided to test the only available version of Intel's new architecture - Xe-LP. To do this, we try to port the numerical integration algorithm on the Intel Iris Xe GPU. Our previous study includes the development of the algorithm for modern GPUs [9], [10], CPUs [11], Hybrid systems [12] as well as Intel Xeon Phi [13], [14], [15]. Extending previous research into the new architecture may provide clues as to whether Intel's direction will allow it to compete with solutions currently dominating the market.

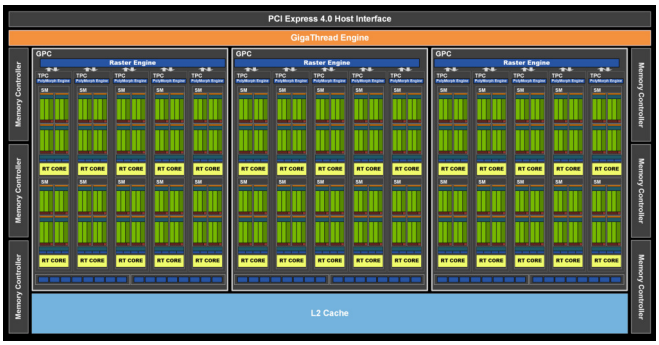

Fig. 2. GeForce RTX 3060 Laptop [16]

II. Graphic Processor Unit (GPU)

To perform the tests, the authors used a mobile version of the GeForce RTX 3060 graphics card as the reference chip, and the Intel i7 11370H processor's integrated graphics chip, Intel Iris XE-LP. Both chips tested were built into the one laptop. The GeForce RTX 3060 graphics card is based on the Ampere architecture [16]. As we can see on the figure 2 the GPU is divided into 3 GPC (Graphics Processing Clusters), which consist of 5 TPC (Texture Processing Clusters). Each TPC cluster contains 2 blocks of streaming multiprocessors (SM), which adds up to 30 SM blocks. The applied GigaThread Engine acts as a scheduler distributing work among streaming multiprocessors.

Each streaming multiprocessor contains 64 cores running at single precision, giving a total of 3840 cores. Additionally, each SM contains 4 Tensor cores, one Ray Tracing core, a 256kb register file, four texture units, and 128KB of L1 Cache/Shared memory [16].

The research used the Intel i7 11370 processor - a quad-core and eight-threaded unit with a base core clock frequency of 3.3 GHz. Built on the Tiger Lake architecture, it was first unveiled in autumn 2020. The processor is equipped with a graphics chip based on the Intel XE-LP architecture. Compared to previous generations, the aforementioned graphics chip architecture can reach a computing power of 2.2 teraflop.

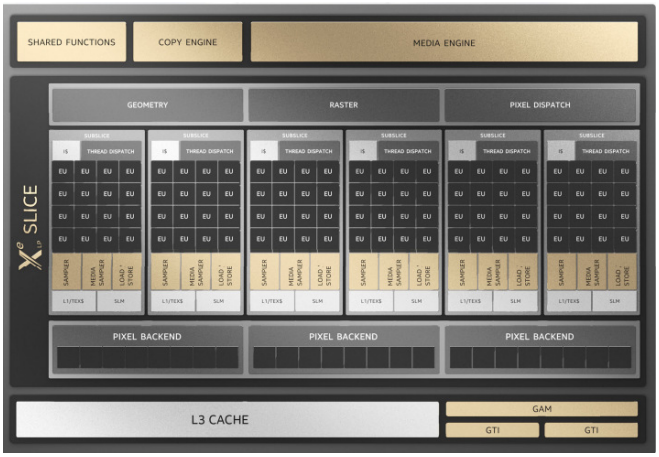

Fig. 3. Intel Iris XE-LP architecture [17]

The architecture of the Intel XE-LP chip is divided into 6 blocks (Fig. 3). There are 16 Execution Units in each of them, which makes 96 of them in total. The execution unit is the smallest block in the architecture which is multithreaded itself - it can execute 7 threads [18]. The computing unit consists of 8 SIMD units (single instruction multiple data) used to perform single and double-precision operations, and 2 SIMD units for extended match operations [17]. The executing units are combined in pairs in which the work is divided using the Thread Controler.

III. Numerical Integration

One of the most challenging engineering tasks in the use of accelerators has proven to be Finite Element Method procedures. One of the fundamental parts of FEM is numerical integration, used to prepare elementary stiffness matrices for the system solver. Most research on the use of accelerator computing power has focused on the use of GPUs to accelerate the solution of the final system of linear equations [19], [20]. Often the solution procedure is optimised first, as it is the most time-consuming part of FEM. However, once the procedure mentioned above is optimised, the earlier computational steps, such as numerical integration or assembling the whole matrix, also significantly affect the execution time [21].

Early research on transferring numerical integration to GPUs for specific FEM applications has been presented for, among others, linear elasticity with higher-order approximation [22], [23], nonlinear elasticity [24] and electromagnetism [25], [26]. A comprehensive study of the finite element assembly process on GPUs is described in [27], [28]. A separate set of papers dealt with attempts to develop tools for the automatic generation of finite element codes [29] in the context of GPU computing described in [30], [31]. Recent publications have extended the scope of research to include energetic aspects of numerical integration and assembly [32] and modern approximations such as isogeometric analysis [33]. Previous Intel architecture, Xeon Phi was also very widely used in finite element method procedures. It included

solving a differential equation using numerical integration in TVD Methods [34], the elastodynamic finite integration technique [35], using the shifted boundary method to solve a FEM problem [36] or solving partial differential equations using hybridised discontinuous Galerkin discretisation [37]. The research presented in this paper is part of a trend to exploit the possibilities of accelerating FEM calculations with modern accelerators.

A. Finite Element Method

The finite element method is used to compute an approximate solution of partial differential equations defined for a given, usually three-dimensional computational area Ω along with given boundary conditions $\partial\Omega$ [38], [39], [40], [41]. The computational area is divided into several elements with simple geometry (tetrahedrons, cubes or prisms). Computations are performed using a so-called weak formulation that defines the problem to be solved.

The approach chosen in this paper is based on the most commonly used assumption that each element in the computational domain is integrated only once. The integration results in a small elementary stiffness matrix whose single element is calculated in the form:

$$(A^e)^{rs} = \int_{\Omega_e} \left(\sum_i \sum_j \boldsymbol{C}^{i;j} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_j} + \sum_i \boldsymbol{C}^{i;0} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \boldsymbol{\phi}^s + \right. \\ \left. + \sum_i \boldsymbol{C}^{0;i} \boldsymbol{\phi}^r \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_i} + \boldsymbol{C}^{0;0} \boldsymbol{\phi}^r \boldsymbol{\phi}^s \right) d\Omega + \mathrm{BCL}, \tag{1}$$

where r and s are local indices from 1 to N_S - the number of shape functions for a given element.

Similarly, the vector on the right-hand side is calculated from the formula:

$$(b^e)^r = \int_{\Omega_e} \left(\sum_i \boldsymbol{D}^i \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} + \boldsymbol{D}^0 \boldsymbol{\phi}^r \right) d\Omega + \mathrm{BCR}, \tag{2}$$

BCL and BCR denote words related to boundary conditions defined for a single element and its shape function.

B. Numerical Integration Algorithm

The numerical integration in the Finite Element Method is correlated with the geometry used and the type of approximation by given elementary shape functions. Therefore, an appropriate geometric transformation must be applied to the mesh geometry used for the calculations. Denoting the physical coordinates in the mesh as $\boldsymbol{x}$, the transformation from the reference element with coordinates $\boldsymbol{\xi}$ is denoted as $\boldsymbol{x}(\boldsymbol{\xi})$. It is usually obtained by the general form of a linear, multilinear, quadratic, cubic or other transformation of the geometric basis functions and the set of degrees of freedom. The use of the Jacobian matrix $J = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}$ is required to transform the coordinates from the reference to the real element, and the whole process is the distinguishing part of numerical integration in the Finite Element Method. This significantly contrasts this algorithm from other integration and matrix multiplication algorithms. A numerical quadrature transforms an analytic integral into a sum over integration points in the reference domain. Of the various possible quadratures, we will focus on the most popular Gauss quadrature [42]. The coordinates in the reference element are denoted as $\boldsymbol{\xi}^Q$ and the weights as $\boldsymbol{w}^Q$ where $Q = 1, ..., N_Q$ (N_Q - the number of Gauss points depending on the type of element and the approximation degree used). 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 obtain:

$$\int_{\Omega_e} \sum_i \sum_j \boldsymbol{C}^{i;j} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_j} d\Omega \approx \\ \approx \sum_{Q=1}^{N_Q} \left(\sum_i \sum_j \boldsymbol{C}^{i;j} \frac{\partial \boldsymbol{\phi}^r}{\partial \boldsymbol{x}_i} \frac{\partial \boldsymbol{\phi}^s}{\partial \boldsymbol{x}_j} \det \boldsymbol{J} \right) \Bigg|_{\boldsymbol{\xi}^Q} \boldsymbol{w}^Q \tag{3}$$

In order to unify and describe the algorithm from the point of view of mathematical computation for the most efficient implementation on hardware, some modifications have been made to the above formulas by introducing the following indices:

- $\boldsymbol{\xi}^Q[i_Q]$, $\boldsymbol{w}^Q[i_Q]$ – tables with local coordinates of integration points (Gauss points) and weights assigned to them, $i_Q = 1, 2, ..., N_Q$, where N_Q – number of Gauss points depending on geometry and type and the chosen approximation degree,
- $\boldsymbol{G}^e$ – element geometry data table (related to the transformation from reference element to real element),
- $\mathbf{vol}^Q[i_Q]$ – table with volumetric elements $\mathbf{vol}^Q[i_Q] = det\left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}}\right) \times \boldsymbol{w}^Q[i_Q]$,
- $\boldsymbol{\phi}[i_Q][i_S][i_D]$, $\boldsymbol{\phi}[i_Q][j_S][j_D]$ – tables with the values of consecutive locally shaped functions and their derivatives relative to global ($\frac{\partial \phi^{i_S}}{\partial x_{i_D}}$) and local coordinates ($\frac{\partial \hat{\phi}^{i_S}}{\partial \xi_{i_D}}$) at subsequent integration points i_Q,
  - $i_S, j_S = 1, 2, ..., N_S$, where N_S – the number of shape functions depending on the chosen geometry and the degree of approximation,
  - $i_D, j_D = 0, 1, ..., N_D$, where N_D – dimension of space. For i_D, j_D different from zero, the tables refer to derivatives with respect to the coordinate with index i_D, and for $i_D = 0$ to the shape function, so $i_D, j_D = 0, 1, 2, 3$,
- $\boldsymbol{C}[i_Q][i_D][j_D]$ – table with values of the problem coefficients (material data, values of degrees of freedom in previous nonlinear iterations and time steps, etc.) at successive Gauss points,
- $\boldsymbol{D}[i_Q][i_D]$ – table with the values of the coefficients d^i at subsequent Gauss points,
- $\boldsymbol{A}^e[i_S][j_S]$ – an array storing the local, elementary stiffness matrix,
- $\boldsymbol{b}^e[i_S]$ – array storing the local, elementary right-hand side vector.

Using the notation presented, general formula for the elementary stiffness matrix was created:

$$\boldsymbol{A}^e[i_S][j_S] = \sum_{i_Q}^{N_Q} \sum_{i_D,j_D}^{N_D} \boldsymbol{C}[i_Q][i_D][j_D]\times \\ \times\boldsymbol{\phi}[i_Q][i_S][i_D] \times \boldsymbol{\phi}[i_Q][j_S][j_D] \times \mathbf{vol}^Q[i_Q], \tag{4}$$

A right-handed vector formula was created analogously:

$$\boldsymbol{b}^e[i_S] = \sum_{i_Q}^{N_Q} \sum_{i_D}^{N_D} \boldsymbol{D}[i_Q][i_D]\times \\ \times\boldsymbol{\phi}[i_Q][i_S][i_D] \times \mathbf{vol}^Q[i_Q]. \tag{5}$$

Through the notation introduced, we create a general numerical integration algorithm for finite elements of the same type and approximation degree (Alg. 1).

The optimal structure of the algorithm must take into account the capabilities of the hardware for which the algorithm should be developed. For external accelerators usually connected through some limiting interface, the cost of transferring data to and from the accelerator can be very high and should be hidden by a sufficiently large number of calculations. This situation is favoured by the designed form of the algorithm 1, where the outer loop is a loop over all elements. Also, the general form of the algorithm 1, which does not take into account the location of the data at different memory levels, allows us to treat each inner loop as independent and to change its order for optimal performance. We can also achieve this because all the necessary data can be calculated in advance and used when needed. This allows us to create different variations of the algorithm depending on the hardware used. We further reference these versions of the algorithm as SQS and SSQ in opposite to the reference QSS type presented on algorithm 1. In this notation, the letters indicate the order of the loops, where Q denotes the loop over Gauss points, and S represent the loop over shape functions.

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 $\boldsymbol{\xi}^Q$ and $\boldsymbol{w}^Q$ 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 $\boldsymbol{C}[i_Q]$ and $\boldsymbol{D}[i_Q]$ 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**
            - $\boldsymbol{A}^e[i_S][j_S] += \boldsymbol{C}[i_Q[[i_D][j_D]\times \times\boldsymbol{\phi}[i_Q][i_S][i_D]\times\boldsymbol{\phi}[i_Q][j_S][j_D]\times \times\mathbf{vol}$
            - **if** $i_S = j_S$ **and** $i_D = j_D$ **then**
              - $\boldsymbol{b}^e[i_S] += \boldsymbol{D}[i_Q][i_D]\times \times\boldsymbol{\phi}[i_Q][i_S][i_D]\times$ **vol**
            - **end**
          - **end**
        - **end**
      - **end**
    - **end**
  - **end**
  - write the entire matrix $\boldsymbol{A}^e$ and vector $\boldsymbol{b}^e$;
- **end**

IV. Problems solved

To test the algorithm for both low and high-intensity tasks, the Poisson and convection-diffusion problems were chosen. The former task is characterised by the fact that its exact solution is known so that its correctness can be tested. It also requires relatively few resources, allowing fine-grained algorithms performing a large number of relatively small tasks to be tested. In contrast to Poisson, the convection-diffusion-reaction problem is resource-intensive, allowing testing of a coarse-grained implementation with fewer large elements to compute. Selecting these two tasks allows in-depth testing of the hardware for tasks commonly encountered in FEM and scientific and engineering computing.

A. Poisson equation

The first of the studied problems - Poisson equation can f.e. describe stationary temperature distribution (6).

$$\nabla^2 u = f \tag{6}$$

For the Poisson task, the matrices of the coefficients of the convection-diffusion-reaction $\boldsymbol{C}[i_Q]$ are of the form (7), for all integration points.

$$\boldsymbol{C}[i_Q] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

This allows the individual words of the stiffness matrix to be obtained according to a simplified formula (8).

$$(A^e)^{rs} = \int_{\Omega_e} \sum_i \sum_j \frac{\partial \phi^r}{\partial \boldsymbol{x}_i} \frac{\partial \phi^s}{\partial \boldsymbol{x}_j} d\Omega \tag{8}$$

The $\boldsymbol{D}[i_Q]$ coefficients vector has the form (9).

$$\boldsymbol{D}[i_Q] = \begin{bmatrix} 0 & 0 & 0 & S_v \end{bmatrix} \tag{9}$$

where S_v is a right-hand-side coefficient, different for each integration point.

B. Generalized convection-diffusion-reaction problem

Another of the tasks studied was the generalised convection-diffusion-reaction problem. In order to maximise the use of resources, it was assumed that the array $\boldsymbol{C}[i_Q]$ and the coefficients of the vector $\boldsymbol{D}[i_Q]$ would be fully filled with values different than 0. Arrays of this type appear, for example, in the convective heat transfer problem, after SUPG stabilisation has been applied.

For the studied convection-diffusion-reaction problem, the coefficients $\boldsymbol{C}[i_Q]$ and $\boldsymbol{D}[i_Q]$ are constant for the whole element. This allows us to generalise the solved problem by freeing it from the details of specific applications while maintaining the increased computational intensity of the algorithm.

C. Approximation

The finite element method is based on discretising the considered continuous area into a specified number of elements. These elements are usually simple geometry elements such as tetrahedrons, cubes or prisms. In the cases considered in this work, prismatic elements were used.

With this type of elements, it is possible to reproduce even the most complex geometry of the computational area Ω [43].

For the standard first-order linear approximation, the degrees of freedom of an element are related to its vertices. The basic shape functions for a reference prismatic element are in the form of a combination of 2D function that depends on x and y coordinations of the given vertice, with the function dependent on coordinate z.

V. Development tools

A. ModFEM

A modular software platform for finite element engineering calculations, ModFEM [44], was used as the primary tool used in the research. Thanks to its modular design, it allows modifying individual parts of FEM calculations, such as approximation, mesh handling and solution solvers.

The program consists of several levels on which the different modules are located. The main module managed by the user is the problems module, which defines the FEM weak formulation and determines which other modules the user uses. The most important modules are the mesh module, the approximation module and the solver module. The rest of the modules in the ModFEM code are used for different kinds of division of labour over several types of hardware. Thanks to its structure, this framework allows working in a distributed environment, which makes it an excellent tool for FEM computations both on accelerators and single computers, as well as in high-performance computing environments such as clusters or specialised supercomputers. In the course of the research, an extension to the approximation module was developed with appropriate accelerator support. The problem modules of the investigated tasks were also modified in order to properly prepare data structures and to comparatively test the numerical integration algorithm in the OpenCL environment.

B. OpenCL

The OpenCL language was used to test different types of accelerators. OpenCL allows programming of virtually any multi-core and vector machines - from modern CPUs to GPUs, through hybrid PowerXCell units, APUs and Intel Xeon Phi accelerators. The OpenCL specification includes a C99-based programming language for programming accelerators and an application programming interface (API) for platform support (defined as a given combination of available computing hardware and system software) and execution on processors [45]. Due to the portability of OpenCL code between devices of different types, each memory area can be physically mapped differently depending on the available hardware resources. OpenCL includes routines to preserve the portability of the code between different hardware platforms by adapting the memory and execution models to a given architecture [46]. Direct implementation on a given machine depends on the OpenCL development platform provider and the corresponding drivers.

With OpenCL technology, it has become possible to write programs on many types of accelerators. However, this has not solved the problem of performance portability, as each architecture requires separate optimizations [15]. For our work, we have developed a system for automatic tuning of the numerical integration algorithm [9].

C. Auto-tuning

Due to the different possibilities of using memory in accelerators based on graphics cards, eight parameters of the numerical integration algorithm affecting its final performance have been extracted:

1) WORKGROUP_SIZE - related to the minimum number of threads possible to launch on the accelerator - in the case of tests on the GPU, it was set to 64 due to the recommendation of the manufacturers of the tested cards.
2) USE_WORKSPACE_FOR_PDE_COEFF - a factor that determines whether problem data should be stored in threads' shared memory or registers (line 5 of the 1 algorithm).
3) USE_WORKSPACE_FOR_GEO_DATA - geometric data stored in threads' shared memory or registers (line 6 of algorithm 1).

4) USE_WORKSPACE_FOR_SHAPE_FUN - data of shape functions and their derivatives stored in the shared memory or registers (lines 3 and 10 of algorithm 1).
5) USE_WORKSPACE_FOR_STIFF_MAT - algorithm's local stiffness matrices and right-hand-side vectors stored in the threads' shared memory or registers (lines 7, 16, and 18 of the 1 algorithm).
6) COMPUTE_ALL_SHAPE_FUN_DER - algorithm computes the values of all shape functions and their derivatives before entering the double loop after the shape functions (line 10 of algorithm 1).
7) COAL_READ - memory readout in a continuous manner.
8) COAL_WRITE - continuously writing to memory.

The last two parameters of the numerical integration algorithm, concern the way of writing and reading data from the accelerator memory. Due to the physical design of this type of device and the operation of threads in groups, how memory is accessed can have a significant impact on the performance of the algorithm. Optimal access to memory should be grouped concerning threads, so that writes and reads are done in a continuous (coalesced) manner - whole vectors of data from memory. The list of parameters in the auto-tuning system is mostly concerned with the way memory is handled in the OpenCL model, which is based on the architecture of graphics cards. Arrays passed as arguments to the numerical integration procedure (e.g., geometric data and problem coefficients) can be used directly in calculations, or previously downloaded to local arrays, stored in registers or shared memory. By using shared memory as a temporary data read buffer, data is read continuously - a single thread reads the next memory cell and writes to the shared buffer. The read data can be stored in the buffer and used at a later time for calculation, or can be written into registers, freeing up the buffer for further use. How this data is stored in appropriate buffers is defined by the USE_WORKSPACE_FOR_* parameters. The total number of combinations of these parameters for any architecture is 40. Since there are 3 variants of the algorithm for each architecture (QSS, SQS, and SSQ), and we have 2 tasks, the total number of combinations per architecture is 240. The developed system consists of a set of scripts and code fragments responsible for compiling the kernel with the appropriate options. Due to the different compilation options available from various software vendors, it was necessary to implement different options depending on the installed accelerator and version of the OpenCL framework.

VI. Results

A. Poisson problem

During testing, it turned out that the QSS version of the numerical integration algorithm achieved the best results for both accelerators tested. Therefore, we have prepared the comparison graph (Fig. 4). The tuning options are arranged in the following order:

1) COAL_READ (CR)
2) COAL_WRITE (CW)
3) COMPUTE_ALL_SHAPE_FUN_DER (CASFD)
4) USE_WORKSPACE_FOR_PDE_COEFF (PDE)
5) USE_WORKSPACE_FOR_GEO_DATA (GEO)
6) USE_WORKSPACE_FOR_SHAPE_FUN (SHP)
7) USE_WORKSPACE_FOR_STIFF_MAT (STIFF)

In comparison with the results for GeForce RTX 3060 we can observe that the results for Intel Xe are more chaotic.

Fig. 4. Automatic tuning on tested GPUs in QSS version of the numerical integration algorithm for Poisson problem

On this graph, we see that the results for Nvidia GPU are very regular and connected with the use of only one option - CW. On Intel Xe, we can observe more chaotic results but with the best results in the same cases - with the CW option enabled. The use of shared memory in GeForce plays almost no role at all - it can be connected with the fact that in the Nvidia GPU all data are cached in L1 and L2 caches which results in no differences in the less resource-consuming cases. In the Iris Xe case, we can observe more memory dependent results - especially with the combination of the coalesced read/write and the use of the shared memory for SHP or GEO data. Although the OpenCL routine returns the presence of 64 kB of Shared Memory on Intel Xe (in opposition to 48 kB in Nvidia), it seems like this memory is mapped to the internal L3 cache memory which is slower than the classical SM used in external GPUs.

TABLE I
Results (in *ns*) of calculation for one element in Poisson problem

	QSS	*SQS*	*SSQ*
Intel Iris Xe-LP	9,185791	16,17432	49,23503
Nvidia GeForce 3060 RTX	0,992839	1,798503	8,390299

TABLE II
Options combination for the best times for Poisson problem

	QSS	*SQS*	*SSQ*
Intel Iris Xe-LP	0100010	1100001	0100000
Nvidia GeForce 3060 RTX	1101000	1110010	0100010

The best results obtained are shown in the table I. We can see that in the QSS and SQS cases GeForce is 9 times

faster than Intel Iris Xe. In SQS case GeForce is almost 6 times faster than Iris. Table II show the combination of options for the best results. As it was observed on the graphs earlier, all best results are connected with the use of coalesced writing of data. It indicates that both devices prefer vectorized organization of data in the memory what is characteristic for modern computing devices (both CPU and GPU).

B. Convection-diffusion-reaction problem

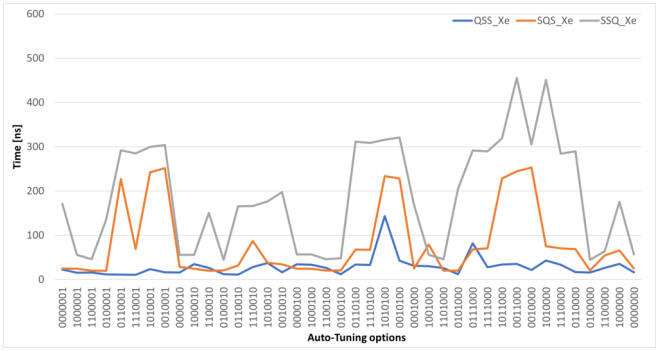

Fig. 5. Automatic tuning on Intel Iris Xe GPU for convection-diffusion problem

In the case of the more resource-consuming convection problem for Intel Iris Xe, we have observed more diversified results, and in some cases, the SQS version of the numerical integration algorithm was better than the QSS version (Fig. 5). The QSS version shows a more even level, only with a peak when the extensive use of shared memory is observed (GEO+CASFD+CR). In the Nvidia case, the results are similar to the Poisson case and are very regular in shape and the QSS version was the best in all the cases.

As in the Poisson problem, when we compare the results for the QSS algorithm for both GPUs, we can observe more chaotic results for the Intels' GPU (Fig. 6).

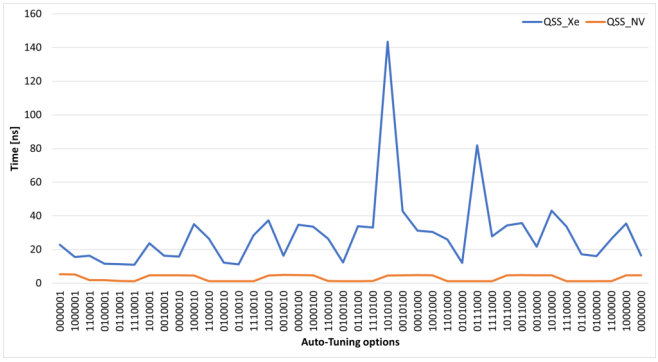

Fig. 6. Automatic tuning on tested GPUs in QSS version of the numerical integration algorithm for convection-diffusion problem

The best results presented in table III also show the same differences between the Nvidia and Intel GPUs, reaching out ten times faster execution on the GeForce RTX 3060. From the best options analysis (Tab. IV) we can still observe the great role of the coalesced writing in all variants of the algorithm. The other options show a more chaotic nature, but in the Nvidia case, we can observe that more resource-demanding types of an algorithm (SQS and SSQ) are making good use of the possibility of storing data in the shared memory.

TABLE III
RESULTS (IN *ns*) OF CALCULATION FOR ONE ELEMENT IN CONVECTION-DIFFUSION PROBLEM

	QSS	*SQS*	*SSQ*
Intel Iris Xe-LP	10,98633	19,90763	44,42342
Nvidia GeForce RTX 3060	1,196289	1,904297	9,090169

TABLE IV
OPTIONS COMBINATION FOR THE BEST TIMES FOR CONVECTION-DIFFUSION PROBLEM

	QSS	*SQS*	*SSQ*
Intel Iris Xe-LP	1110001	1100010	0100000
Nvidia GeForce 3060 RTX	1100000	0110010	0100100

VII. CONCLUSIONS

Intel Xe seems to have a more chaotic nature of execution and its parametric tuning graphs are more reminiscent of CPU graphs or the Xeon Phi we researched earlier [11], [47]. This may be due to its more complex design and different memory organisation than classic GPUs. Despite this, the results indicate a quite high potential of the discussed architecture and give hope that its versions intended for scientific and technical computing equipped with additional levels of memory [17] will be able to take an equal fight with the existing players that is Nvidia and AMD. In our future work, we are planning to test Intel Xe-LP architecture with the more time-consuming discontinuous Galerkin approximation for the Finite Element Method numerical integration and test the OpenCL 3.0 features connected with the use of the shared memory buffer between CPU and GPU cores in the integrated heterogeneous architectures. After this study, the authors hope that moving to the Intel Xe-HPC units will be easier to perform and provide the High-Performance Computing community with the answer if the new Intels' architecture can be competitive with existing solutions.

VIII. REFERENCES

[1] "NVIDIA Tesla C870 Professional Graphics Card," tech. rep., VideoCardz, 2015.
[2] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer, *Top500 The List*, 2020.
[3] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, and P. Dubey, "Larrabee: a many-core x86 architecture for visual computing," *SIGGRAPH 08: ACM SIGGRAPH 2008 papers*, pp. 1–15, 2008.
[4] R. Goodwins, "Intel unveils many-core Knights platform for HPC," *ZdNet*, 2010. Accessed on 27th November 2015.
[5] Intel, "Product change notification 116378 - 00," July 23, 2018.
[6] Intel, "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*, 2019.
[7] I. Cutress, "Intel's Xe for HPC: Ponte Vecchio with Chiplets, EMIB, and Foveros on 7nm, Coming 2021," *AnandTech*, 2019.

[8] R. Smith, "The Intel Xe-LP GPU Architecture Deep Dive: Building Up The Next Generation," *AnandTech*, 2020.

[9] K. Banaś, F. Krużel, and J. Bielański, "Optimal kernel design for finite element numerical integration on GPUs," *Computing in Science and Engineering*, vol. Volume 22, no. Issue 6, pp. 61–74, 2020.

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

[11] F. Krużel, "Vectorized implementation of the FEM numerical integration algorithm on a modern CPU," in *European Conference for Modelling and Simulation*, vol. Volume 33, pp. 414–420, 2019.

[12] F. Krużel and K. Banaś, "AMD APU systems as a platform for scientific computing," *Computer Methods in Materials Science*, vol. 15, no. 2, pp. 362–369, 2015.

[13] K. Banaś and F. Krużel, "Comparison of Xeon Phi and Kepler GPU performance for finite element numerical integration," in *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS), 2014 IEEE Intl Conf on*, pp. 145–148, Aug 2014.

[14] F. Krużel and K. Banaś, "Finite element numerical integration on Xeon Phi coprocessor," *Annals of Computer Science and Information Systems*, pp. 603–612, 10 2014.

[15] K. Banaś and F. Krużel, "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*, vol. 8806 of *Lecture Notes in Computer Science*, pp. 158–169, Springer International Publishing, 2014.

[16] Nvidia Corporation, *NVIDIA AMPERE GA102 GPU ARCHITECTURE: Ampere GA10x*, 2021. Whitepaper.

[17] Intel Corporation, *Intel Architecture Day 2020 Presentation Slides*, 2020. Whitepaper.

[18] Intel Corporation, *oneAPI GPU Optimization Guide*, 2022. Intel Developer Guide.

[19] M. Geveler, D. Ribbrock, D. Göddeke, P. Zajac, and S. Turek, "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*, vol. 80, pp. 327 – 332, 2013. Selected contributions of the 23rd International Conference on Parallel Fluid Dynamics ParCFD2011.

[20] L. Buatois, G. Caumon, and B. Levy, "Concurrent number cruncher: A GPU implementation of a general sparse linear solver," *Int. J. Parallel Emerg. Distrib. Syst.*, vol. 24, no. 3, pp. 205–223, 2009.

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

[22] P. Plaszewski, P. Maciol, and K. Banas, "Finite element numerical integration on GPUs.," in *PPAM'09: Proceedings of the 8th international conference on Parallel processing and applied mathematics*, (Berlin, Heidelberg), pp. 411–420, Springer-Verlag, 2010.

[23] P. Macioł, P. Płaszewski, and K. Banaś, "3D finite element numerical integration on GPUs," in *Proceedings of the International Conference on Computational Science, ICCS 2010, University of Amsterdam, The Netherlands, May 31 - June 2, 2010* (P. M. A. Sloot, G. D. van Albada, and J. Dongarra, eds.), vol. 1 of *Procedia Computer Science*, pp. 1093–1100, Elsevier, 2010.

[24] J. Filipovic, I. Peterlík, and J. Fousek, "GPU acceleration of equations assembly in finite elements method-preliminary results," in *SAAHPC: Symposium on Application Accelerators in HPC*, 2009.

[25] A. Dziekoński, P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Progress In Electromagnetics Research*, vol. 128, pp. 249–265, 2012.

[26] A. Dziekoński, P. Sypek, A. Lamecki, and M. Mrozowski, "Generation of large finite-element matrices on multiple graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 94, no. 2, pp. 204–220, 2013.

[27] C. Cecka, A. J. Lew, and E. Darve, "Application of assembly of finite element methods on graphics processors for real-time elastodynamics," in *GPU Computing Gems. Jade edition*, pp. 187–205, Morgan Kaufmann, 2011.

[28] C. Cecka, A. J. Lew, and E. Darve, "Assembly of finite element methods on graphics processors," *International Journal for Numerical Methods in Engineering*, vol. 85, no. 5, pp. 640–669, 2011.

[29] A. Logg, K.-A. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[30] G. R. Markall, D. A. Ham, and P. H. Kelly, "Towards generating optimised finite element solvers for GPUs from high-level specifications," *Procedia Computer Science*, vol. 1, no. 1, pp. 1815 – 1823, 2010. ICCS 2010.

[31] G. R. Markall, A. Slemmer, D. A. Ham, P. H. J. Kelly, C. D. Cantwell, and S. J. Sherwin, "Finite element assembly strategies on multi-core and many-core architectures," *International Journal for Numerical Methods in Fluids*, vol. 71, no. 1, pp. 80–97, 2013.

[32] L. Tang, X. Hu, D. Chen, M. Niemier, R. Barrett, S. Hammond, and G. Hsieh, "GPU acceleration of data assembly in finite element methods and its energy implications," in *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, pp. 321–328, June 2013.

[33] A. Karatarakis, P. Karakitsios, and M. Papadrakakis, "Gpu accelerated computation of the isogeometric analysis stiffness matrix," *Computer Methods in Applied Mechanics and Engineering*, vol. 269, pp. 334–355, 2014.

[34] F. Cabral, C. Osthoff, G. Costa, D. Brandao, M. Kischinhevsky, and S. Gonzaga de Oliveira, "Tuning Up TVD HOPMOC Method on Intel MIC Xeon Phi Architectures with Intel Parallel Studio Tools," *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2017 International Symposium on, SBAC-PADW*, pp. 19 – 24, 2017.

[35] I. W. C. Schneck, E. D. Gregory, and C. A. Leckey, "Optimization of elastodynamic finite integration technique on intel xeon phi knights landing processors.," *Journal of Computational Physics*, vol. 374, pp. 550 – 562, 2018.

[36] N. M. Atallah, C. Canuto, and G. Scovazzi, "The second-generation shifted boundary method and its numerical analysis.," *Computer Methods in Applied Mechanics and Engineering*, vol. 372, 2020.

[37] S. Muralikrishnan, M.-B. Tran, and T. Bui-Thanh, "An improved iterative hdg approach for partial differential equations.," *Journal of Computational Physics*, vol. 367, pp. 295 – 321, 2018.

[38] O. Zienkiewicz and R. Taylor, *Finite element method. Vol 1-3*. London: Butterworth Heinemann, 2000.

[39] C. Johnson, *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press, 1987.

[40] E. Becker, G. Carey, and J. Oden, *Finite Elements. An Introduction*. Englewood Cliffs: Prentice Hall, 1981.

[41] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, and A. Zdunek, *Computing with Hp-Adaptive Finite Elements, Vol. 2: Frontiers Three Dimensional Elliptic and Maxwell Problems with Applications*. Chapman & Hall/CRC, 2007.

[42] J. N. Lyness, "Quadrature methods based on complex function values," *Mathematics of Computation*, vol. 23, no. 107, pp. 601–619, 1969.

[43] Y. Kallinderis, "Adaptive hybrid prismatic-tetrahedral grids," *International Journal for Numerical Methods in Fluids*, vol. 20, pp. 1023–1037, 1995.

[44] K. Michalik, K. Banaś, P. Płaszewski, and P. Cybułka, "ModFEM – a computational framework for parallel adaptive finite element simulations," *Computer Methods in Materials Science*, vol. 13, no. 1, pp. 3–8, 2013.

[45] A. Howes, L.; Munshi, *The OpenCL Specification*. Khronos OpenCL Working Group, 2014. version 2.0, revision 26.

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

[47] K. Banaś, F. Krużel, and J. Bielański, "Finite element numerical integration for first order approximations on multi- and many-core architectures," *Computer Methods in Applied Mechanics and Engineering*, vol. 305, pp. 827 – 848, 2016.