Kazimierz Michalik
Łukasz Rauch

MESH COMPRESSION ALGORITHM FOR GEOMETRICAL COORDINATES IN COMPUTATIONAL MESHES

Abstract *Application of advanced mesh based methods, including adaptive finite element method, is impossible without theoretical elaboration and practical realization of a model for organization and functionality of computational mesh. One of the most basic mesh functionality is storing and providing geometrical coordinates for vertices and other mesh entities. New algorithm for this task based on on-the-fly recreation of coordinates was developed. Conducted tests are proving that, for selected cases, it can be orders of magnitude faster than naive approach or other similar algorithms.*

Keywords mesh compression, finite element method, high performance computing, computational efficiency

Citation Computer Science 24(4) 2023: 473–489

1. Motivation

The research and analysis described so far in many papers shows that operator of getting geometric coordinates for all vertices belonging to a certain finite element is very important to the overall performance of the mesh management scheme. Assuming that:

- $E \in \mathcal{E}$ is a single element belonging to set of all finite elements in computational mesh M
- $\mathcal{W}(\cdot)$ is an operator providing set of mesh vertices belonging to element $E \in \mathcal{E}$
- $\mathcal{G}_{eo}(\cdot)$ is an operator providing geometrical coordinates for given vertices

One can define operator of getting geometric coordinates for all vertices belonging to a certain finite element as:

$$\mathcal{G}_{eo}(\mathcal{W}(E)) \to \{x_1, y_1, z_1, ..., x_n, y_n, z_n\}, \quad E \in \mathcal{E} \wedge x_i, y_i, z_i \in \mathbb{R}, i \in \mathbb{N} \tag{1}$$

1. Although the operator itself is a rather simple and intuitive grid operator, the high frequency $f_{\mathcal{G}_{eo}(\mathcal{W}(E))}$ of its occurrence in each of the cases involving geometrical mesh simulations and low processing intensity AI of realization of this operator makes it of fundamental importance for performance. Conclusions resulting from the research the analysis of the literature incl. [4–6, 8, 9] allow us to formulate the following observations:
2. The most desirable situation is when, for a given M mesh, using the P representation, you can define linear orders for $(\mathcal{E}, \preceq)$ and $(\mathcal{W}, \preceq)$ vertices such as that:

   $$AI(\mathcal{G}_{eo}(\mathcal{W}(E))) \to \frac{|\mathcal{E}|}{|\mathcal{W}|} \tag{2}$$

   or at least

   $$AI(\mathcal{G}_{eo}(\mathcal{W}(E))) \to 1 \tag{3}$$

   in practice, this means that each point coordinate downloaded to the processor's *cache* memory from the main memory was used for calculations at least once, and preferably on average as many times as is the upper limit of the computational intensity AI for this mesh operator. At the same time, from the research mentioned earlier, this coefficient assumed the value of $AI(\mathcal{G}_{eo}(\mathcal{W}(E))) \to 0.1$.
3. There is no algorithm for renumbering the elements and nodes of the computational mesh for arbitrarily selected M mesh, which allows it to iterate over the nodes of the elements in a linear manner while maintaining the desired value of mentioned AI characteristics. Also considered here are *space-filling-curves* and their implementation in the form of Morton code [1, 3].
4. There are two types of vertices in computational meshes: edge and inner. It can be noticed that for the meshes with stationary nodes (considered in this paper), the outer (edge) nodes contain information about the area geometry, while the inner ones do not, because they only define elements in the area surrounded

by the border. The position of the former is important for the definition of the initial-boundary problem, but in practice it is obtained with a relatively low accuracy (10^{-3} with respect to the mesh dimensions), which results from the practical aspects of the way of mapping reality (technical drawing, photos, inaccuracy in the production of tested elements) and significant limitations related to the physics of modeled phenomena (thermal expansion, pressure influence, etc.). The position of internal nodes affects the quality of the finite elements, and thus the numerical accuracy (in particular the characteristic dimension h of the finite element), but otherwise it is arbitrary and can be changed as long as the quality of the elements is maintained (which is used e.g.: in r-adaptation) or to reduce the coefficient h influencing the error of the solution (h-adaptation and de-adaptation, r-adaptation).

5. Typically, the coordinates of points in irregular meshes are stored as floating-point numbers, and the single or double precision decision depends on how the degrees of freedom values are represented, so that arithmetic operations are performed on the same numeric representation without overloading the processor with type conversions. In fact, storing the position of points with an accuracy of 10^{-6} (single precision accuracy) or 10^{-15} (double precision accuracy) relative to the actual mesh size for a given geometry is redundant.
6. The accuracy of the geometry description is based on the assumptions of the applicability of the finite element method as the principle of physical continuity of the modeled medium. From a geometric point of view, the significant distinction of z points as low as one millionth in relation to the size of the studied medium goes beyond this assumption and if the phenomenon requires such accuracy, it also requires the use of multiscale modeling [7].
7. In an h-adaptive mesh, newly emerging vertices have coordinates that are completely dependent on the position of the vertex coordinates of the parent mesh objects, which in practice means that the coordinates of each resulting vertex can be expressed by some function whose arguments are parent vertices.

2. Mesh compression operator – formal description

Given is the M grid using the P representation for which are defined:

- Topological object identifier operator $Guid(T)$
- Vertex access operator $W(E)$
- Coordinate access operator $Geo(V)$

One can define a projection that assigns an arbitrarily selected number to the coordinates of each point

$$C(\cdot) : \mathbb{R}^3 \to \mathbb{N}^+ \tag{4}$$

what is used, among others in the definitions of space-filling curves. Then we can define an inverse mapping to $C(\cdot)$:

$$C_M^{-1} = D(\cdot)_M : \mathbb{N}^+ \to \mathbb{R}^3 \tag{5}$$

Since in the M mesh the set of points $\boldsymbol{\mathcal{W}}_M \subset \mathbb{R}^3$ is defined and finite ($n_{max} = |\{\mathcal{W}_M\}|$), we can write that for this mesh

$$\begin{aligned} C_M(\cdot) :&\boldsymbol{\mathcal{W}}_M \to \mathbb{N}_M \\ C^{-1} = D(\cdot) :&\mathbb{N}_M \to \boldsymbol{\mathcal{W}}_M, \\ \text{where} \qquad &\mathbb{N}_M \subset \mathbb{N}^+ \wedge |\mathbb{N}_M| = n_{max} \wedge \max \mathbb{N}_M = n_{max} \end{aligned} \tag{6}$$

Then we can assume that as long as C_M is the inverse relation to D_M, this relation is a bijection between two sets with the same size (cardinality):

$$\forall w \in \boldsymbol{\mathcal{W}}_M \quad Guid_M(w) = n \quad \Leftrightarrow \quad C_M(\mathcal{G}_{eo}(w)) = n \quad \Leftrightarrow \quad D_M(Guid_M(w)) = \mathcal{G}_{eo}(w) \tag{7}$$

In this way, we have obtained a relation that, for a specific grid, projects a discrete subset of a three-dimensional space into one-dimensional space and vice versa. Then it can be seen that the dependencies 6 will also be satisfied for any other n_{max} satisfying the condition

$$n_{max} \geq |\boldsymbol{\mathcal{W}}_M| \tag{8}$$

Note also that for a specific M mesh, the set of W_M geometric points is contained in a small subset of the three-dimensional space G_M which is the Cartesian product of the intervals containing the coordinates of all M mesh points.

$$\boldsymbol{G_M} \subset \mathbb{R}^3 : \forall w \in \boldsymbol{\mathcal{W}}_M \quad \mathcal{G}_{eo}(w) \in \boldsymbol{G_M} \tag{9}$$

Thus, it is possible to define a discrete subset of G_M^D of the G_M space, having at least n_{max} points, which include all the coordinates of the vertices contained in the M grid:

$$\begin{aligned} \boldsymbol{G_M^D} \subset \boldsymbol{G_M} \subset \mathbb{R}^3 : \quad &|\boldsymbol{G_M^D}| \geq |\boldsymbol{\mathcal{W}}_M| \\ &\wedge \forall w \in \boldsymbol{\mathcal{W}}_M \quad \mathcal{G}_{eo}(w) \in \boldsymbol{G_M^D} \end{aligned} \tag{10}$$

One can note, that in set G_M^D minimal point can defined as $\boldsymbol{G_M^D} \ni g_{min} = [min_x, min_y, min_z]$, as well as, maximal point as $\boldsymbol{G_M^D} \ni g_{max} = [max_x, max_y, max_z]$. Using set $\boldsymbol{G_M} \subset \mathbb{R}^3$, within it we can define another discrete subset $\boldsymbol{P_M}$ including $m \geq n_{max}$ points such that:

$$\begin{aligned} &\boldsymbol{P_M} \subset \boldsymbol{G_M^D} \subset \boldsymbol{G_M} \subset \mathbb{R}^3 : \\ &\qquad |\boldsymbol{P_M}| = m, \quad m \gg n_{max} \\ &\qquad \wedge \forall p \in P_M \\ &\qquad p = [x_p, y_p, z_p] = [min_x + d_x \cdot i_x, min_y + d_y \cdot i_y, min_z + d_z \cdot i_z] \\ \text{where} \quad &\qquad x_p, y_p, z_p, min_x, min_y, min_z, d_x, d_y, d_z \in \mathbb{R} \\ &\qquad i_x, i_y, i_z \in \mathbb{N} \end{aligned} \tag{11}$$

Note that m can be written as

$$m = m_x \cdot m_y \cdot m_z, \qquad m, m_x, m_y, m_z \in \mathbb{N}^+ \wedge m \geq m_x, m_y, m_z \tag{12}$$

And then the values of the natural multipliers i_x, i_y, i_z should be defined as

$$\begin{aligned} i_x &\in \{0, ..., m_x\} \\ i_y &\in \{0, ..., m_y\} \\ i_z &\in \{0, ..., m_z\} \end{aligned} \tag{13}$$

A previously unknown value d_x, d_y, d_z can be defined as:

$$\begin{aligned} &d_x, d_y, d_z \in \mathbb{R}^+ \\ &\quad d_x =(max_x - min_x)/m_x \\ &\quad d_y =(max_y - min_y)/m_y \\ &\quad d_z =(max_z - min_z)/m_z \end{aligned} \tag{14}$$

It is worth noting that for the limit values $i_0 = [0, 0, 0]$ and$i_m = [m_x, m_y, m_z]$, we obtain minimum and maximum elements for the set $\boldsymbol{P_M}$, which are also for the set $\boldsymbol{G_M^D}$:

$$\begin{aligned} [x_0, y_0, z_0] =&[min_x + d_x \cdot 0, min_y + d_y \cdot 0, min_z + d_z \cdot 0] = g_{min} \\ [x_m, y_m, z_m] =&[min_x + d_x \cdot m_x, min_y + d_y \cdot m_y, min_z + d_z \cdot m_z] = g_{max} \end{aligned} \tag{15}$$

Having two discrete sets $\boldsymbol{G_M^D}$ and $\boldsymbol{P_M}$ defined in this way, we can define a relationship that will assign to each element of the set $\boldsymbol{G_M^D}$ certain element of the set $\boldsymbol{P_M}$. Since $|\boldsymbol{G_M^D}| >> |\boldsymbol{P_M}|$ there are many points in the set $\boldsymbol{P_M}$ that can be assigned to one point in the set $\boldsymbol{G_M^D}$. So, let's consider two cases:

$$\begin{aligned} \exists ! [i_x^p, i_y^p, i_z^p] :&[g_x, g_y, g_z] = [min_x + d_x \cdot i_x^p, min_y + d_y \cdot i_y^p, min_z + d_z \cdot i_z^p] \\ \text{where} \quad &[g_x, g_y, g_z] \in \boldsymbol{G_M^D} \\ \text{where} \quad &i_x^p, i_y^p, i_z^p \in \mathbb{N} \end{aligned} \tag{16}$$

So let us note that using (6), (7) and (11) for the M grid defining the $\boldsymbol{G_M^D}$ coordinate space and the discrete subset $\boldsymbol{P_M}$ defined for it, we can define mapping:

$$\begin{aligned} &\forall w \in \boldsymbol{\mathcal{W}}_M \wedge \mathcal{G}_{eo}(w) \in \boldsymbol{P_M} \\ &\qquad Guid_M^P(w) = n_p \\ &\qquad\qquad n_p = i_x^p \cdot \frac{m}{m_x} + i_y^p \cdot \frac{m}{m_y} + i_z^p \cdot \frac{m}{m_z} \end{aligned} \tag{17}$$

for which n_p is a linear combination, depending for a constant mesh only on the i_x^p, i_y^p, i_z^p, because the expressions $\frac{m}{m_x}, \frac{m}{m_y}, \frac{m}{m_z}$ are invariant for each point in M.

Further the same as in (7):

$$\begin{aligned} &\forall w \in \boldsymbol{\mathcal{W}}_M \wedge \mathcal{G}_{eo}(w) \in \boldsymbol{P_M} \\ &C_M^P(\mathcal{G}_{eo}(w)) = n_p = i_x^p \cdot \frac{m}{m_x} + i_y^p \cdot \frac{m}{m_y} + i_z^p \cdot \frac{m}{m_z} \\ &\Leftrightarrow \\ &D_M(Guid_M^D(w)) = \mathcal{G}_{eo}(w) \\ &\Leftrightarrow \\ &D_M^P(i_x^p, i_y^p, i_z^p) = \mathcal{G}_{eo}(w) \end{aligned} \tag{18}$$

For a point from the set $\boldsymbol{G_M^D}$, there is no exact equivalent in the set $\boldsymbol{P_M}$, so we can calculate the distance from the point $g \in \boldsymbol{G_M^D}$ to the nearest point $p \in \boldsymbol{P_M}$. Note that from the definition of P_M (11) it follows that the maximum distance between two points in P_M is $\sqrt{d_x^2 + d_y^2 + d_z^2}$, so since we know that g does not correspond to any of the points, it means that it must be somewhere between them, so the maximum distance of the g point from the p point is $\sqrt{d_x^2 + d_y^2 + d_z^2}/2$. Using the same reasoning as in 16 - 18 we get:

$$\begin{aligned} &\forall w \in \boldsymbol{\mathcal{W}}_M \wedge \mathcal{G}_{eo}(w) \notin \boldsymbol{P_M} \\ &C_M^P(\mathcal{G}_{eo}(w)) = n_p = i_x^p \cdot \frac{m}{m_x} + i_y^p \cdot \frac{m}{m_y} + i_z^p \cdot \frac{m}{m_z} \\ &\Leftrightarrow \\ &D_M(Guid_M^D(w)) = \mathcal{G}_{eo}(w) + c_{err} \\ &\Leftrightarrow \\ &D_M^P(i_x^p, i_y^p, i_z^p) = \mathcal{G}_{eo}(w) + c_{err} \\ \text{where} \quad &c_{err} \le \frac{\sqrt{d_x^2 + d_y^2 + d_z^2}}{2} \\ &oraz \lim_{m \to \inf} c_{err} = 0 \end{aligned} \tag{19}$$

As a result, for the M mesh having the set of W_M points, we obtained the operator C_M^P assigning a natural number to each point of this mesh and the inverse D_M^P operator reproducing the coordinates of this point based on this number with an accuracy of at least c_{err}. Both of these operators depend on 12 parameters, 9 of which are common to all points in the mesh.

In particular, it can be assumed, for example, that - depending on the number of points in the computational grid - discrete points in the $\boldsymbol{P_M}$ space will be, for example: $m_{32} = 4,294967296 \cdot 10^9$ (i.e. as many as can be distinguished , using an integer in 32-bit notation) or $m_{64} = 18,44674407 \cdot 10^{18}$ (for 64 bit numbers, respectively). This gives a number (on average) of 10^3 or 10^6 points in each direction. Which means that regardless of the actual coordinates in space, we are able to map a point with an error of at most $c_{err}^{32} = \frac{10^{-3}}{2}$ or $c_{err}^{64} = \frac{10^{-6}}{2}$ relative to the span of the computational mesh in a given direction.

3. Mesh compression – algorithm

In practical application for the M mesh using C_M^P, D_M^P described formally in the previous point, the following operators were implemented:

$$\begin{aligned} \mathcal{G}_{eo}(t), &\qquad t \in \{\mathcal{W}\} \cup \{\mathcal{K}\} \cup \{\mathcal{S}\} \cup \{\mathcal{E}\} \\ Guid(t), &\qquad t \in \{\mathcal{W}\} \cup \{\mathcal{K}\} \cup \{\mathcal{S}\} \cup \{\mathcal{E}\} \end{aligned} \tag{20}$$

For which actions it is necessary to implement the following algorithms:

- Create database for compressed mesh representation M (algorithm 1). It corresponds to the definition of the boundary space of the discrete space of real points G_M^D and the calculation of all the coefficients necessary to perform the compression of the coordinates.
- Compression of geometric coordinates (algorithm 2). Responsible for the implementation of the operator $Guid(t)$ of the M grid using the operator C_M^P (formulas (18), (19)).
- Geometric coordinate decompression (algorithm 3). Contrary to the above, it is responsible for the implementation of the $\mathcal{G}_{eo}(t)$ operator of the M grid using the D_M^P operator.

Algorithm 1: Computing of compressed mesh representation coefficients

```
Data: $N$ - number of points in the mesh,
$W[N][3]$ – the array with the coordinates of the points
$m$ - number of points in $P_M$
Result: $M_{encoded}[3][3]$ – an array with the parameters of the encoded mesh
$min = max = W[1]$ // Determining the minimum and maximum point in $P_M$ ;
for $n \leq N$ do
    for $i \leq 3$ do
        if $W[n][i] \leq min[i]$ then
            $min[i] = W[n][i]$
        end
        else if $W[n][i] \geq max[i]$ then
            $max[i] = W[n][i]$
        end
        ++i ;
    end
    ++n ;
end
$span = [0,0,0]$ // Determination of base coefficients of the span of $P_M$ ;
$n_{span} = 0$ ;
$mm = [0,0,0]$ ;
$d = [0,0,0]$ ;
for $i \leq 3$ do
    $span[i] = max[i] - min[i]$ ;
    $n_{span} = n_{span} + span[i]$ ;
end
for $i \leq 3$ do
    $span[i] = span[i]/minimum(span)$ // Normalizing coefficients ;
    $mm[i] = \frac{m}{n_{span}} span[i]$ // Determination of span factors ;
    $d[i] = span[i]/mm[i]$ // Determining basis vectors ;
    ++i ;
end
$M_{encoded}[1] = min$ ;
$M_{encoded}[2] = d$ ;
$M_{encoded}[3] = mm$ ;
```

Algorithm 2: Mesh coordinates compression algorithm.

```
Data: $N$ - number of points in the mesh,
$W[N][3]$ - the array with the coordinates of the points
$M_{encoded}[3][3]$ - an array with the parameters of the encoded mesh
Result: $IDS[N]$ - an array with parameters of the encoded mesh
$min = M_{encoded}[1]$ ;
$d = M_{encoded}[2]$ ;
$mm = M_{encoded}[3]$ ;
for $n \leq N$ do
    $IDS[n] = 0$ ;
    for $i \leq 3$ do
        $IDS[n] = IDS[n] + (W[n][i] - min[i])/d[i] * mm[i]$ ;
        ++i ;
    end
    ++n ;
end
Note: in practice, multiplication by the $mm$ factor or its inversed value
  requires a few more treatments related to the representation of numbers on
  the selected processor, which are omitted here as a technical detail.
```

Algorithm 3: Mesh coordinates decompression algorithm.

```
Data: $N$ - number of points in the mesh,
$IDS[N]$ - the array with ids of the points
$M_{encoded}[3][3]$ - an array with the parameters of the encoded mesh
Result: $W[N][3]$ - an array with parameters of the encoded mesh
$min = M_{encoded}[1]$ ;
$d = M_{encoded}[2]$ ;
$mm = M_{encoded}[3]$ ;
for $n \leq N$ do
    $W[n] = [0, 0, 0]$ ;
    for $i \leq 3$ do
        $W[n][i] = min[i] + d[i] * (IDS[n]/mm[i])$ ;
        ++i ;
    end
    ++n ;
end
Note: in practice, multiplication by the $mm$ factor or its inversed value
  requires a few more treatments related to the representation of numbers on
  the selected processor, which are omitted here as a technical detail.
```

4. Theoretical performance analysis

Consider the case in which we want to move from standard notation of points to the discussed compressed notation. According to the proper mesh model both memory and compute performance should be considered.

4.1. Memory requirements

In the classical representation, the vertex coordinates are usually stored as 3 double-precision variables of $8B$ each, so for n_{vt}, $n_{vt} \cdot 24B$ memory is needed. At the same time, assuming the simplest assumption that the mesh is tetrahedral, for each of n_{el} we must store $n_{el}vt = 4$ vertex identifiers, each occupying $4B$ (assuming that the identifier is a 32-bit number). Thus, moving from the classic explicit storage of point coordinates to a compressed discrete representation, we change the memory requirement for storing the mesh as shown in the table 1 and in the figure 1 for 32 and 64 bit identifiers respectively.

Table 1
Comparison of the memory requirement against the number of n_{el} elements for an uncompressed and compressed mesh. Where n_{vt} – number of vertices in the mesh, $n_{el}vt$ – number of vertices in a single mesh element.

Mesh	Uncompressed	Compressed
int32	$(n_{vt} \cdot 24B) + (n_{el} \cdot n_{el}vt \cdot 4B)$	$48B + (n_{el} \cdot n_{el}vt \cdot 4B)$
int64	$(n_{vt} \cdot 24B) + (n_{el} \cdot n_{el}vt \cdot 8B)$	$48B + (n_{el} \cdot n_{el}vt \cdot 8B)$

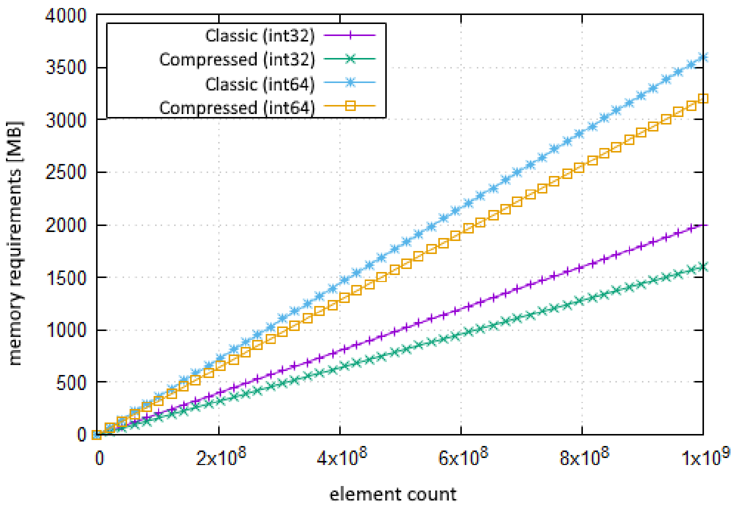


Figure 1. Comparison of memory requirements for uncompressed and compressed mesh.

4.2. Computational complexity

When analyzing the algorithms presented in 3, you can notice that the algorithms 1, 2 and 3 will be performed for each set of coordinates. So assuming the classical notation of computational complexity $O(\cdot)$:

1. The computational complexity for the algorithm 1 for n points in the mesh requires an average of $3n$ comparisons to be made once and additionally about 15 arithmetic operations, which results in a linear $O(n)$ complexity. It follows that the computing intensity $AI \to 1$.
2. For the algorithm 2 , note that we can omit the initial assignments as they are only for spelling clarity. In the main loop, run for each of n points, the operation is performed three times, but it contains the dependency carried in the loop of the Read-After-Write (RAW) and Write-After-Write (WAW) classes due to $IDS[n]$, so unfortunately there is no optimization here. Ultimately, this algorithm also has a complexity of $O(n)$ for n points. At the same time, it is worth noting that the computational intensity in this case is also $AI \to 1$.
3. The 3 algorithm is analogous to the algorithm 2 , so its computational complexity is $O(n)$ and its computational intensity is $AI \to 1$.

Note that using the above compression method, we have defined a linear order that satisfies the $AI \to 1$ processing intensity condition.

5. Performance tests

Figures 2, 3 and 4 show graphs showing how the access time to vertex data changes depending on their number, threads and access method. A simple operation of calculating the sum of the coordinates of all element vertices was selected as a test case. Two extreme cases of vertex index organization were chosen as the access method:

1. according to linear order, which is the optimal case for an uncompressed representation as it provides the best use of the data. In practical $3D$ meshes, this case never occurs,
2. in random order, which represents the pessimistic case of a real $3D$ computational mesh. In practical adaptive meshes, depending on the numbering methods adopted, there is usually *some small* indexing locality.

The graphs presented in Figures 2 and 3 show very clearly that in practically every case the disproportion between the results for compression and its absence increases more than linearly in favor of the former. It can also be seen that only in the case of the linear organization of indexes, which is unreal, the results obtained with a normal notation are comparable with a compressed one. The compressed variant shows practically the same results for both extreme cases of vertex index organization, which indicates independence from indexing organization. For each pair (regular, compressed) processed by the same number of threads, the compressed representation is faster. When examining the acceleration of calculations shown in the figure 4, it can be seen that in specific cases the obtained acceleration due to the compressed representation is different. In particular, you can see the impact of access to the processor cache for smaller numbers of items for which vertices are retrieved.

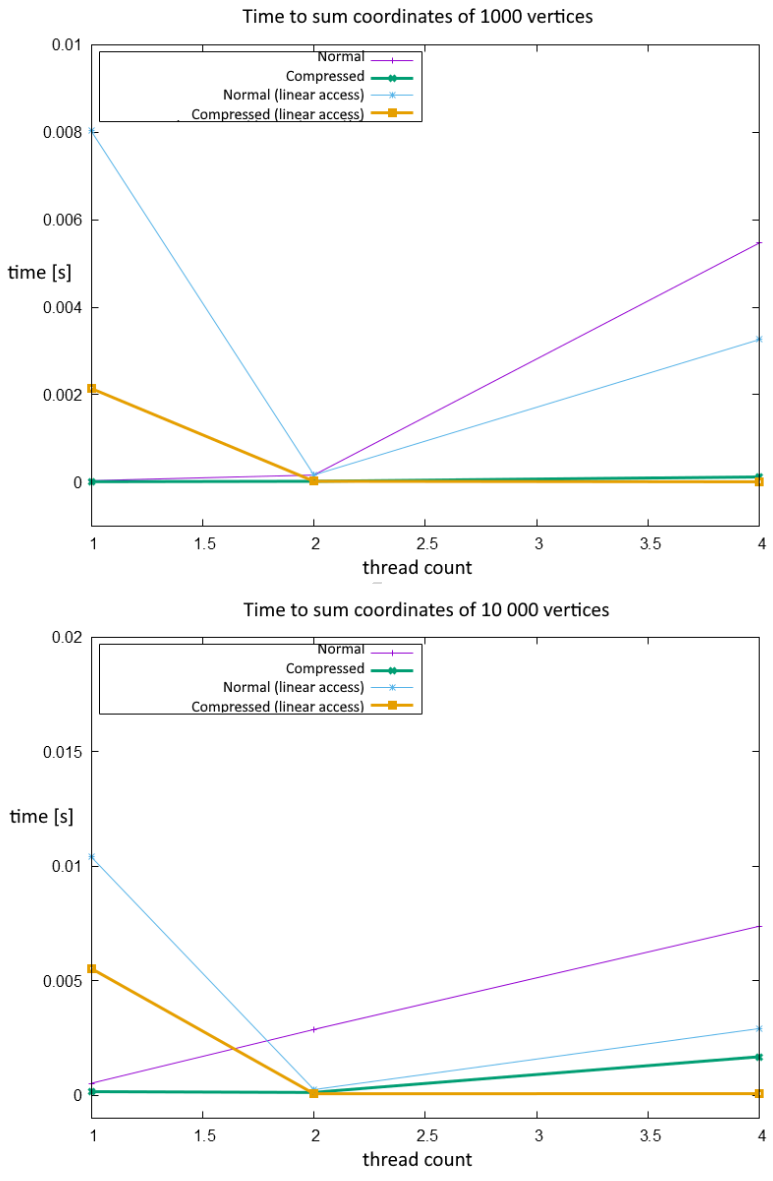
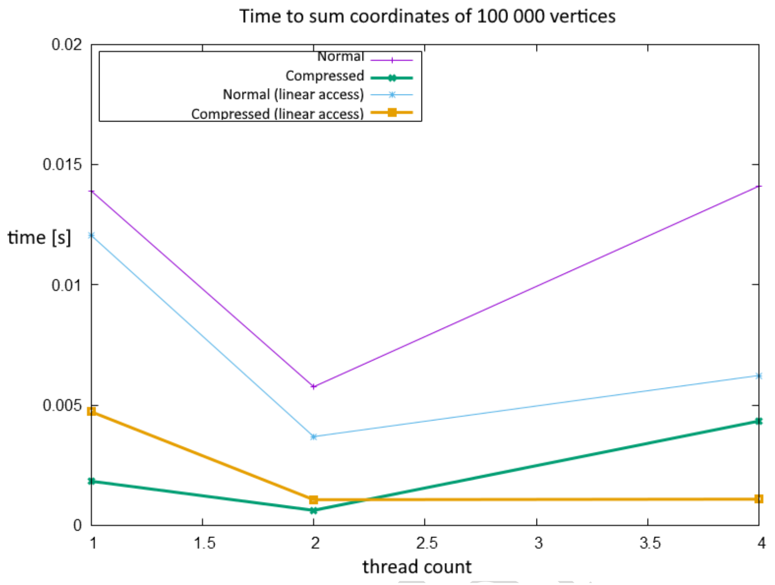


Figure 2. Times obtained in test case for 1k - 10k elements.

This algorithm allows very efficiently, both in the context of the memory used and the computation time, to obtain data on the coordinates of vertices, especially in



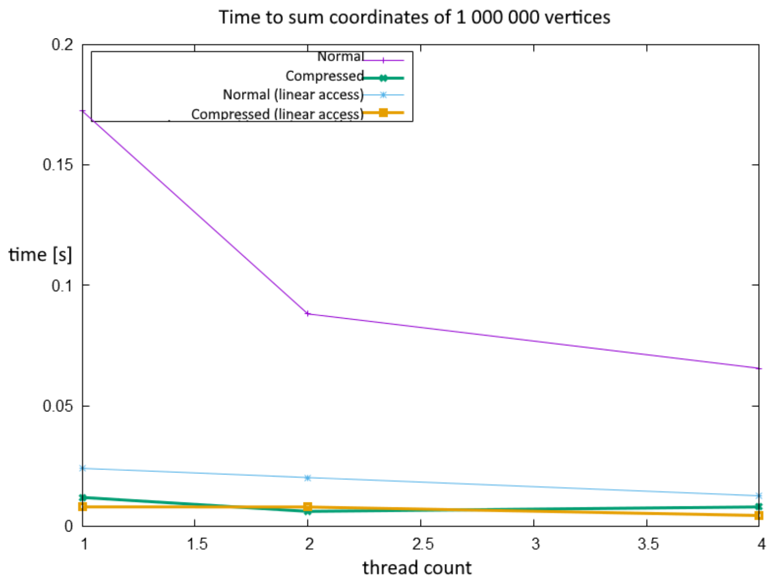


Figure 3. Times obtained in test case for 100k - 1M elements.

the case where the data would lie in distant memory areas. The following conclusions can be drawn from the analysis of the obtained results:

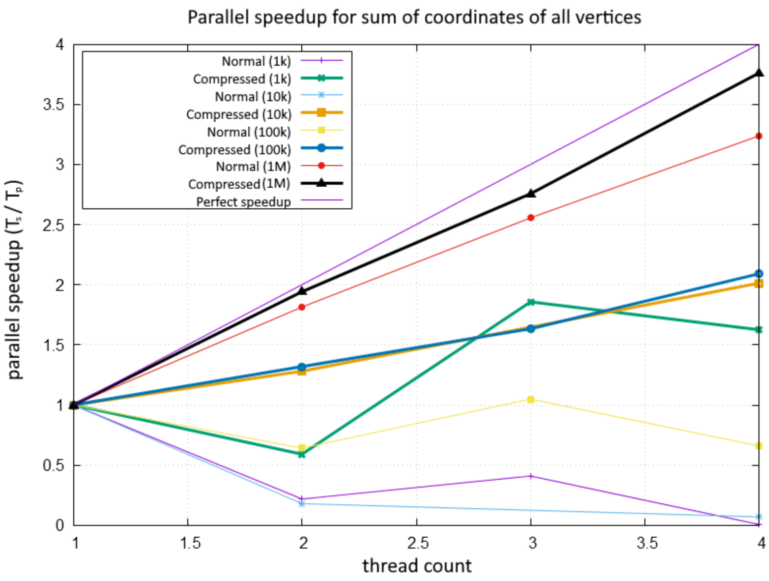


Figure 4. Computational speedup based on times obtained in test cases.

- Regardless of the selected case, the obtained results show that the compressed mesh is always faster in providing information about the coordinates of the vertices in the element than the standard storage of them in the memory (figure 2),
- as the number of nodes increases, the solution time decreases by more than ten times compared to the standard approach (figure 3),
- the applied solution is scalable, as shown in the picture 4.

5.1. Comparison with the space filling curves

The analysis of the literature mentioned at the beginning shows that the functionality simislar to the discussed solution is provided by the use of curves filling the space, in the form of *Z curves*, which are implemented using Morton's code. It is a well-known and widely used solution, also for assigning grid identifiers, e.g. in the computing package p4est [6] or in other applications. For comparison with the algorithm discussed in this paper, the highly efficient *libMorton* library used in practical applications was chosen [1]. It allows you to use both the standard encoding / decoding algorithm of Morton's code, as well as an implementation based, among others, on on logical tables, the so-called *lookup tables*.

While the idea and implementation of Morton's code was the subject of the above-mentioned In the publication, it should be noted that, formally, Morton's code is a linear mapping of $\mathbb{R}^3 \to \mathbb{N}$, but it requires prior writing of the real coordinates of the discussed subset in the form of integers. Most often this is done using the so-called normalize to the range $[0, 1]$ or $[0, MAX]$, where MAX is the largest integer that can be written in the selected representation. Normalizing is not *strictly* part of Morton's encoding, so it is not implemented in the *libMorton* library and *not* has been included in the comparison. It should also be emphasized that Morton's code from a numerical point of view converts two numbers into a number with twice the bit representation. Therefore, to keep the single precision of the point variables in $3D$, after normalizing it, use Morton's code $(x32, x32) \to (x64)$. The analysis of the obtained results, presented in the figure 5, clearly shows that the developed algorithm is more efficient than Morton's code in the process of encoding / decoding coordinates by an order of magnitude. You can also see that this difference is stable and holds, regardless of the size of the task. Also for Morton $(x16, x16) \to (x32)$ encoding the presented solution is more efficient. It should be emphasized that for Morton coding it is necessary to perform additional normalization each time, not included in the graphs.

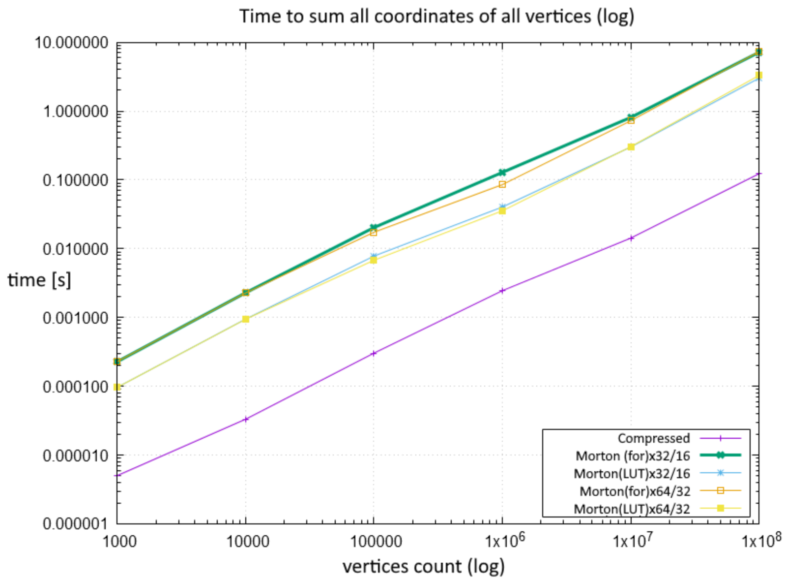


Figure 5. Performance comparison between presented algorithm and Morton code (logarithmic scale).

5.2. Comparison with available mesh management packages

During the analysis of the developed algorithm, a comparison was also made of the speed of the operator $Geo(W(Element))$ realized with the algorithms 2 and 3 in comparison with one of the most famous high computing packages performance. The Portable, Extensible Toolkit for Scientific Computation (PETSC) was chosen as a representative, having at least 767 use cases documented by publications in scientific and technical calculations according to the authors of the [2] package. As PETSC is modular software, the library directly responsible for meshing management, which is libMesh in PETSC, was used to maintain the objectivity of the comparison. The latest (2016) version of PETSC and libMesh was used. The aim of the test task was to compare the access time to the coordinates of the vertices of the elements, assuming a standard tetrahedral mesh with a cuboid geometry. The attached examples with the available documentation were adopted as the basis for the correct definition of the task. For reference, an example was also checked in which the vertices and their coordinates are stored in a regular array. The multiple attempts are summarized in the graph 6.

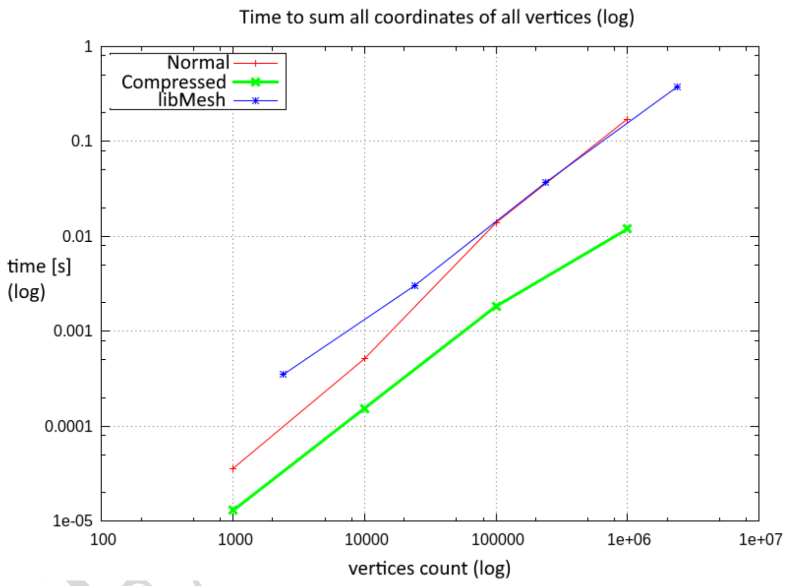


Figure 6. Performance comparison between presented algorithm and PETSC/libMesh (logarithmic scale).

The diagram 6 shows that for a representative test task the developed algorithm is about an order of magnitude faster than the libMesh implementation used in PETSC.

6. Disadvantages and limitations

The above method has the following disadvantages and limitations, partly due to the method itself, and partly characteristic of methods implementing similar functionality. For meshes using non-stationary vertices, the described method requires updating identifiers and compressed entry, which has a negative impact on the performance of the application. The compressed representation of the points has a (very slight, but still) approximation error. For large adaptive meshes, it is required to use the representation based on 64-bit numbers due to the geometric narrowing of the area of available points to choose from with the reduction of the linear size of the elements. For example, for an object calculation mesh with a length of $1[m]$ the linear size of the finite element would have to be smaller than $0.0006153[m] = 0.6[mm]$. For the 64-bit representation, the corresponding size would have to be smaller than $3 \cdot 10^{-7}[m]$, which is 2 orders of magnitude smaller than e.g. the expansion of iron at room temperature and an order smaller than the size of some grains in steel (the material can no longer be considered a continuous medium). In extreme cases, the method may result in assigning the same identifier to two different points.

7. Conclusion

Operator for providing geometrical coordinates for vertices based on on-the-fly recreation of coordinates was presented both formally and in practice. Necessary algorithms were provided for this task along with performance analysis. Presented tests shown that, for selected cases, new approach is around 10x faster then naive implementation and algorithm used in well know libMesh package. Also for given tests promising scalability results were obtained, showing increasing efficiency of algorithm for bigger meshes.

References

[1] Baert J., Lagae A., Dutré P.: Out-of-core Construction of Sparse Voxel Octrees. In: *Proceedings of the 5th High-Performance Graphics Conference*, pp. 27–32, HPG '13, ACM, New York, NY, USA, 2013. doi: 10.1145/2492045.2492048.

[2] Balay S., Adams M.F., Brown J., Brune P., Buschelman K., Eijkhout V., Gropp W.D., Kaushik D., Knepley M.G., McInnes L.C., Rupp K., Smith B.F., Zhang H.: PETSc Web page, http://www.mcs.anl.gov/petsc, 2014. http://www.mcs.anl.gov/petsc.

[3] Bangerth W., Burstedde C., Heister T., Kronbichler M.: Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes, *ACM Trans Math Softw*, vol. 38(2), pp. 14:1–14:28, 2012. doi: 10.1145/2049673.2049678.

[4] Berger M., Colella P.: Local adaptive mesh refinement for shock hydrodynamics, *Journal of Computational Physics*, vol. 82(1), pp. 64 – 84, 1989. doi: http://dx.doi.org/10.1016/0021-9991(89)90035-1.

[5] Berger M.J., Oliger J.: Adaptive mesh refinement for hyperbolic partial differential equations, *Journal of Computational Physics*, vol. 53(3), pp. 484 – 512, 1984. doi: http://dx.doi.org/10.1016/0021-9991(84)90073-1.

[6] Burstedde C., Wilcox L., Ghattas O.: p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees, *SIAM Journal on Scientific Computing*, vol. 33http://epubs.siam.org/doi/pdf/10.1137/100791634(3), pp. 1103–1133, 2011. doi: 10.1137/100791634. http://epubs.siam.org/doi/pdf/10.1137/100791634.

[7] Efendiev Y., Hou T.Y.: *Multiscale finite element methods : theory and applications*, Surveys and tutorials in the applied mathematical sciences, Springer, New York, NY, 2009. http://opac.inria.fr/record=b1127939.

[8] Garimella R.V.: Mesh Data Structure Selection for Mesh Generation and FEA Applications, *International Journal of Numerical Methods in Engineering*, vol. 55(4), pp. 451–478, 2002.

[9] Tautges T.J., Meyers R., Merkley K., Stimpson C., Ernst C.: *MOAB: A Mesh-Oriented Database*, SAND2004-1592, Sandia National Laboratories, 2004. Report.

Affiliations

Kazimierz Michalik
AGH University of Krakow, Faculty of Metals Engineering and Industrial Computer Science, Krakow, Poland, kazimierz.michalik@agh.edu.pl

Łukasz Rauch
AGH University of Krakow, Faculty of Metals Engineering and Industrial Computer Science, Krakow, Poland, lrauch@agh.edu.pl

Received: 26.12.2023
Revised: 04.01.2024
Accepted: 04.01.2024