Weierstraß-Institut für
Angewandte Analysis und Stochastik

Praktikumsbericht

WIAS FG3 Numerische Mathematik und Wissenschaftliches Rechnen

Dauer: 01.05.2019 – 30.09.2019
Betreuer: Dr. Wilbrandt, Ulrich
Verfasser: Simonov, Kirill

3. Dezember 2020

Inhaltsverzeichnis

1 Die Stellensuche

Im Laufe meines Studiums habe ich das Pflichtmodul „Wissenschaftliches Informationsmanagement“ belegt, welches mein Interesse zu der Arbeit in einer Bibliothek geweckt hat. In der darauffolgender vorlesungsfreien Zeit habe ich nach Stellenanzeigen in Bibliotheken in Berlin gesucht und letztendlich eine Stelle für eine studentische Hilfskraft in der Bibliothek des Weierstraß-Institutes für Angewandte Analysis und Stochastik (WIAS) in Berlin-Mitte gefunden. Dort habe ich mehrere Jahre lang gearbeitet und mich zwischendurch unter anderem auch an der Humboldt-Universität beraten lassen, ob ich mit einem NidI-Abschluss eine Aussicht auf eine Karriere im Bereich der Bibliotheks- und Informationswissenschaft habe. Letztendlich habe ich mich gegen diesen Gedanken entschieden, da es mein Studium um mindestens 1.5 Jahre verlängern würde, wenn ich alle nötigen Module belegen würde.
Im Laufe der Zeit habe ich mich mehr und mehr mit Pflicht- und Wahlpflichtmodulen aus den Bereichen Mathematik und Informatik beschäftigt. Da ich vor habe nach dem Bachelorabschluss einen Master in „Scientific Computing“ an der Technischen Universität zu machen, habe ich schon immer Ausschau nach einer Möglichkeit gehalten, das Berufspraktikum in einer der Forschungsgruppen am WIAS zu absolvieren, vor allem in der Forschungsgruppe 3 „Numerische Mathematik und Wissenschaftliches Rechnen“, welche thematisch meinem Interessenbereich entspricht. Nach einem Beratungsgespräch mit dem Leiter der FG3 Herrn Prof. John fand sich auch schnell ein Betreuer für mein Berufspraktikum, Herr Dr. Wilbrandt. Gemeinsam haben wir ein Thema ausgesucht und einige Wochen später konnte nach dem Unterschreiben eines Praktikumsvertrages mein 5-monatiges Praktikum losgehen.

2 Über das Institut

„Das WIAS wurde zum 1.1.1992 als Teil des Forschungsverbund Berlin e.V. gegründet. Es ist Mitglied der Leibniz-Gemeinschaft.“ [WIAa]
„Das Weierstraß-Institut betreibt projektorientierte Forschung in der Angewandten Mathematik mit dem Ziel, zur Lösung komplexer Probleme in Technik, Wissenschaft und Wirtschaft beizutragen. Es bearbeitet den gesamten wissenschaftlichen Lösungsprozess, beginnend mit einer mathematischen Modellierung über eine theoretische Analyse des Modells bis hin zur numerischen Simulation der Lösung.“ [WIAb]
Das Hauptgebäude vom WIAS befindet sich in der Mohrenstraße, jedoch sitzen einige Forschungsgruppen und auch die Bibliothek in zwei anderen Gebäuden am nahegelegenen Hausvogteiplatz.
Die Forschungsgruppe 3 „entwickelt, analysiert und implementiert moderne numerische Methoden für die Lösung von nichtlinearen Systemen partieller Differentialgleichungen und Algebro-Differentialgleichungen.“ [WIAc]

3 Ziel des Praktikums

Die Forschungsgruppe 3 entwickelt seit vielen Jahren das Software-Projekt ParMooN (Parallel Mathematics and object oriented Numerics). „ParMooN löst Gleichungen der Strömungsmechanik, wie Konvektions-Diffusions-Gleichungen, inkompressible Navier-Stokes-Gleichungen und gekoppelte Systeme dieser Gleichungen, auf Basis von Finite-Elemente-Diskretisierung und impliziten Zeitschrittverfahren.“ [WIAd]
Ich wusste schon im Voraus, dass ich mich an diesem Projekt beteiligen würde. Die Finite-Elemente-Methode wurde in Numerischer Mathematik leider nur sehr knapp behandelt, jedoch hatte ich mit dem mathematischen Teil der Software im meinem Praktikum letztendlich wenig zu tun.
Meine Aufgabe bestand darin, ein neues Datenformat zum Speichern und Visualisieren von Lösungen zu implementieren. Die bereits verwendeten Datenformate sollten um ein Format ergänzt werden, von welchem erwartet wurde, dass es das Datenvolumen von gespeicherten Lösungsdateien verringert. Das würde vor allem dann hilfreich sein, wenn entweder auf sehr feinen Gittern mit sehr vielen Elementen gerechnet wird, oder wenn mehrere Tausend Zeitschritte aufgenommen werden sollen, denn dabei kann die Dateigröße mehrere Gigabyte erreichen.
Beim Lösen von Gleichungssystemen werden zwei- oder dreidimensionale Arrays als Lösungen ausgegeben und gespeichert. Die Lösungen können anschließend mithilfe des Programms „ParaView “ veranschaulicht und ausgewertet werden. Deshalb wurden gleich am Anfang des Praktikums mehrere Teilaufgaben oder Meilensteine festgelegt, die eine sinnvolle Nutzung des zu implementierenden Datenformates ermöglichen sollten. Es sollte möglich sein zwei- und dreidimensionale zuerst stationäre und danach auch zeitabhängige Lösungen zu speichern, die zu speichernden Daten zu komprimieren und auch parallele Berechnungen durchzuführen.
Mein persönliches Ziel war im Laufe des Praktikums mir Kenntnisse der Programmiersprache C++ anzueignen und Erfahrungen in einem großen wissenschaftlichen Software-Projekt zu sammeln.

4 Praktikumsablauf

4.1 Die Einarbeitung

Am Anfang des Praktikums habe ich einen Arbeitsplatz in dem Computerraum des Institutes bekommen, wo ich die ersten Wochen gearbeitet habe. Danach habe ich einen Schlüssel von einem der Büroräume bekommen, welchen ich mit zwei anderen Mitarbeitern geteilt habe. Das hatte den Vorteil, dass mein Betreuer jetzt nur wenige Büroräume von mir entfernt war und jederzeit für Rückfragen erreichbar war.
In den ersten Tagen hat mein Betreuer für mich eine lokale Kopie des Projektes erstellt, an welcher ich arbeiten konnte ohne mir Sorgen darüber zu machen, dass ich durch meine

Änderungen die Arbeit von anderen Mitarbeitern negativ beeinflusse. Vor dem Praktikum habe ich zwar in einem Onlinekurs C++ gelernt, da ich davor noch nie mit dieser Sprache gearbeitet habe, aber dieses Grundlagenwissen hat sich als unzureichend erwiesen und ich musste viel Recherche betreiben und mich oft von meinem Betreuer beraten lassen, wenn ich Probleme hatte. Der Betreuer hat regelmäßig meinen Fortschritt angesehen und im Anschluss konnten wir immer besprechen, ob etwas verbessert werden muss und was ich als nächstes tun sollte.

4.2 Die Arbeit an dem Code

Ich habe hauptsächlich an der Datei *DataWriter.c* gearbeitet. Es handelt sich um eine C++-Sourcedatei, welche alle Funktionen beinhaltet, die zum Speichern von Lösungen nötig sind. Am Anfang war der Quellcode etwa 2.000 Zeilen lang und es waren bereits die Dateiformate VTK und VTU vollständig implementiert. Eine studentische Hilfskraft hat vor meinem Praktikum auch das Dateiformat CASE teilweise implementiert. Da ich mit dem Format XDMF arbeiten sollte, musste ich in den ersten Wochen viel über die Eigenschaften und Funktionen des Formates lesen. Sogar die Installation von XDMF gestaltete sich als schwierig.
Mein Betreuer hat mir den Editor KDevelop empfohlen, da es sich gut für größere Projekte eignete. So konnte man direkt in dem Editor jederzeit nach einer anderen Projektdatei suchen, sich die Beschreibungen bestimmter Funktionen durchlesen oder zum Beispiel alle Vorkommen einer Variablen anzeigen lassen. Unterschiedliche Variablentypen wurden auch farblich gekennzeichnet, was mir das Schreiben vom Code sehr erleichtert hat. Zwei Beispielabbildungen von dem Quellcode, welchen ich erstellt habe, sind im Anhang zu sehen.
Von Anfang an hatte ich fünf unterschiedliche Beispielfunktionen bzw. Probleme zum Testen von meiner Arbeit:

- CD2D = Convection-Diffusion-reaction equation 2D
- CD3D = Convection-Diffusion-reaction equation 3D
- NSE2D = Navier-Stokes Equation 2D
- NSE3D = Navier-Stokes Equation 3D
- TNSE2D = Time-dependent Navier-Stokes Equation 2D

Am Anfang habe ich mich nur mit stationären 2D-Problemen beschäftigt, und meine erste große Aufgabe war die Implementation des Codes, welcher beim Lösen der Testprobleme eine XDMF-Datei Zeile für Zeile schreiben sollte. Diese Textdatei beinhaltet alles, was für eine Visualisierung der Lösung benötigt wird, nämlich die Geometrie, Topologie und die Lösungspunkte. Dabei legen die Topologie das Gitter für die Darstellung der Lösung und die Geometrie die Position der Knotenpunkte des Gitters fest. Die Lösung bzw. die

Lösungspunkte, aber auch die Topologie und Geometrie werden dabei als eine Zahlenmatrix entweder als plain-text, in einer binären Datei oder in einem sogenannten Heavy Data Format (HDF5) .h5 gespeichert und in der XDMF-Datei als Verweis verlinkt.

Ein Beispiel für eine stationäre 3D-Lösung im XDMF-Format ist in der Abbildung 1. zu sehen.

```
<?xml version="1.0" ?>
<!-- # This file was generated by ParMooN -->
<Xdmf xmlns:xi="http://www.w3.org/2001/XInclude" Version="3.0">
  <Domain>
    <Grid>
      <Geometry GeometryType="XYZ">
        <DataItem Format="HDF" DataType="Float" Dimensions="64 3" Precision="8">
          parmoon_h5grid.h5:/Geometry
        </DataItem>
      </Geometry>
      <Topology TopologyType="Mixed" NumberOfElements="8">
        <DataItem Format="HDF" DataType="Int" Dimensions="72">
          parmoon_h5grid.h5:/Topology
        </DataItem>
      </Topology>

      <Attribute AttributeType="Scalar" Name="p">
        <DataItem Format="HDF" Dimensions="64" Precision="8">
          parmoon_solution.h5:/DS_Scalar_0
        </DataItem>
      </Attribute>
      <Attribute AttributeType="Vector" Name="u">
        <DataItem Format="HDF" Dimensions="64 3" Precision="8">
          parmoon_solution.h5:/DS_Vector_0
        </DataItem>
      </Attribute>
    </Grid>
  </Domain>
</Xdmf>
```

Abbildung 1: *Inhalt einer XDMF-Datei, welche die Lösung eines stationären 3D-Problems beinhaltet.*

Das zeilenweise Erstellen der XDMF-Dateien stellte sich letztendlich als eine einfache Aufgabe dar, auch wenn man immer aufpassen musste, dass man richtige Zahlenwerte und Parameter in die Datei schreibt. Das größere Problem stellte jedoch das Schreiben der Lösungen in binäre und in HDF5 Dateien. Es ließ sich in beiden Fällen nur unter großem Aufwand direkt kontrollieren, ob auch tatsächlich richtige Zahlen in der richtigen Reihenfolge gespeichert wurden. Meistens konnte man aber schon beim Visualisieren der Lösungen an einer Fehlermeldung oder gar einem Programmabsturz erkennen, dass etwas schief gelaufen ist.

Bei der Arbeit an dem Code war ich immer auf die Visualisierung angewiesen, da es mir erlaubt hat sofort nachzuschauen, wie sich die jeweilige Änderung ausgewirkt hat. In der Abbildung 2. sind jeweils zwei unterschiedliche Ansichten einer Lösung des NSE2D-Problems zu sehen, und in der Abbildung 3. ist die Visualisierung des NSE3D-Problems zu sehen.

Die nächste große Hürde stellte die Implementation des Speicherns von Lösungen von zeitabhängigen Problemen. Das war eines der Gründe, warum sich die Mitarbeiter der Gruppe für das XDMF-Format entschieden haben. Es sollte nämlich die Arbeit mit vielen Zeitschritten erleichtern, indem man die einzelnen Schritte direkt in einer XDMF-Datei speichert und nicht für jeden Schritt eine neue Datei erstellt. Bei größeren Problemen, bei welchen mehrere Zehntausend Zeitschritte gespeichert werden sollten, bedeutete dies

eine deutliche Verringerung des Speicheraufwandes und der Speicherplatzbedarfes. Ich habe aber beide Möglichkeiten beim Speichern ermöglicht, sodass die Mitarbeiter auch für Probleme mit einem kleineren Aufwand alle Zeitschritte in separaten Dateien speichern könnten.

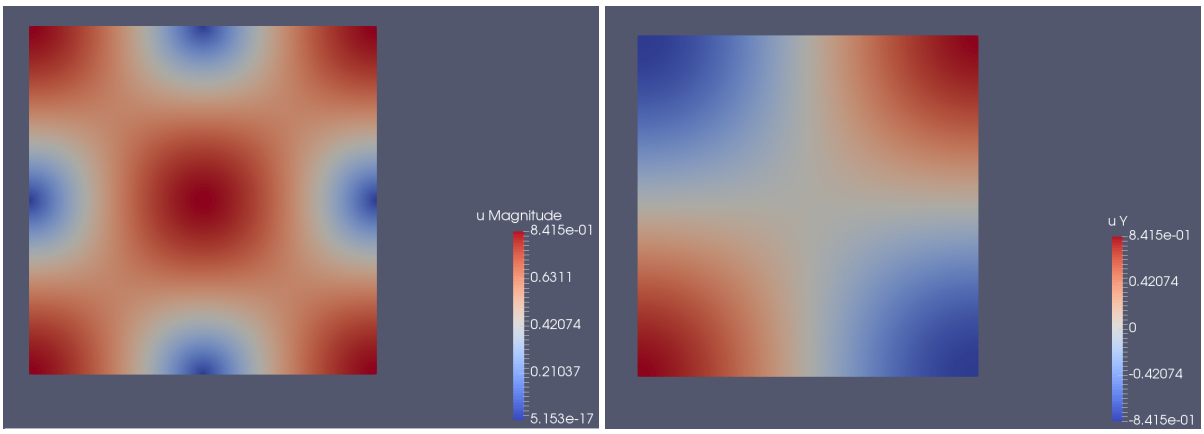

Abbildung 2: *Darstellung der Lösung der Navier-Stokes Gleichung NSE2D. Links: Ansicht der x-Komponente. Rechts: Ansicht der y-Komponente.*

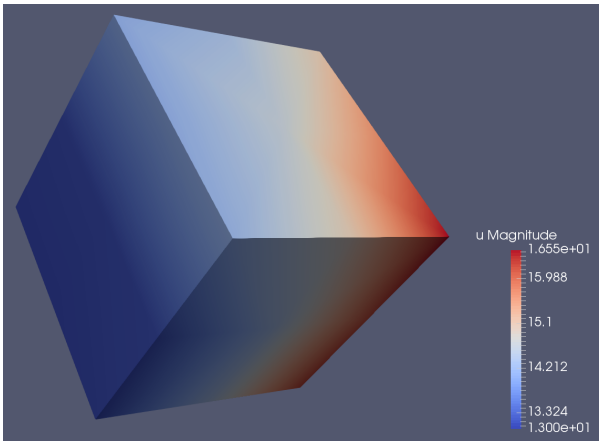

Abbildung 3: *Darstellung der Lösung der Navier-Stokes Gleichung NSE3D.*

Als Nächstes sollte ich die Möglichkeit einer Komprimierung der Lösungen implementieren. Dabei erwies sich XDMF wieder als eine gute Wahl, denn nur das Einfügen einer einzigen Zeile im Code erlaubte bereits eine Verwendung der Komprimierung:

H5Pset_deflate (prop, 9); // Enables zlib compression.

Die Zahl in der Klammer steht dabei für die Komprimierungsstufe, jedoch haben wir beim Testen wunderlicherweise keinen Unterschied zwischen den einzelnen Stufen entdeckt.

Das nächste große Ziel war die Implementierung des Speicherns bei einer Parallelrechnung. ParMooN erlaubt die Nutzung der Parallelrechner zum Lösen von Differentialgleichungen. Dabei wird das Lösen auf mehreren Prozessoren parallel durchgeführt und am

Ende zu einer Gesamtlösung zusammengefügt. Die Forschungsgruppe hatte bereits mit ParMooN Rechenzeiten auf einem Cray-Superrechner durchgeführt und die Lösungsdateien waren mehrere Gigabyte groß.

Zu diesem Punkt war meine Praktikumszeit aber fast vorbei, also entschied sich mein Betreuer dafür, diese Aufgabe wegzulassen. Am Ende meines Praktikums habe ich mich dann darum gekümmert, dass ich meinen geschriebenen Code „aufgeräumt“, also in Module verpackt und ausführlich und korrekt kommentiert habe. Außerdem habe ich in den letzten Tagen die Abbildungen für meinen Praktikumsbericht erstellt und eine Vergleichstabelle angefertigt, auf welche ich im nächsten Abschnitt näher eingehen werde.

5 Ergebnisse

Eine Idee meines Betreuers war es, am Ende von meinem Praktikum einen Vergleich zwischen unterschiedlichen im Projekt verwendeten Dateiformaten zu machen um herauszufinden, ob XDMF auch hält, was es verspricht. Die Formate VTK und VTU waren vor meinem Praktikumsbeginn bereits in Verwendung und von mir wurde das Format XDMF hinzugefügt. Für dieses Format gibt es drei Möglichkeiten des Speicherns von Lösungen: plain-text (XML), binäre Dateien und HDF-Dateien. Ich habe sechs unterschiedliche Probleme gelöst und in jedem zur Verfügung stehendem Format gespeichert. Anschließend habe ich die Größen von allen erstellten Dateien (in kB) zusammengefasst in die Tabelle eingetragen, damit man direkt vergleichen kann, welches Format den kleinsten Speicherplatzbedarf hat.

Funktion	VTK	VTU	XDMF		
			XML	Binary	HDF5
CD2D	3.7	25.6	8.6	8.2	16.2
CD3D	5.9	50.2	17.2	19.5	18.2
NSE2D	13.3	38.1	19.6	14.5	28.9
NSE3D	5.0	11.5	4.2	4.8	17.9
TNSE2D	529.5	1646.7	665.7	389.0	681.7
TNSE2D Fine	74974.6	243620.2	63541.5	53355.0	85807.8

Abbildung 4: *Vergleich von Dateigrößen (in kB) unterschiedlicher Speicherformate beim Lösen von Problemen.*

Erwartet war, dass XDMF unter Verwendung von HDF5 den geringsten Speicherplatz verbraucht, jedoch liegt das Format in fast allen Fällen zwischen VTK und VTU und die binären Dateien benötigen entgegen unserer Erwartung weniger Speicherplatz als die HDF5 Dateien. Selbst für ein zeitaufgelöstes Problem mit einem feineren Gitter und 1000 Schritten ist kein Vorteil von HDF5 zu erkennen gewesen. Um die Ursache von diesem

Verhalten herauszufinden fehlte uns leider die Zeit, daher nahm sich mein Betreuer vor, nach meinem Praktikumsende weiter sich mit HDF5 zu beschäftigen und rauszufinden, wie man dessen Vorteile besser ausnutzen könnte.

6 Fazit

Mein persönliches Ziel war es, die Programmiersprache C++ zu lernen und in diesen Monaten habe ich tatsächlich sehr viel gelernt, hauptsächlich durch praktische Arbeit an dem Code. Nebenbei habe ich auch sehr viel mit Linux gearbeitet und dabei auch viel Neues gelernt. Ich halte mein persönliches Ziel für erreicht und finde, dass mein Praktikum mir einen guten Einblick in die wissenschaftliche Software-Entwicklung und in die Arbeit in einem mathematischen Institut gegeben hat. Die Atmosphäre in der Forschungsgruppe war immer positiv und mein Betreuer hat mich immer unterstützt, daher habe ich nur positive Erfahrungen in dieser Zeit gesammelt und bin sehr dankbar für diese Möglichkeit.

Literatur

[WIAa] WIAS. URL: https://www.wias-berlin.de/about/facts.jsp?lang=0.

[WIAb] WIAS. URL: https://www.wias-berlin.de/about/mission.jsp?lang=0.

[WIAc] WIAS. URL: https://www.wias-berlin.de/research/rgs/fg3/index.jsp?lang=0.

[WIAd] WIAS. URL: https://www.wias-berlin.de/research/rgs/fg3/software.jsp?lang=0.

7 Abbildungen

```cpp
template <unsigned int filedim,typename T>
void write_h5_file(
  unsigned int d, unsigned int nrows, unsigned int dim, unsigned int maxdim,
  hid_t& space, hid_t& set, hid_t& prop, hid_t file,
  const char* ds_name, const T* data)
{
  // Write and expand the datasets in the h5 files, in which the heavy data
  // of the grid and the solutions is stored.
  //herr_t status; // status is negative, if the function call was unsuccessful.
  bool is_int = std::is_same<T,unsigned int>::value;

  // Create the h5 dataset with current and maximal possible dimensions of the
  // data to be stored. Then write the data into the file.
  if(nrows == 0)
  {
    hsize_t dims[filedim], maxdims[filedim];
    dims[0] = dim;
    maxdims[0] = maxdim;
    if(filedim == 2)
    {
      dims[1] = d;
      maxdims[1] = d;
    }

    prop = H5Pcreate (H5P_DATASET_CREATE);
    H5Pset_chunk (prop, filedim, dims);
    H5Pset_deflate (prop, 9);           // Enables zlib compression.
    space = H5Screate_simple (filedim, dims, maxdims);
    set = H5Dcreate2 (file, ds_name, (is_int ? H5T_STD_I64BE : H5T_IEEE_F64LE),
                      space, H5P_DEFAULT, prop, H5P_DEFAULT);
    H5Dwrite (set, (is_int ? H5T_NATIVE_INT : H5T_NATIVE_DOUBLE),
                       H5S_ALL, H5S_ALL, H5P_DEFAULT, data);
  }
  // If the dataset exists already, it needs to be extended by the dimensions
  // of the next block of data, which is then added to the dataset.
  else
  {
    hsize_t size[filedim], offset[filedim], dimsext[filedim];
    hid_t filespace, memspace;
    size[0] = dim + nrows;
    dimsext[0] = dim;
    offset[0] = nrows;
    if(filedim == 2)
    {
      size[1] = d;
      dimsext[1] = d;
      offset[1] = 0;
    }

    H5Dset_extent (set, size);
    filespace = H5Dget_space (set);
    H5Sselect_hyperslab (filespace, H5S_SELECT_SET, offset,
                         NULL, dimsext, NULL);
    memspace = H5Screate_simple (filedim, dimsext, NULL);
    H5Dwrite (set, (is_int ? H5T_NATIVE_INT : H5T_NATIVE_DOUBLE),
                       memspace, filespace, H5P_DEFAULT, data);
    H5Sclose (memspace);
    H5Sclose (filespace);
  }
}
```

Abbildung 5: *Die Funktion **write_h5_file**, welche eine HDF5-Datei erzeugt.*

```
template<int d>
void DataWriter<d>::write_vector_solution(
  hid_t& h5file, const std::string dataformat, const unsigned int n_cells,
  const unsigned int n_vertices, const bool isTemporal,
  const unsigned int n_timesteps, const std::string timestep_name) const
{
  unsigned int nrows = 0;
  //herr_t status; // status is negative, if the function call was unsuccessful.
  hid_t prop = 0;

  std::ofstream v;
  if(xdmf_format == xdmf_data_formats::binary)
  {
    v.open(testcaseDir + "/" + testcaseName + "_vector_solution"
        + timestep_name + ".bin", std::ios::binary);
  }

  for(unsigned int i = 0; i < FEVectFunctArray.size(); i++)
  {
    const auto* fe_vect = FEVectFunctArray[i];

  // Create a new dataset for every solution function.
    std::string vds_name = "DS_Vector_" + std::to_string(i);
    if(n_timesteps >= 1 && !separate_h5)
    {
      vds_name += timestep_name;
    }

  // Header for the vector solution attribute of the XDMF file.
    xdmf_buffer.push_back("      <Attribute AttributeType=\"Vector\" Name=\""
      + fe_vect->GetName() + "\">\n        <DataItem Format=\"" + dataformat
      + "\" Dimensions=\""+std::to_string(n_vertices)+" 3\" Precision=\"8\">");

    hid_t vspace = 0, vset = 0;
    nrows = 0;
    for(unsigned int icell = 0; icell < n_cells; icell++)
    {
      const TBaseCell* cell = Coll->GetCell(icell);
      unsigned int n_Loc_vert = cell->GetN_Vertices();
      const int max_n_Loc_vert = 8;
  // Vectors must have 3 elements even for 2D models
      double vdata[3*max_n_Loc_vert] = {0.0};

      for(unsigned int j = 0; j < n_Loc_vert; j++)
      {
  // Compute and store the vector solution array as an array for every vertex.
        double x, y, z;
        cell->GetVertex(j)->GetCoords(x, y, z);
  // Vectors must have 3 elements even for 2D models
#ifdef __2D__
        fe_vect->FindValueLocal(cell, icell, x, y, &vdata[j*3]);
#else
        fe_vect->FindValueLocal(cell, icell, x, y, z, &vdata[j*3]);
#endif
      }
```

Abbildung 6: *Der erste Teil der Funktion* ***write_vector_solution****, welche den vektoriellen Teil der Lösung erzeugt und speichert.*