BiggMann++

BiggMann ++

- C++ open source software for complete highly customizable physics simulations from mesh generation to post process.

Features :

- Curvilinear Mesh Generation
- Arbitrarily spaced Finite Difference Schemes (up to 6th order accuracy)
- High Performance Computing (multi-thread / CUDA matrix / vector operations, ~~sparse matrix operations~~ (to be announced)
- Linear Algebra tools (direct & iterative solvers, matrix eigendecomposition, statistical utility etc)

Motivation

- Initially a personal drive to learn more about the art of Computational Mechanics from the cryptic mathematical notations to memory adress pointers.

- Leisure time activity. I am 30 and I got bored of pc games.

- Fulfillment of my intrinsic need to express, explore and create. It is an (almost) cost free physics sandbox. Computational mechanics are the poor man's experiment.

- Serotonin boost from:
  - Watching my GPU filled with numerical matrices
  - Watching the CPU being 100% occupied as i commanded
  - RAM overflow due to bad coding or huge matrices
  - Convergence to correct solution and other computational small thrills.

- Serotonin Draaaaaaiiiiinage
  - SIGSEV
  - Deadlines

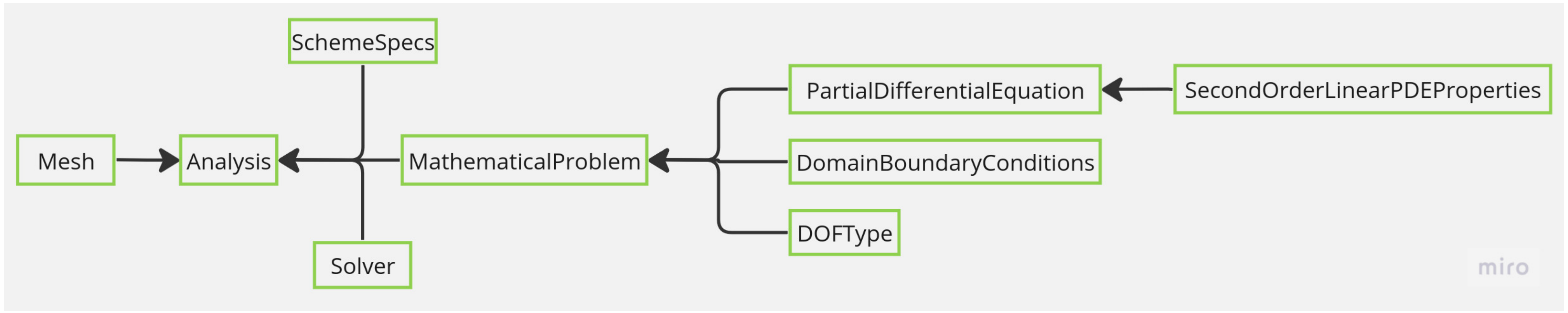
Software Design– Core Ideas

- Complete avoidance of external libraries (only std)
- Parametrize what is parametrizable with ease.
- Conceive and programm valid mathematical abstractions behind the physics/engineering cases
- Complete simulation solution the user from mesh generation to ParaView ready result data
- Clear and descriptive naming
- OOP principles to avoid code repetition and boost maintainability. Main target is to avoid ifs and extensive case consideration (FD schemes, matrix storage formats etc)
- Cache friendly data storage for frequently accessed objects and numerical data structures

Software Design– Core Ideas

- Avoidance of clustered “master classes”. Each class and function should perform only one task
- Wide usage of enums to describe Directions, Positions, CoordinateTypes etc. Great and intuitive dictionary keys!

- Consistent node identification combined with the iso-parametric curves of the mesh lead to a an intuitive and robust description of the nodal relations with *O(1)* complexity.



Software Design– Mathematical Application

Aim of the software is to solve the generalized second order PDE for a scalar or vector field.

$$\sum_{i,j=1}^{4} A_{ij} \frac{\partial^2 \phi}{\partial x_i \partial x_j} + \sum_{i=1}^{4} B_i \frac{\partial \phi}{\partial x_i} + C\phi = D$$

- It can can be applied to a domain with up to 4 dimensions (any combination of an orthonormal coordinate system (1-2-3) plus time.) Space and dimensions are used in an abstract way and can represent anything from physical space to frequencies (up to 3 +1)

Software Design– Mathematical Application

- With some proper manipulation almost all other transport / conservation laws can be abstractly viewed as child classes of the generalized second order equation.

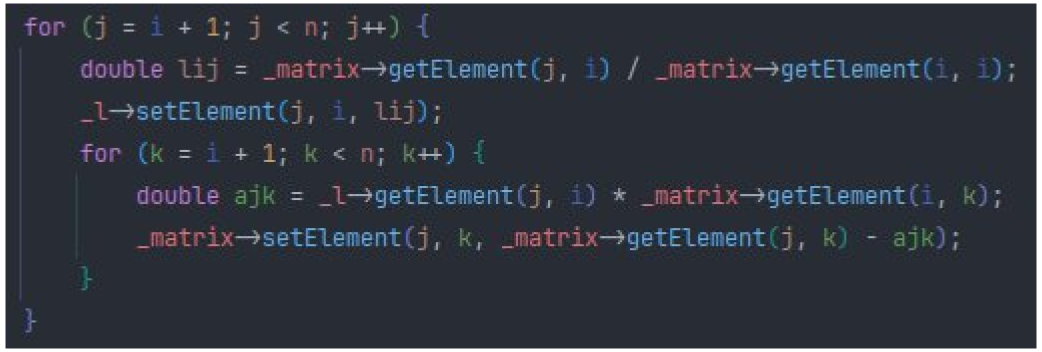
- The main idea behind the design is to be as physics agnostic as possible. The main mathematical essence of the general equation is inherited to each child class that represents a transport equation and more specifications are added if needed.

- Various DOF specifications for scalar and vector fields(Temperature, velocity, position, UnknownScalarVariable, UnknownVectorFieldVariableComponentI etc)

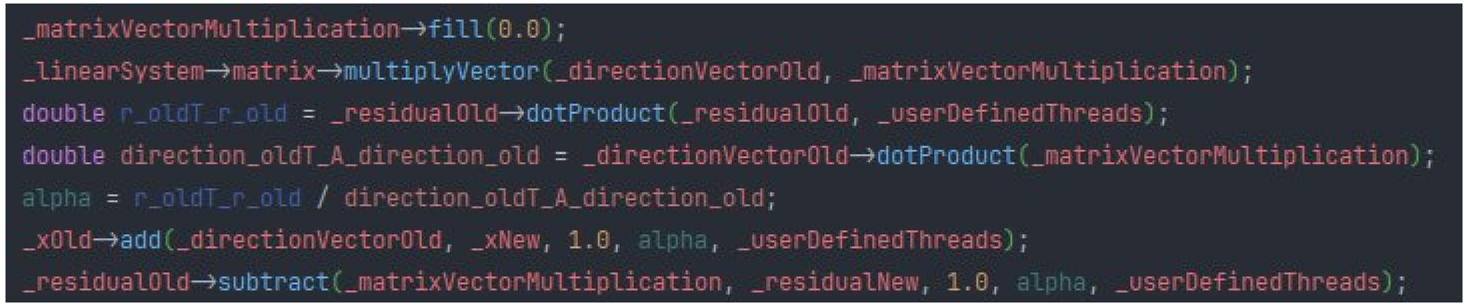
Software Design– Data access

MATLAB / Scipy (wannabe) user experience through carefully designed containers of std::vector.

- **Easily access and operate** on matrices and vectors in element or in full data structure level.
- Set the available threads for each operation.



```
for (j = i + 1; j < n; j++) {
    double lij = _matrix->getElement(j, i) / _matrix->getElement(i, i);
    _l->setElement(j, i, lij);
    for (k = i + 1; k < n; k++) {
        double ajk = _l->getElement(j, i) * _matrix->getElement(i, k);
        _matrix->setElement(j, k, _matrix->getElement(j, k) - ajk);
    }
}
```



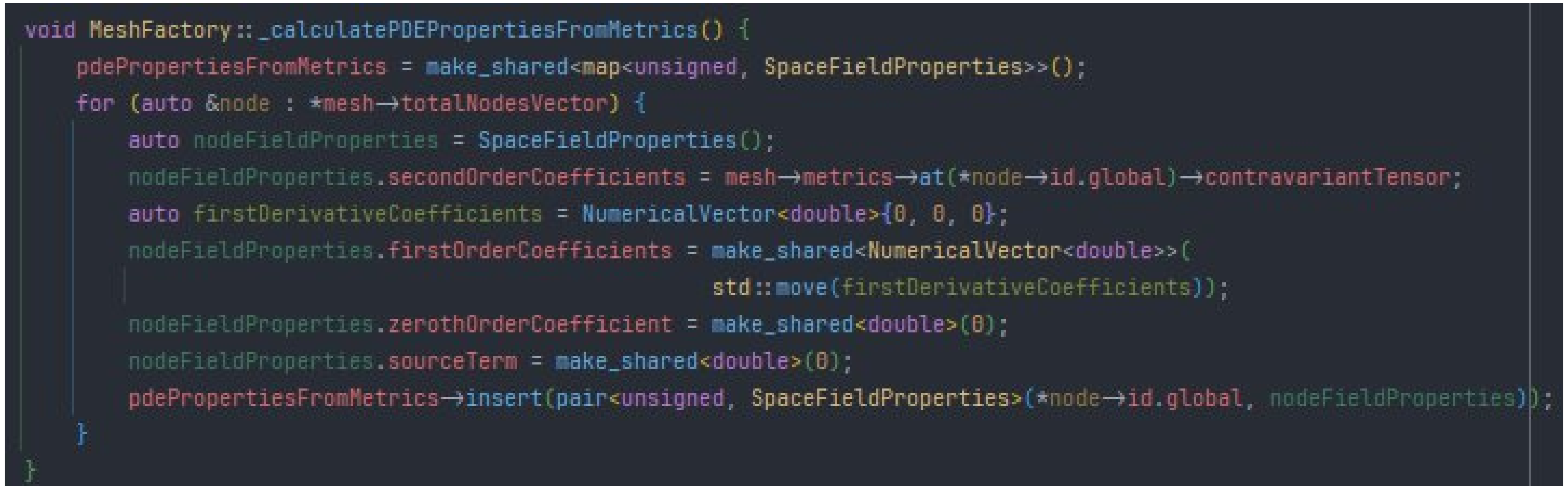
```
_matrixVectorMultiplication->fill(0.0);
_linearSystem->matrix->multiplyVector(_directionVectorOld, _matrixVectorMultiplication);
double r_oldT_r_old = _residualOld->dotProduct(_residualOld, _userDefinedThreads);
double direction_oldT_A_direction_old = _directionVectorOld->dotProduct(_matrixVectorMultiplication);
alpha = r_oldT_r_old / direction_oldT_A_direction_old;
_xOld->add(_directionVectorOld, _xNew, 1.0, alpha, _userDefinedThreads);
_residualOld->subtract(_matrixVectorMultiplication, _residualNew, 1.0, alpha, _userDefinedThreads);
```

Snippets from LUP and Conjugate Gradient Solvers

Software Design– Parametrization of Model Parameters

- Properties can be uniform for all nodes or vary with each node

- Constants ,where possible, are expressed in matrix/vector/scalar forms (isotropic-anisotropic, non-homogenous etc. host medium properties)

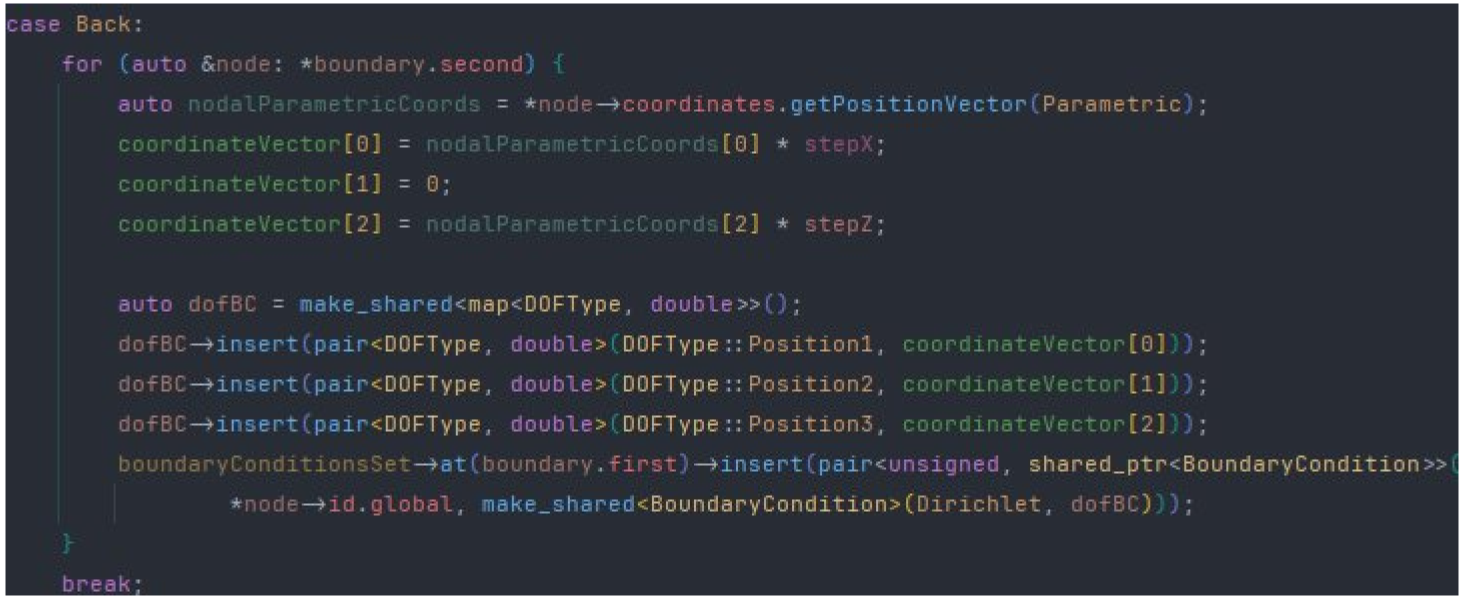


```cpp
void MeshFactory::_calculatePDEPropertiesFromMetrics() {
    pdePropertiesFromMetrics = make_shared<map<unsigned, SpaceFieldProperties>>();
    for (auto &node : *mesh->totalNodesVector) {
        auto nodeFieldProperties = SpaceFieldProperties();
        nodeFieldProperties.secondOrderCoefficients = mesh->metrics->at(*node->id.global)->contravariantTensor;
        auto firstDerivativeCoefficients = NumericalVector<double>{0, 0, 0};
        nodeFieldProperties.firstOrderCoefficients = make_shared<NumericalVector<double>>(
                                                     std::move(firstDerivativeCoefficients));
        nodeFieldProperties.zerothOrderCoefficient = make_shared<double>(0);
        nodeFieldProperties.sourceTerm = make_shared<double>(0);
        pdePropertiesFromMetrics->insert(pair<unsigned, SpaceFieldProperties>(*node->id.global, nodeFieldProperties));
    }
}
```

Snippet from non-symmetric and anisotropic properties assignement during mesh generation

Software Design– Parametrization of Boundary Conditions

- Dirichlet and Neumann
- Boundary Conditions can be uniform for all nodes or vary with each node
- Boundary values can be expressed as a scalar or a lambda of the nodal coordinates
- Interpolation between two values of the same BC type at edge nodes



```
case Back:
    for (auto &node: *boundary.second) {
        auto nodalParametricCoords = *node→coordinates.getPositionVector(Parametric);
        coordinateVector[0] = nodalParametricCoords[0] * stepX;
        coordinateVector[1] = 0;
        coordinateVector[2] = nodalParametricCoords[2] * stepZ;

        auto dofBC = make_shared<map<DOFType, double>>();
        dofBC→insert(pair<DOFType, double>(DOFType::Position1, coordinateVector[0]));
        dofBC→insert(pair<DOFType, double>(DOFType::Position2, coordinateVector[1]));
        dofBC→insert(pair<DOFType, double>(DOFType::Position3, coordinateVector[2]));
        boundaryConditionsSet→at(boundary.first)→insert(pair<unsigned, shared_ptr<BoundaryCondition>>(
                *node→id.global, make_shared<BoundaryCondition>(Dirichlet, dofBC)));
    }
    break;
```

Snippet from the boundary conditions setting for the Back boundary of a parallelepiped

Curvilinear Mesh Generation

The equation solved for the generation of a curvilinear mesh is expressed as follows :

$$g^{ij}\frac{\partial^2 x_r}{\partial x^i \partial \xi^j} - g^{ij}\Gamma^{ij}\frac{\partial x_r}{\partial \xi^k} = 0$$

$$\Gamma^{ij} = \frac{1}{2}g^{lk}\left(\frac{\partial g_{il}}{\partial \xi^j} + \frac{\partial g_{jl}}{\partial \xi^i} + \frac{\partial g_{ij}}{\partial \xi^l}\right)$$

After some routine mathematical manipulations:

$$g^{ij}\frac{\partial^2 x_m}{\partial \xi^i \partial \xi^j} - \frac{\partial x_m}{\partial \xi^j} f^j = 0$$

g_{ij} is the covariant tensor and g^{ij} is the contravariant tensor that quantify how the natural coordinate system changes over the parametric and vice versa.

$$g^{ij} = g_i \cdot g_j \text{ where } g_i = \frac{\partial \boldsymbol{r}}{\boldsymbol{\xi_i}} \text{ and } g^{ij} = \boldsymbol{\nabla}\xi_i$$

Linear Algebra – Differentiation at arbitrarily spaced curvilinear grids

Numerical Differentiation with accuracy up to 6th order

- Get number of points needed for input order accuracy (Hard-coded). For example:

  Derivative Order 1, Error Order 2 :

  points(+)->2

  points(-)->2

  points(+-)->1

- For each node get the neighbor graph and check if it has sufficient number nodes .

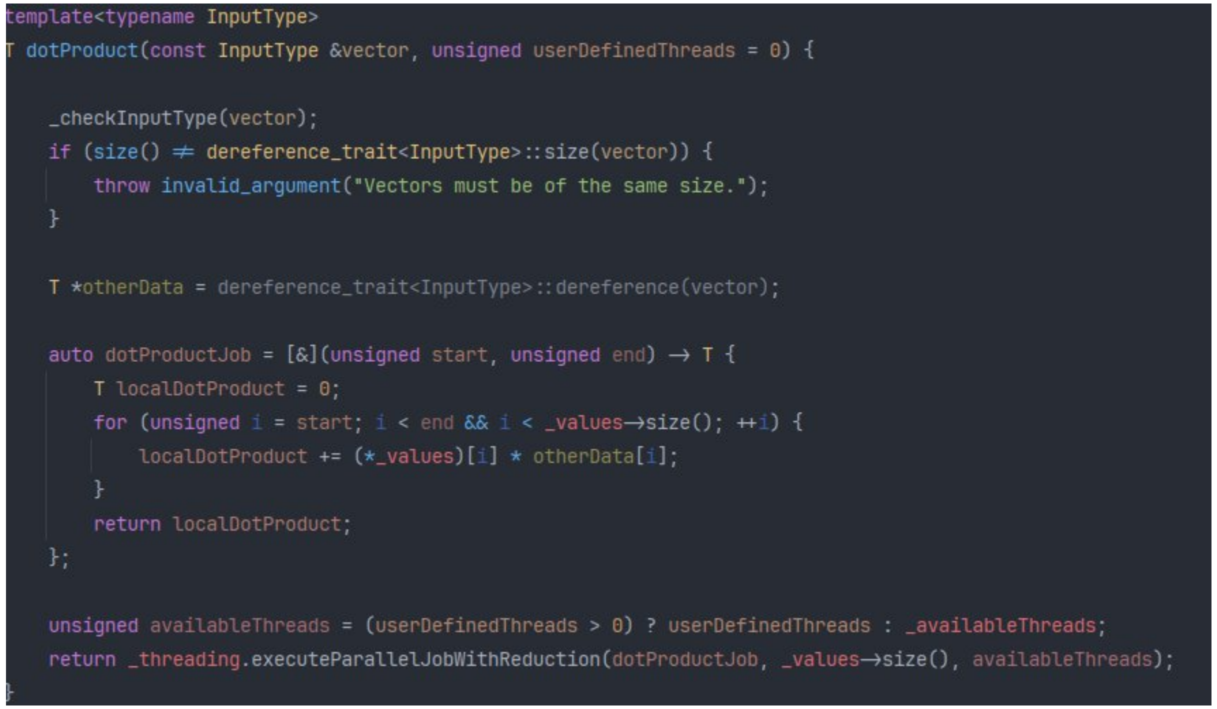
- Get the coordinates (in any coordinate system) of the qualified neighbours

- Get the weights for arbitrarily space points

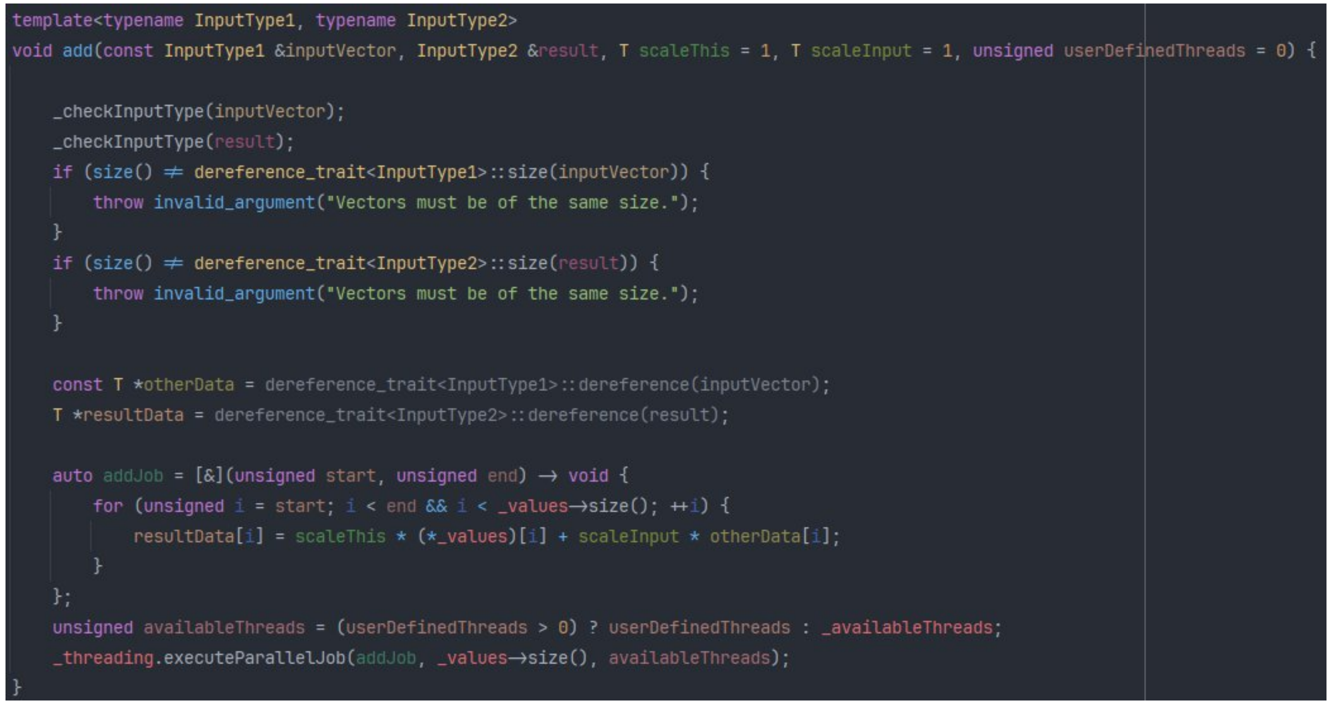
Linear Algebra – Numerical Vector

- std::vector<T> template container class
- Deference traits to operate between different types (ptr, smart ptr, stack objects)
- Operates only on numerical data types (double, unsigned, short etc.)
- Operators : =,==, !=, []
- Norms : L1, L2, LInf, Lp
- Iterators
- Operations : add, subtract, dotProduct, crossProduct,
  deepCopy, scale, sum, magnitude, average, normalize,
  distance, angle, fillRandom, variance, covariance,
  correlation, standardDeviation

Linear Algebra – Numerical Vector



```cpp
template<typename InputType>
T dotProduct(const InputType &vector, unsigned userDefinedThreads = 0) {

    _checkInputType(vector);
    if (size() ≠ dereference_trait<InputType>::size(vector)) {
        throw invalid_argument("Vectors must be of the same size.");
    }

    T *otherData = dereference_trait<InputType>::dereference(vector);

    auto dotProductJob = [&](unsigned start, unsigned end) → T {
        T localDotProduct = 0;
        for (unsigned i = start; i < end && i < _values→size(); ++i) {
            localDotProduct += (*_values)[i] * otherData[i];
        }
        return localDotProduct;
    };

    unsigned availableThreads = (userDefinedThreads > 0) ? userDefinedThreads : _availableThreads;
    return _threading.executeParallelJobWithReduction(dotProductJob, _values→size(), availableThreads);
}
```



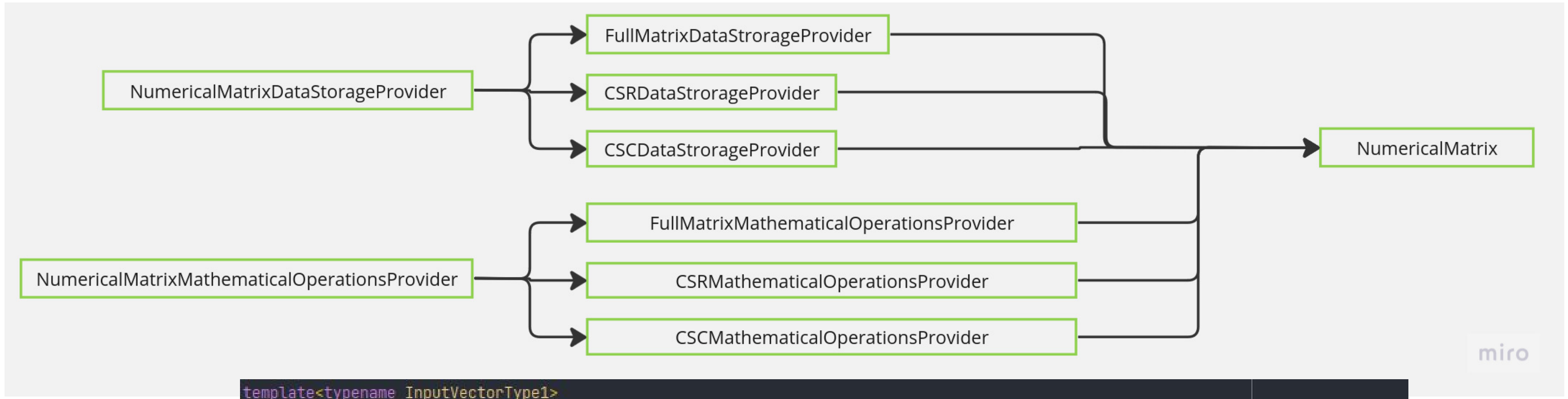
```cpp
template<typename InputType1, typename InputType2>
void add(const InputType1 &inputVector, InputType2 &result, T scaleThis = 1, T scaleInput = 1, unsigned userDefinedThreads = 0) {

    _checkInputType(inputVector);
    _checkInputType(result);
    if (size() ≠ dereference_trait<InputType1>::size(inputVector)) {
        throw invalid_argument("Vectors must be of the same size.");
    }
    if (size() ≠ dereference_trait<InputType2>::size(result)) {
        throw invalid_argument("Vectors must be of the same size.");
    }

    const T *otherData = dereference_trait<InputType1>::dereference(inputVector);
    T *resultData = dereference_trait<InputType2>::dereference(result);

    auto addJob = [&](unsigned start, unsigned end) → void {
        for (unsigned i = start; i < end && i < _values→size(); ++i) {
            resultData[i] = scaleThis * (*_values)[i] + scaleInput * otherData[i];
        }
    };
    unsigned availableThreads = (userDefinedThreads > 0) ? userDefinedThreads : _availableThreads;
    _threading.executeParallelJob(addJob, _values→size(), availableThreads);
}
```

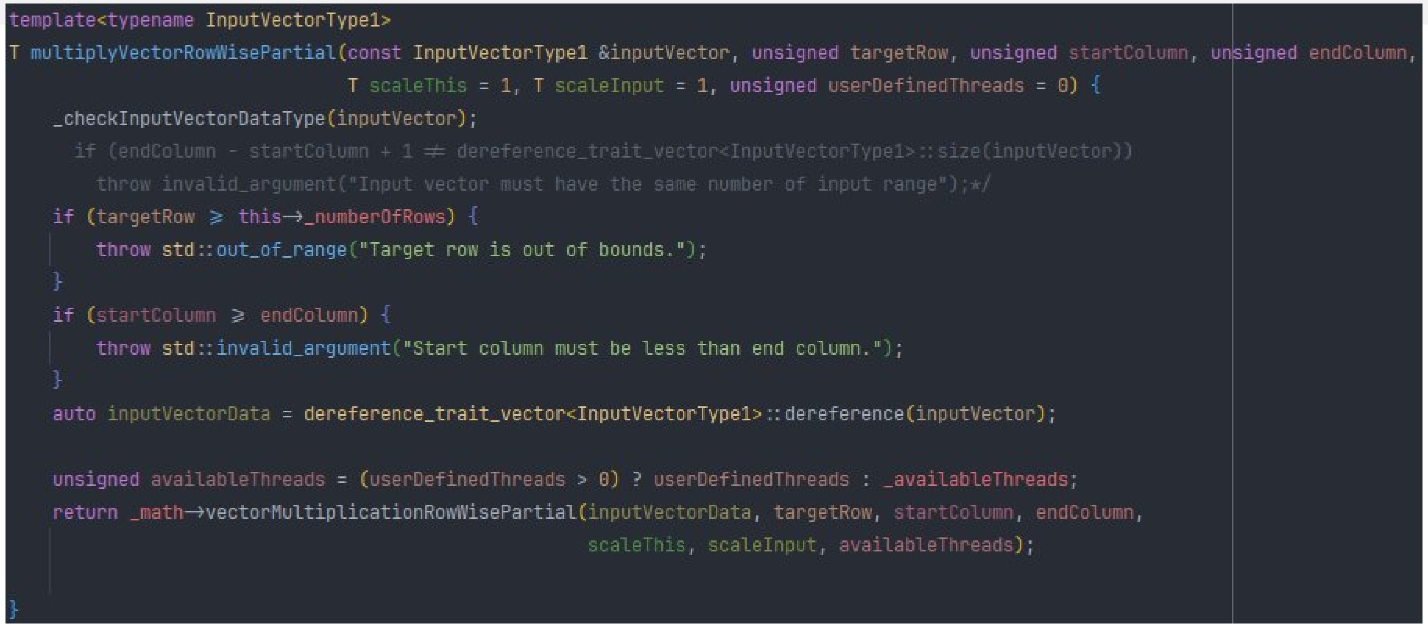
Linear Algebra – Numerical Matrix

NumericalVector<T> template container class

- Deference traits to operate between different types (ptr, smart ptr, stack objects)
- Operates only on numerical data types (double, unsigned, short etc.)
- Operators : =,==, !=, []
- Norms : L1, L2, LInf, Lp
- Operations : add, subtract, matrixMultiply, vectorMultiply, vectorMultiplyPartial
- Export to .m file

Linear Algebra – Numerical Matrix


NumericalMatrixDataStorageProvider
FullMatrixDataStrorageProvider
CSRDataStrorageProvider
CSCDataStrorageProvider
NumericalMatrix
NumericalMatrixMathematicalOperationsProvider
FullMatrixMathematicalOperationsProvider
CSRMathematicalOperationsProvider
CSCMathematicalOperationsProvider
miro



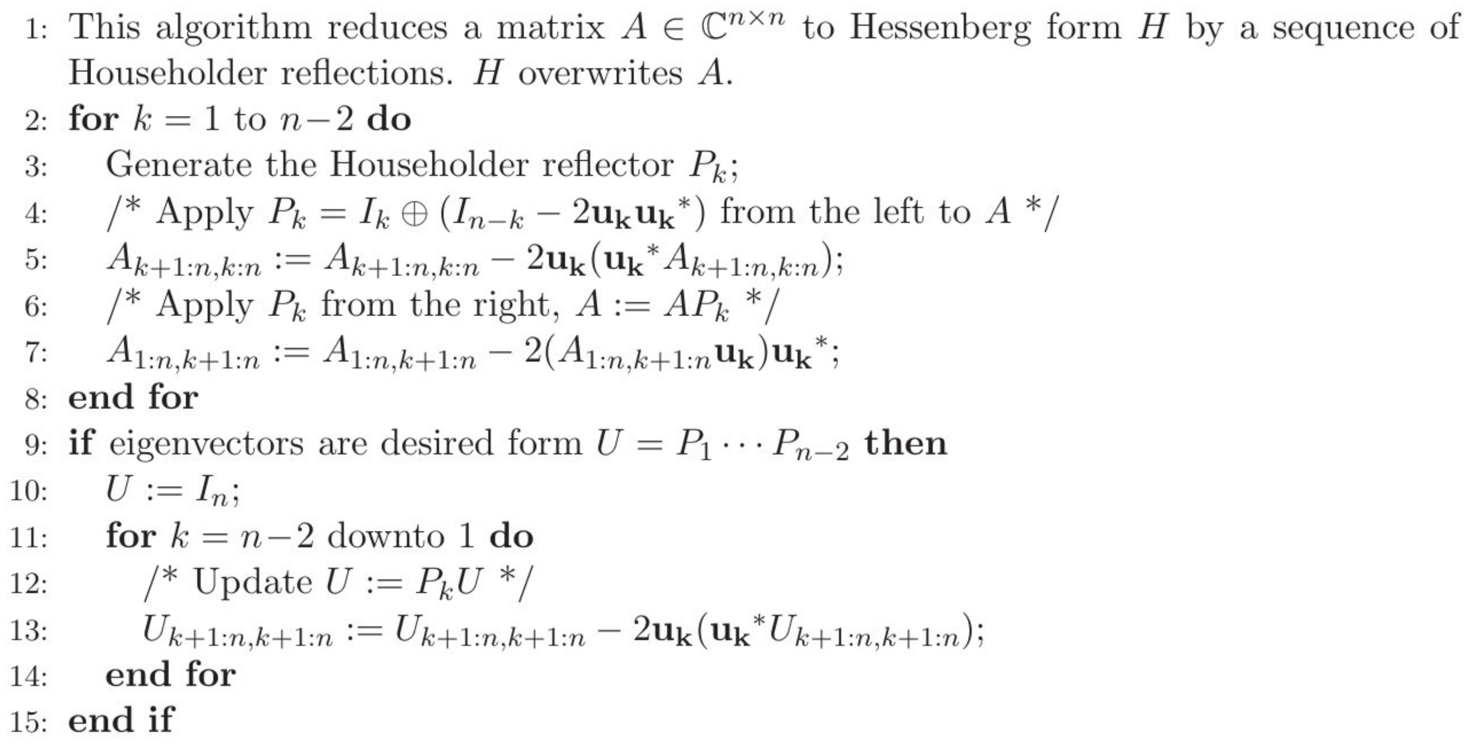
```cpp
template<typename InputVectorType1>
T multiplyVectorRowWisePartial(const InputVectorType1 &inputVector, unsigned targetRow, unsigned startColumn, unsigned endColumn,
                               T scaleThis = 1, T scaleInput = 1, unsigned userDefinedThreads = 0) {
    _checkInputVectorDataType(inputVector);
     if (endColumn - startColumn + 1 ≠ dereference_trait_vector<InputVectorType1>::size(inputVector))
      throw invalid_argument("Input vector must have the same number of input range");*/
    if (targetRow ≥ this→_numberOfRows) {
        throw std::out_of_range("Target row is out of bounds.");
    }
    if (startColumn ≥ endColumn) {
        throw std::invalid_argument("Start column must be less than end column.");
    }
    auto inputVectorData = dereference_trait_vector<InputVectorType1>::dereference(inputVector);

    unsigned availableThreads = (userDefinedThreads > 0) ? userDefinedThreads : _availableThreads;
    return _math→vectorMultiplicationRowWisePartial(inputVectorData, targetRow, startColumn, endColumn,
                                                    scaleThis, scaleInput, availableThreads);

}
```

Linear Algebra – Solvers

- Stationary
  - LUP
  - Cholesky
- Block Iterative with multi-thread vector operations
  - Conjugate Gradient
- Point Iterative
  - Jacobi
  - Parallel Jacobi (Multithread & CUDA)
  - SOR
  - Gauss Seidel

Linear Algebra – Eigendecomposition

- QR Decomposition with Householder Transformation
  - Industry standard algorithm for every type of matrix (LAPACK, MATLAB, SciPy)
  - Expensive but very powerful method
  - Can be as used as a “precondtioner” for iterative eigendecompostion methods



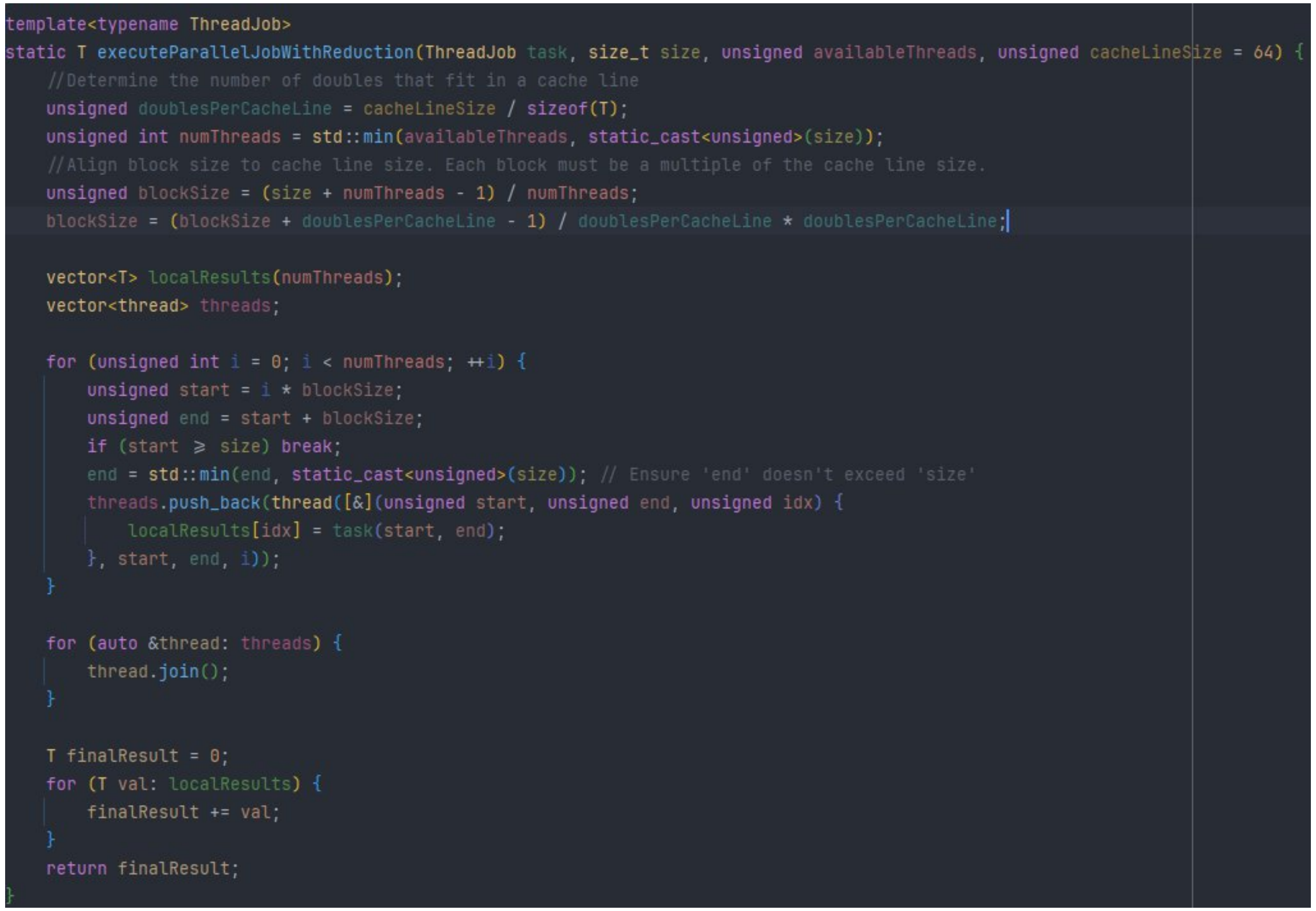
```
This algorithm reduces a matrix $A \in \mathbb{C}^{n\times n}$ to Hessenberg form $H$ by a sequence of
Householder reflections. $H$ overwrites $A$.
for $k = 1$ to $n-2$ do
    Generate the Householder reflector $P_k$;
    /* Apply $P_k = I_k \oplus (I_{n-k} - 2\mathbf{u_k}\mathbf{u_k}^*)$ from the left to $A$ */
    $A_{k+1:n,k:n} := A_{k+1:n,k:n} - 2\mathbf{u_k}(\mathbf{u_k}^* A_{k+1:n,k:n})$;
    /* Apply $P_k$ from the right, $A := AP_k$ */
    $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n}\mathbf{u_k})\mathbf{u_k}^*$;
end for
if eigenvectors are desired form $U = P_1 \cdots P_{n-2}$ then
    $U := I_n$;
    for $k = n-2$ downto $1$ do
        /* Update $U := P_k U$ */
        $U_{k+1:n,k+1:n} := U_{k+1:n,k+1:n} - 2\mathbf{u_k}(\mathbf{u_k}^* U_{k+1:n,k+1:n})$;
    end for
end if
```

Linear Algebra – Eigendecomposition

- Lanczos Iteration
  - Fast Krylov-subspace method
  - Finds some eigenvalues of the matrix
  - Can be optimized by applying an approximate of an eigenvector as initial solution (from a QR iteration)

- Power Method
  - Finds the most dominant eigenvalue of the matrix
  - Very fast

Linear Algebra – Threading

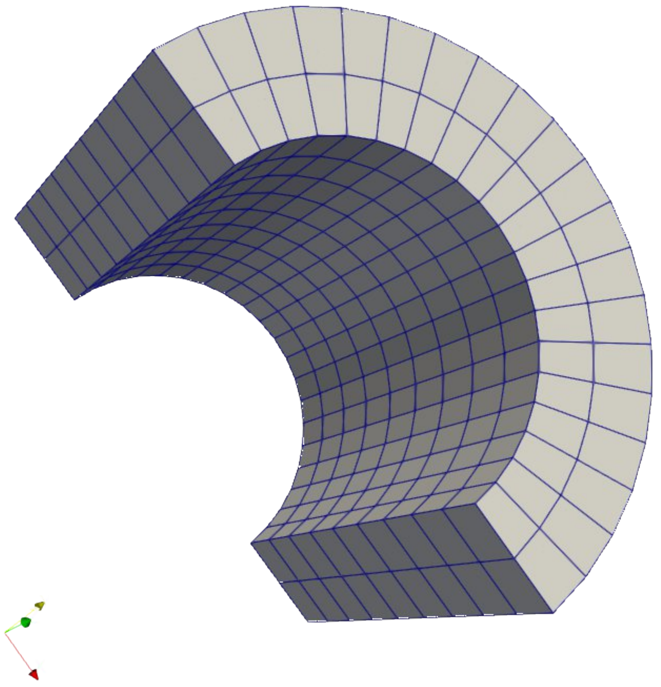
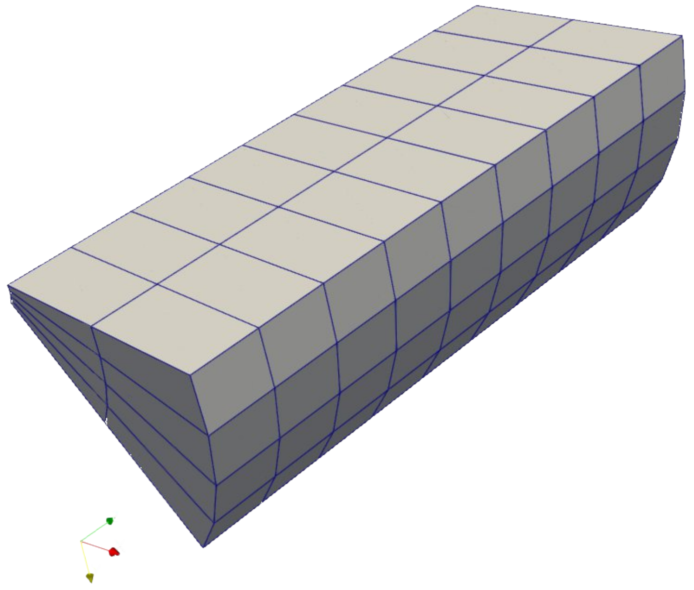
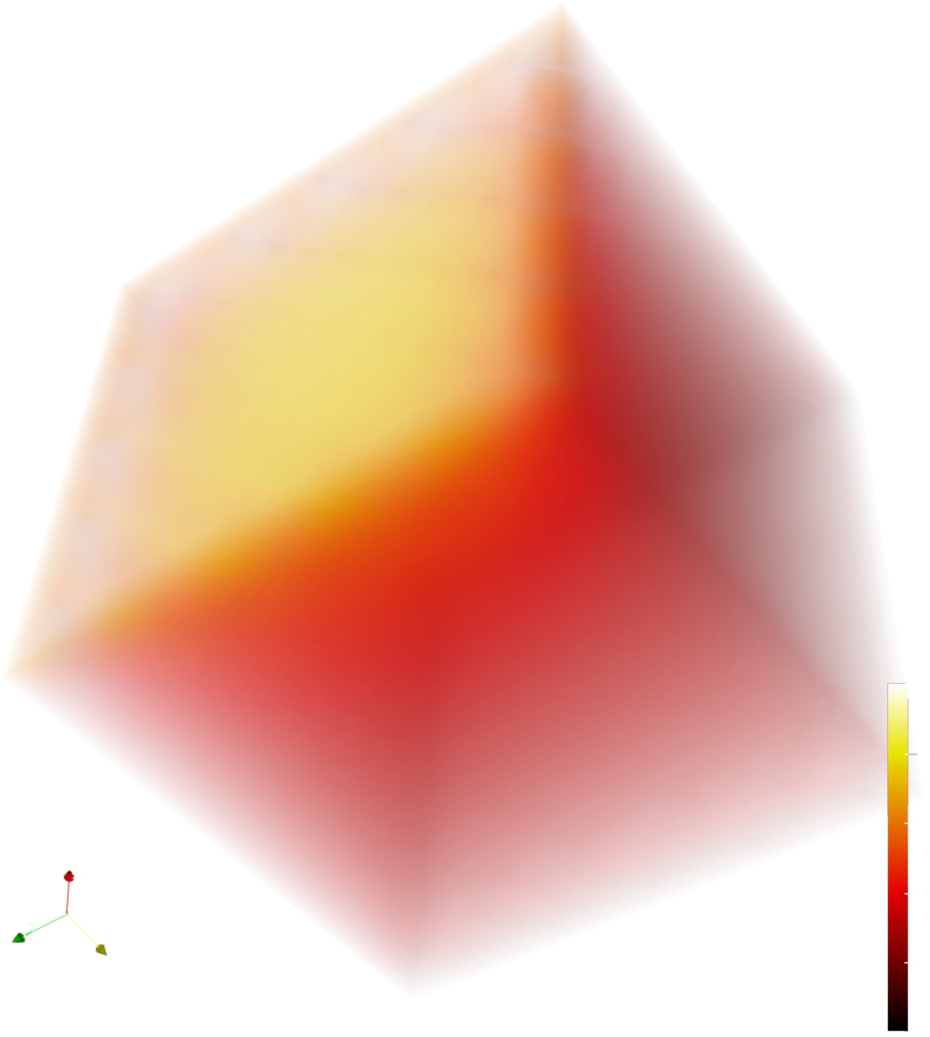


```cpp
template<typename ThreadJob>
static T executeParallelJobWithReduction(ThreadJob task, size_t size, unsigned availableThreads, unsigned cacheLineSize = 64) {
    //Determine the number of doubles that fit in a cache line
    unsigned doublesPerCacheLine = cacheLineSize / sizeof(T);
    unsigned int numThreads = std::min(availableThreads, static_cast<unsigned>(size));
    //Align block size to cache line size. Each block must be a multiple of the cache line size.
    unsigned blockSize = (size + numThreads - 1) / numThreads;
    blockSize = (blockSize + doublesPerCacheLine - 1) / doublesPerCacheLine * doublesPerCacheLine;

    vector<T> localResults(numThreads);
    vector<thread> threads;

    for (unsigned int i = 0; i < numThreads; ++i) {
        unsigned start = i * blockSize;
        unsigned end = start + blockSize;
        if (start >= size) break;
        end = std::min(end, static_cast<unsigned>(size)); // Ensure 'end' doesn't exceed 'size'
        threads.push_back(thread([&](unsigned start, unsigned end, unsigned idx) {
            localResults[idx] = task(start, end);
        }, start, end, i));
    }

    for (auto &thread: threads) {
        thread.join();
    }

    T finalResult = 0;
    for (T val: localResults) {
        finalResult += val;
    }
    return finalResult;
}
```

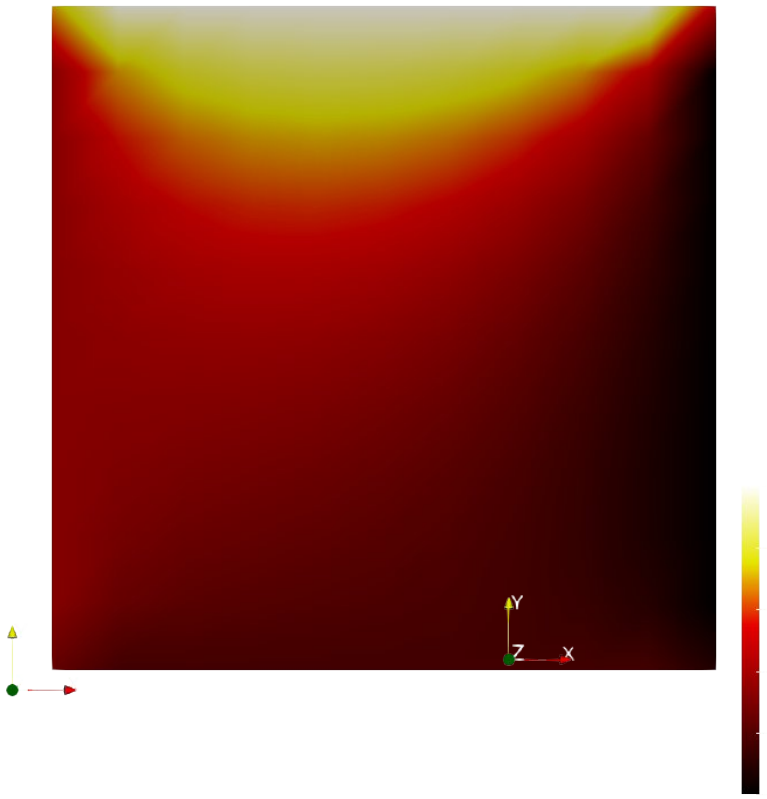
Linear Algebra – Threading

- All CPU operations of vectorized data structures are performed by executeParallelJob and executeParallelJobWithReduction
- Each thread executes the operations for a specific number of rows
- Block size (rows operated by thread) tries to align with the cache line size, in order to be cache friendly

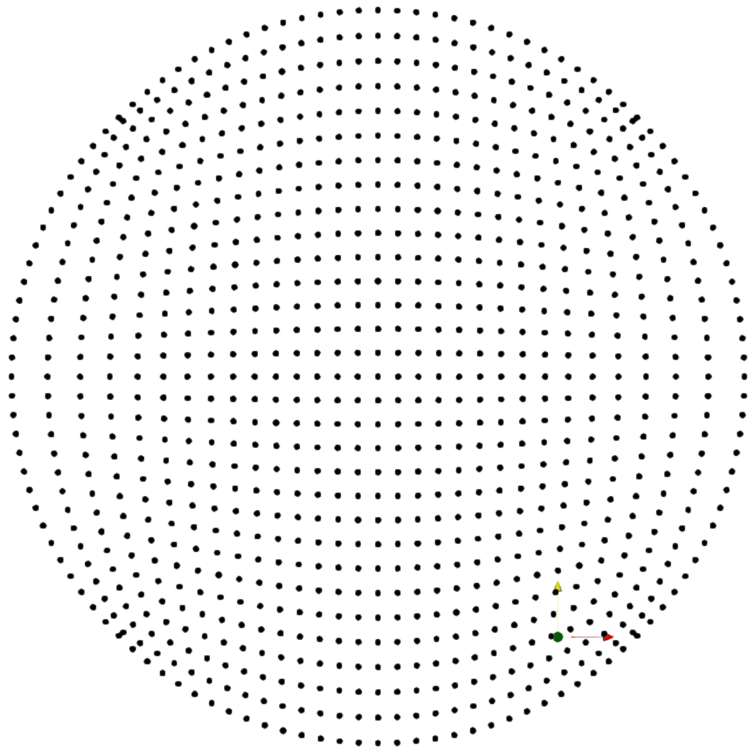
Results

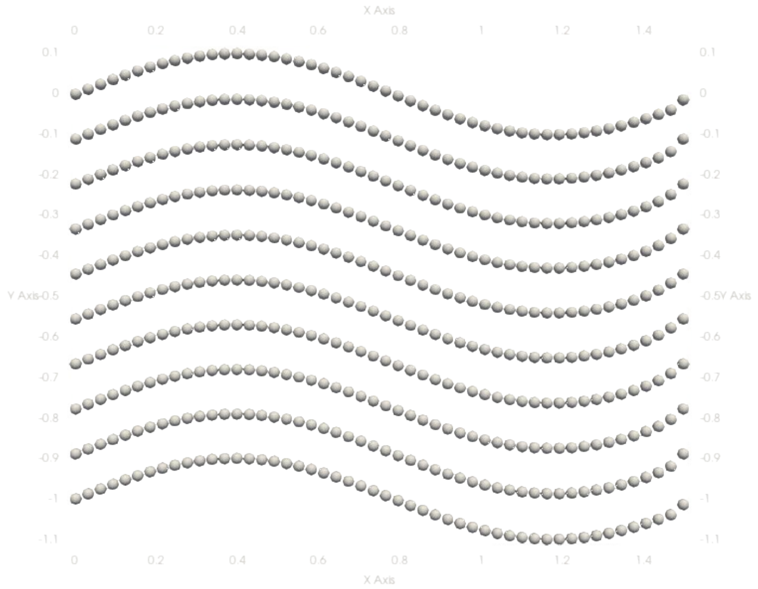




Results



Y
Z
X





X Axis
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0.1
0
-0.1
-0.2
-0.3
-0.4
Y Axis -0.5
-0.6
-0.7
-0.8
-0.9
-1
-1.1
X Axis