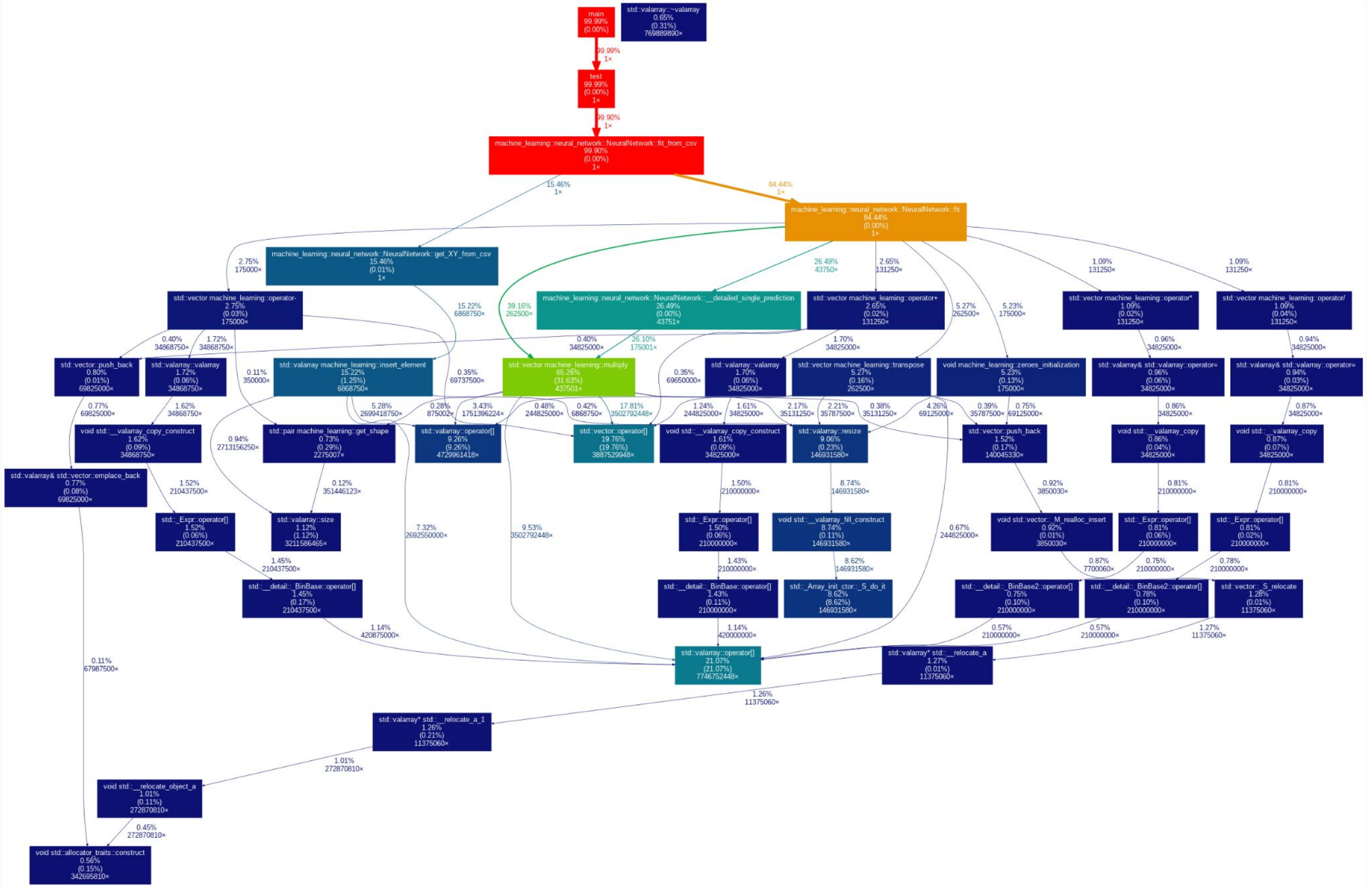
# Neural Network - C/C++

Paralelização com OpenMP

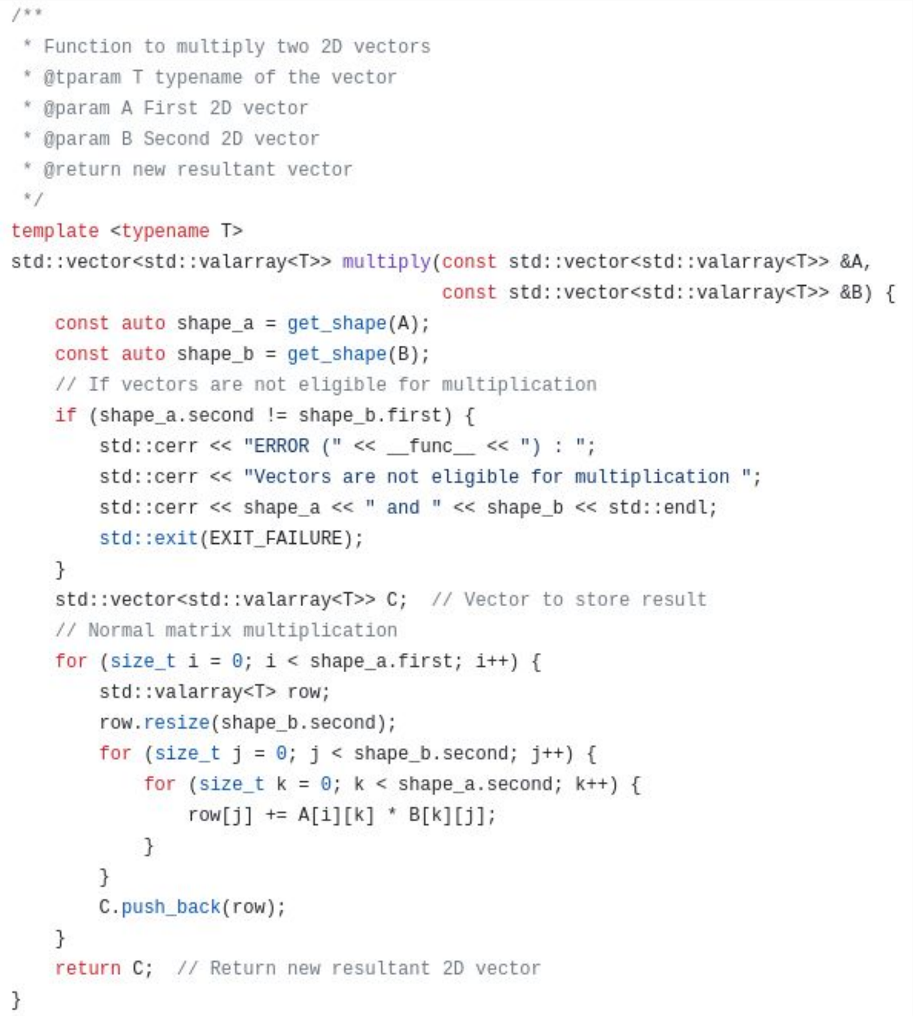
Componentes: Claudiano Leonardo da Silva,
Gabriel Barros Lins Lelis de Oliveira,
Jefferson dos Santos Leocadio,
Sidney Alves dos Santos Junior,
Victor do Nascimento Gomes

# RESULTADOS ANTERIORES


main 99.99% (0.00%)
std::valarray::~valarray 0.65% (0.31%) 769889890×
test 99.99% (0.00%) 1×
machine_learning::neural_network::NeuralNetwork::fit_from_csv 99.90% (0.00%) 1×
machine_learning::neural_network::NeuralNetwork::fit 84.44% (0.00%) 1×
machine_learning::neural_network::NeuralNetwork::get_XY_from_csv 15.46% (0.01%) 1×
std::vector machine_learning::operator- 2.75% (0.03%) 175000×
machine_learning::neural_network::NeuralNetwork::__detailed_single_prediction 26.49% (0.00%) 43751×
std::vector machine_learning::operator+ 2.65% (0.02%) 131250×
std::vector machine_learning::operator* 1.09% (0.02%) 131250×
std::vector machine_learning::operator/ 1.09% (0.04%) 131250×
std::vector::push_back 0.80% (0.01%) 69825000×
std::valarray::valarray 1.72% (0.06%) 34868750×
std::valarray machine_learning::insert_element 15.22% (1.25%) 6868750×
std::vector machine_learning::multiply 65.26% (31.63%) 437501×
std::valarray::valarray 1.70% (0.06%) 34825000×
std::vector machine_learning::transpose 5.27% (0.16%) 262500×
void machine_learning::zeroes_initialization 5.23% (0.13%) 175000×
std::valarray& std::valarray::operator= 0.96% (0.06%) 34825000×
std::valarray& std::valarray::operator= 0.94% (0.03%) 34825000×
void std::__valarray_copy_construct 1.62% (0.09%) 34868750×
std::pair machine_learning::get_shape 0.73% (0.29%) 2275007×
std::valarray::operator[] 9.26% (9.26%) 4729961418×
std::vector::operator[] 19.76% (19.76%) 3887529948×
void std::__valarray_copy_construct 1.61% (0.09%) 34825000×
std::valarray::resize 9.06% (0.23%) 146931580×
std::vector::push_back 1.52% (0.17%) 140045330×
void std::__valarray_copy 0.86% (0.04%) 34825000×
void std::__valarray_copy 0.87% (0.07%) 34825000×
std::valarray& std::vector::emplace_back 0.77% (0.08%) 69825000×
std::_Expr::operator[] 1.52% (0.06%) 210437500×
std::valarray::size 1.12% (1.12%) 3211586465×
std::_Expr::operator[] 1.50% (0.06%) 210000000×
void std::__valarray_fill_construct 8.74% (0.11%) 146931580×
void std::vector::_M_realloc_insert 0.92% (0.01%) 3850030×
std::_Expr::operator[] 0.81% (0.06%) 210000000×
std::_Expr::operator[] 0.81% (0.02%) 210000000×
std::__detail::_BinBase::operator[] 1.45% (0.17%) 210437500×
std::__detail::_BinBase::operator[] 1.43% (0.11%) 210000000×
std::_Array_init_ctor::_S_do_it 8.62% (8.62%) 146931580×
std::__detail::_BinBase2::operator[] 0.75% (0.10%) 210000000×
std::__detail::_BinBase2::operator[] 0.78% (0.10%) 210000000×
std::vector::_S_relocate 1.28% (0.01%) 11375060×
std::valarray::operator[] 21.07% (21.07%) 7746752448×
std::valarray* std::__relocate_a 1.27% (0.01%) 11375060×
std::valarray* std::__relocate_a_1 1.26% (0.21%) 11375060×
void std::__relocate_object_a 1.01% (0.11%) 272870810×
void std::allocator_traits::construct 0.56% (0.15%) 342695810×

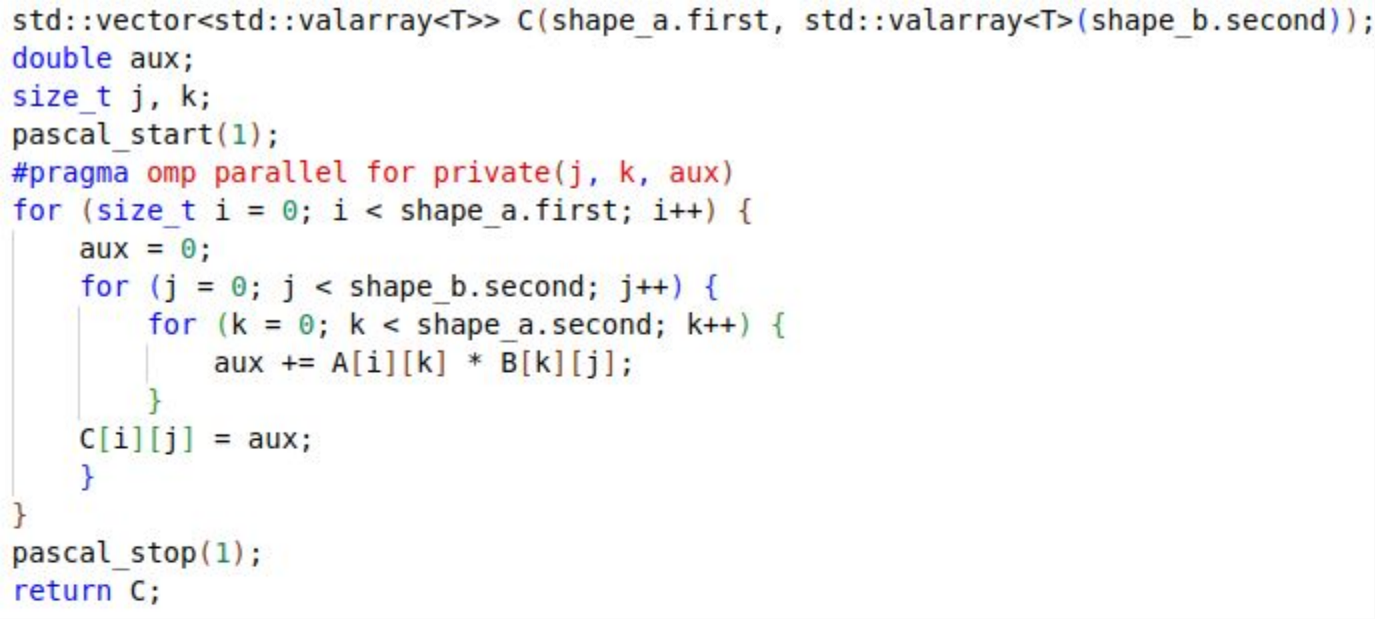
# PROBLEMA

Laços aninhados para a realização da multiplicação de matrizes



```
/**
 * Function to multiply two 2D vectors
 * @tparam T typename of the vector
 * @param A First 2D vector
 * @param B Second 2D vector
 * @return new resultant vector
 */
template <typename T>
std::vector<std::valarray<T>> multiply(const std::vector<std::valarray<T>> &A,
                                       const std::vector<std::valarray<T>> &B) {
    const auto shape_a = get_shape(A);
    const auto shape_b = get_shape(B);
    // If vectors are not eligible for multiplication
    if (shape_a.second != shape_b.first) {
        std::cerr << "ERROR (" << __func__ << ") : ";
        std::cerr << "Vectors are not eligible for multiplication ";
        std::cerr << shape_a << " and " << shape_b << std::endl;
        std::exit(EXIT_FAILURE);
    }
    std::vector<std::valarray<T>> C;  // Vector to store result
    // Normal matrix multiplication
    for (size_t i = 0; i < shape_a.first; i++) {
        std::valarray<T> row;
        row.resize(shape_b.second);
        for (size_t j = 0; j < shape_b.second; j++) {
            for (size_t k = 0; k < shape_a.second; k++) {
                row[j] += A[i][k] * B[k][j];
            }
        }
        C.push_back(row);
    }
    return C;  // Return new resultant 2D vector
}
```

# PORTAR PARA O OPENMP



```cpp
std::vector<std::valarray<T>> C(shape_a.first, std::valarray<T>(shape_b.second));
double aux;
size_t j, k;
pascal_start(1);
#pragma omp parallel for private(j, k, aux)
for (size_t i = 0; i < shape_a.first; i++) {
    aux = 0;
    for (j = 0; j < shape_b.second; j++) {
        for (k = 0; k < shape_a.second; k++) {
            aux += A[i][k] * B[k][j];
        }
    C[i][j] = aux;
    }
}
pascal_stop(1);
return C;
```

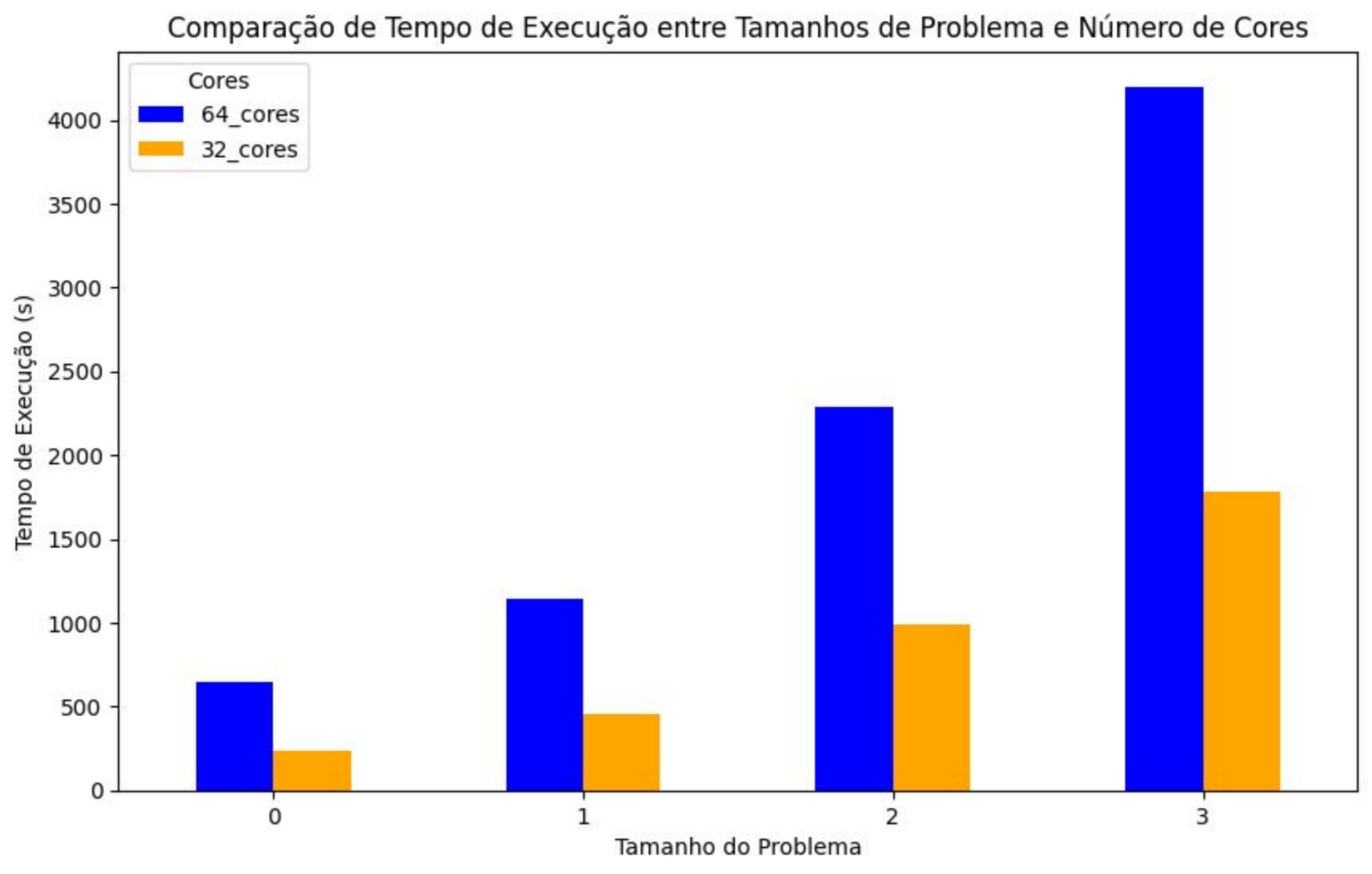
# DIRETIVAS

```
#pragma omp parallel

for

private(j, k, aux)
```

# MODIFICAÇÕES

```
double  aux;

size_t j, k;

std::vector<std::valarray<T>> C(shape_a.first, std::valarray<T>(shape_b.second));

pascal_start(1);

pascal_stop(1);
```

# RESULTADOS PARCIAIS


Comparação de Tempo de Execução entre Tamanhos de Problema e Número de Cores
Cores
64_cores
32_cores
Tempo de Execução (s)
Tamanho do Problema