Institut für Informatik — TUM
Scientific Computing in Computer Science
R. Wittmann

WT 2015

Tutorial (Advanced Programming) Worksheet 11:

Assignment 1: Cache optimization

In the last worksheet we did some first optimization attempts for the matrix-matrix multiplication algorithm:

$$C = A \cdot B, \text{ with } C \in \mathbb{R}^{rows \times columns}, A \in \mathbb{R}^{rows \times k}, B \in \mathbb{R}^{k \times columns}$$

In this assignment, we will show how one can modify the matrix-matrix multiplication itself to enhance cache utilization. For this, we will rewrite the matrix-matrix multiplication into a blocked matrix-matrix multiplication as follows:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

In the remainder of this assignment you shall now extend this approach to use even more blocks. To simplify things a bit, you may now further assume that each matrix block has size $b \times b$, where b will be provided through the stub implementation `mm_mult_cache.cpp` on moodle. Implement the blocked matrix-matrix multiplication in the function `mm_mult_rck_block`. The corresponding test function will now be `testFunctionBlock`.

1. Determine the total number of blocks in our big matrices A, B and C. Then, think about how you can you map position inside a given block to a position in the big matrix. This is required as we are still working on the original big matrices.

2. Start with three loops which traverse over all required blocks in a (block-row, block-column, block-k) = (br,bc,bk) fashion. Further set up pointers which point to the first element of each involved block.

3. Finally, we kick off the actual computation by adding the three more loop levels to actually compute the small matrix multiplication. For this, use the loop ordering (row,column,k) as well. In total you should end up with the following loop ordering: (block-row,block-column,block-k,row,column,k). You might have to use some code transformations here as well to handle the first values in C_{ij}

When you run the test setup you should notice a performance improvement with larger block sizes up to a certain point. However, after this point the performance will start to decrease with even larger block sizes. What is the reason for this behavior and do you know a way how one could find the optimum block size?

Class Assignment 2: Square function using templates

In this assignment, we will look at different ways on how a square function may be implemented for different data types. Please download the file `templates.cpp` from moodle which provides stub implementations for the following assignments.

- Work through the code and get familiar with the different implementations of the `square` function and how they are used.
- Explain how the compiler decides which implementation to use when calling the `square` function.
- Override this decision by explicitly calling the specialized version of the `square` function for the `int` data type.

The provided code skeleton also provides a macro for the `square` operation. Activate it by compiling with the flag `-DSQUAREMACRO`. You will notice that the file does not compile anymore. Explain this behavior and move the macro to a safer location in the code. Then explain the effects which you observe after the change and restore the intended behavior by fixing the macro.

Assignment 3: Generic array class using templates

We can use templates to provide more flexible memory management for arbitrary data types. Implement a template class `TArray` which offers the following functionality:

- The class shall work for any data type, especially user defined one.
- The constructor takes an argument `size` which will allocate `size` elements of the provided data type on the heap.
- The destructor shall free the allocated memory.
- The class shall provide a `size()` method, which returns the total number of allocated elements.
- The class shall provide an implementation for `operator()[int idx]` which returns a reference to the element at position `idx`.

If you have finished your implementation enable the macro `ASSIGNMENT2_DONE` in the provided source code to enable the test routine.