Research Report: Study on Dense Vector Operation with Expression Template

Chen Fang

March 29, 2015

Abstract

It is a common belief that the expression template (ET) helps achieve as efficient C++ codes as manual implementations of C-like codes while providing great convenience for users to form the expressions. Most researches focus on sufficiently large operations that benefit the most out of the ET technique. We, however, limit this study on ET in the context of dense linear algebra operations within reservoir simulations where vector sizes are small. Several benchmarks are generated to reveal some limitations of ET, followed by possible solutions and advices for future applications.

1 Motivation of Study

In reservoir simulation, variables have different properties in terms of their dependence on the unknown variables. Due to the discretization scheme, the discretized terms in the governing equations will be evaluated in completely different manners. In black-oil model, we have the following expression. The accumulation term depends on variables only from the central grid block while the flux term involving heavy stenciling, making it dependent on variables from other grid blocks. In compositional simulation, phase behavior and its entailed choices of variables make the evaluation process even more complicated. The essence of which, however, remains unchanged and stays on vector operations. For clarity and brevity, the following analysis will be built upon black oil simulation. The common evaluation procedures of the governing equation is as follows: 1. Evaluate individual properties, for instance, density, porosity, volume factors and so on; 2. Evaluate the accumulation term; 3. Evaluate each term involved in the flux term; 4. Evaluate the flux term by combining all individual terms.

A general solution is to apply sparse data structure to the entire problem, but the computation performance could be driven from theoretical optimum because of the intense index comparison and the resulting pipelining disruption. In the above procedures, the evaluation of some variables and also the accumulation term is completely sparse because they depend on only some other variable from the same grid block. In this case, sparse operation is no longer a wise choice because all the extra work for indices are unnecessary and thus wasted. Therefore, it is important to specify the most suitable data structure for various scenarios. There are three most common implementations for dense operators with using C++. The first is the operator overloading, which provides natural mathematical expression for users but suffers from temporary objects and repeated for-loops due

to pairwise evaluations. The second implementation is the hand-written fused loop, which carries out all computations within a single loop. Theoretically this is the most efficient method without any other optimization methods except from the compiler. But it requires explicit expressions and indexing, making it hard to code and maintain as the expression get more and more complicated. To combine the advantages of the two, we can use the expression template technique. The detail of ET will be illustrated as follows.

2 Expression Template

Expression template was first introduced by someone [reference]. Rather than immediate pairwise evaluation, a temporary type containing its subexpression's information is generated and returned. A parse tree consisted of these temporary nodes is built and no real arithmetic evaluation is performed until the assignment operator is encounter, which gives the signal to start the evaluation process by querying information recursively. With the help of type deduction and inlining mechanism, this repeated query process can be fully optimized away such that the identical assembly code can be generated as the hand-written fused loop. [Advantage]

[Disadvantage]

3 Benchmarks

Different from massive computational tasks like matrix-vector multiplication or matrix-matrix multiplication, the vectors to be evaluated in reservoir simulation is rather small and the dense stage of that is even smaller. The size depends on the number of unknown variables defined in the problem.

[Test Environment]

[Test Cases]

4 Summary