Working Title

Adam Fidel, Lena Olson, Antal Buss, Timmie Smith, Gabriel Tanase,
Nathan Thomas, Mauro Bianco, Nancy M. Amato, Lawrence Rauchwerger

Parasol Lab, Dept. of Computer Science, Texas A&M University
{afidel,lolson,abuss,timmie,gabrielt,nthomas,bmm,amato,rwerger}@cs.tamu.edu

Abstract

The Standard Template Adaptive Parallel Library (`STAPL`) is a superset of C++'s Standard Template Library (`STL`) which allows high-productivity parallel programming in both distributed and shared memory environments. This framework provides parallel equivalents of `STL` containers and algorithms enabling ease of development for parallel systems. In this paper, we will discuss our methodology for implementing a fast and efficient matrix multiplication algorithm in `STAPL`. Our implementation employs external linear algebra libraries, specifically the Basic Linear Algebra Subprograms (`BLAS`) library which includes highly optimized matrix operations. The paper will describe the benefits of creating a parallel matrix multiplication algorithm whose library calls are specialized based on both the matrix storage and traversal. This specialization technique ensures that the most appropriate implementation in terms of data access and structure will be used, resulting in increased efficiency compared to a non-specialized approach.

1 Introduction

Parallel libraries such as the Standard Template Adaptive Parallel Library (`STAPL`) allow developers to focus their programming efforts on higher-level abstract issues, rather than the intricacies of the parallelization itself. By providing a standard set of operations and procedures to the programmer, parallel code can be produced in a manner comparable to the development of sequential programs. With this in mind, we sought to incorporate a matrix multiplication algorithm which took advantage of parallelization in a way that is transparent to the end developer.

In STAPL, one of the major components is the `pAlgorithms`, which are the direct equivalents of `STL`'s sequential algorithms. The input for these algorithms includes a view, which is an abstraction of the data access, and a work function which specifies the operations to be executed on the data. In the case of matrix multiplication, a view can be taken over an entire row, an entire column or a block of both rows and columns.

2 Background

Matrix multiplication is a key part of many applications within scientific computing, where matrices can also be very large. For this reason, it is important that a parallel framework be able to handle distributed matrix multiplication in an efficient way. The manner in which the data is stored should also be flexible, in order to fit the needs of various applications.

For serial matrix multiplication, `BLAS` can be used. `BLAS` contains a number of highly optimized subroutines, including the level 3 subroutine general matrix multiply (`gemm`). By basing a parallel implementation on `gemm`, it is unnecessary to optimize the serial multiplication, instead relying on `BLAS` to provide an efficient implementation.

3 Proposed Method

Each matrix can be stored in block-band format in one of two directions, hence there are eight combinations of data partitioning possible. A different algorithm is needed for each case, because the data locality determines which of the three matrices should be rotated, as well as which elements to multiply and where to store the result. However, six of these algorithms are very similar; only when the matrices to be multiplied are stored partitioned row-wise and the result is stored partitioned column-wise, or vice versa, is a substantially different algorithm needed. For purposes of this paper, we will explain in detail one example: the case where A is partitioned row-wise, B is partitioned column-wise, and C is partitioned row-wise.

```
 FOR 1 to num_procs
  multiply sub-blocks of A by sub-blocks of B using gemm
  store result in C
  rotate sub-blocks of B
END FOR
```

Figure 1: Our algorithm

In addition to the eight algorithms discussed above, each algorithm also has several specializations. Currently, there are specializations for when local blocks are stored in either row-major or column-major ordering. These specializations simply determine the correct values and transpose flags to pass to the gemm call. In the future it may be desirable to extend the algorithms to handle local blocks stored in other formats, such as smaller blocks; this could be accomplished by adding a new, more complicated specialization which would multiply smaller blocks together.

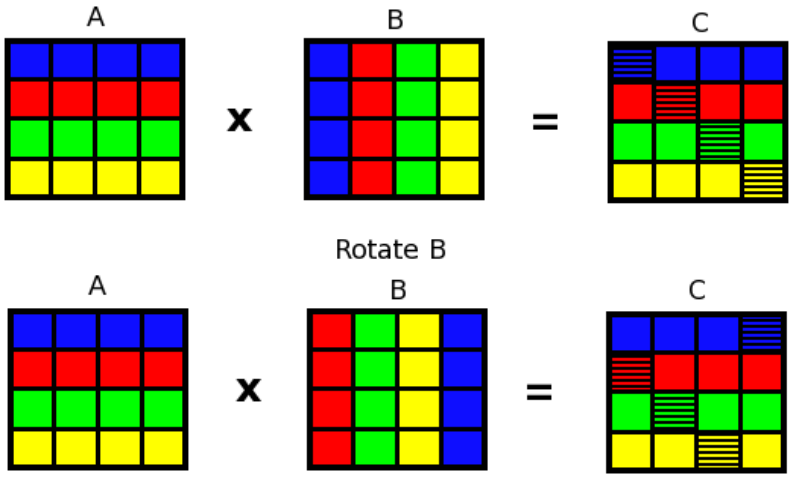
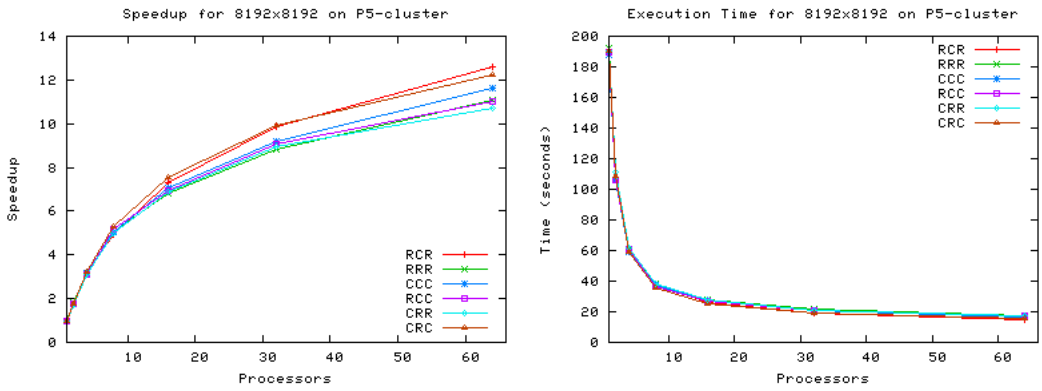


Figure 2: Example of the roll algorithm

4 Experimental Results



(a) (b)

Figure 3: Execution time (a) and relative speedup (b) of several matrix multiplication specializations on P5-cluster (doubles)

5 Related Work

6 Summary