Efficient graph algorithms

Bruno Rodriguez Carrillo [1]

Supervisors: Daniel Kressner [1] & Aristeidis Mastoras [2]

[1]École Polytechnique Fédérale de Lausanne (EPFL)
[2]Computing Systems Laboratory, Huawei Zurich Research Center

Background

GraphBLAS is a framework for graph algorithms that are expressed as linear algebra operations. The key idea behind GraphBLAS is to leverage the mathematical foundations of linear algebra to provide a unified framework for graph computations. GraphBLAS classifies operations on scalars, vectors and matrices as level-1, level-2 and level-3 operations, respectively. The GraphBLAS API C establishes two execution modes for each GraphBLAS function [1]. In the *blocking* mode, invoking a function implies its execution; that is, once the function is called all computations the function is responsible for are performed and then the function returns. Conversely, in the *nonblocking* mode, calling a function does not necessarily imply its execution; a function in this mode may return without having performed its operations. The main idea of the nonblocking mode is to delay the execution of a GraphBLAS function so that it is executed whenever it is necessary for correct results in the program. On the other hand, ALP/GraphBLAS is a `C++11` implementation that follows the GraphBLAS API C [4].

Sparse matrix - sparse matrix (SpM-SpM) product

In the blocking mode of ALP/GraphBLAS, the SpM-SpM product $\mathbf{C} = \mathbf{AB}$ for matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$ is performed by using the Gustavson's row-wise methods, i.e., $\mathbf{C}$ is computed one row at a time based on (1).

$$\mathbf{C}(i,:) = \sum_k \mathbf{A}(i,k)\mathbf{B}(k,:), \tag{1}$$

where $\mathbf{A}(i,k)$ and $\mathbf{B}(k,:)$ represent the element of $\mathbf{A}$ at position i,k and the kth row of $\mathbf{B}$, respectively.

In ALP/GraphBLAS, the operation $\mathbf{C} = \mathbf{AB}$ is performed by the function `mxm(C,A,B,phase)`. Invoking `mxm(C,A,B,RESIZE)` implies allocating sufficient memory for storing all nonzeros of $\mathbf{C}$ whereas `mxm(C,A,B,EXECUTE)` performs the actual computations.

Let us say that in a program we are interested in computing $\mathbf{C} = \mathbf{AB}$ and $\mathbf{E} = \mathbf{CD}$ for $\mathbf{D}, \mathbf{E} \in \mathbb{R}^{n \times n}$. These operations access $\mathbf{C}$. In blocking execution, these operations must be called in the program in the order: `mxm(C,A,B,RESIZE)`, `mxm(C,A,B,EXECUTE)`, `mxm(E,C,D,RESIZE)`, `mxm(E,C,D,EXECUTE)`. This implies that data from $\mathbf{C}$ is loaded into cache twice, which impacts the program performance. In the current implementation of the nonblocking mode in ALP/GraphBLAS, level-3 operations are not considered.

Problem statement

Consequently, based on the ideas presented in [2], in this work, the current design and implementation of ALP/GraphBLAS are extended such that level-3 operations are supported in the the nonblocking execution mode. Particular emphasis is found on the implementation of the spM-spM multiplication.

Application to the triangle counting problem

The implementation of `mxm` in the nonblocking execution mode is applied to the triangle counting problem. From Theorem 1.1. in [3], given the adjacency matrix $\mathbf{A}$ of an undirected or directed simple graph $G = (V, E)$ with vertices V, edges E, and n vertices, the number of triangles in G can be computed using (2).

$$\#\text{triangles in } G = \frac{1}{6}\text{tr}\left(\mathbf{A}^3\right). \tag{2}$$

References

[1] Aydin Buluç et al. "Design of the GraphBLAS API for C". In: *2017 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE. 2017, pp. 643–652.

[2] Aristeidis Mastoras, Sotiris Anagnostidis, and Albert-Jan N Yzelman. "Design and implementation for nonblocking execution in GraphBLAS: tradeoffs and performance". In: *ACM Transactions on Architecture and Code Optimization* 20.1 (2022), pp. 1–23.

[3] Nazanin Movarraei and MM Shikare. "On the number of paths of lengths 3 and 4 in a graph". In: *International Journal of Applied Mathematics Research* 3.2 (2014), p. 178.

[4] AN Yzelman et al. "A C++ GraphBLAS: specification, implementation, parallelisation, and evaluation". In: *Preprint* (2020).

Nonblocking execution of the (SpM-SpM) product

Following [2], the nonblocking execution mode in ALP/GraphBLAS is achieved by delaying the execution of ALP/GraphBLAS functions, loop fusion, loop tiling, and loop parallelization. Let use imagine that the operations $\mathbf{C} = \mathbf{AB}$, $\mathbf{D} = \mathbf{B} + \mathbf{C}$, and $s = \sum_{i,j=1}^{n} \mathbf{D}(i,j)$ are to be performed in a program. $\mathbf{C}$ and $\mathbf{D}$ are divided into 4 row-wise tiles, see Figure 1.

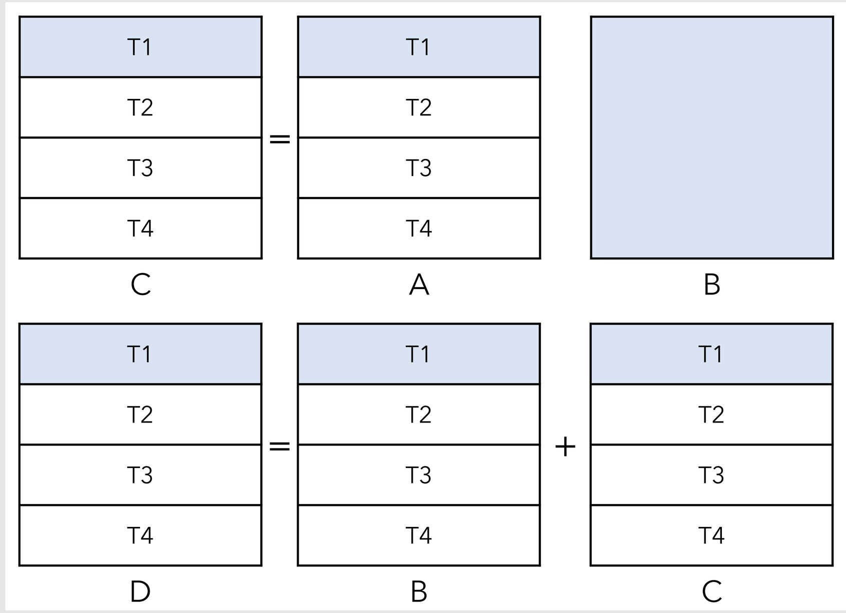

Figure 1. Matrix tiling.

Idea of nonblocking execution: tile T1 delays all its computations for $\mathbf{C}$ and $\mathbf{D}$ until it has to compute the sum of all its elements. By doing this, T1 loads into cache the data it needs from all matrices involved in the operations once; same ideas apply to the other tiles and each tile is independent from the others.

Results

Matrix in Table 1 is an adjacency matrix from an undirected graph, and density of nonzeros corresponds to $\frac{\#\text{ nnz}}{\#\text{rows}^2} * 100\%$. We compare the computing time of `mxm` in the nonblocking mode with the blocking's in equation (2).

Matrix name	#rows	#nnz	Density of nonzeros (%)	#triangles
coPapersDBLP	540,486	30,491,458	1.04E-02	444095058

Table 1. Matrix for test.

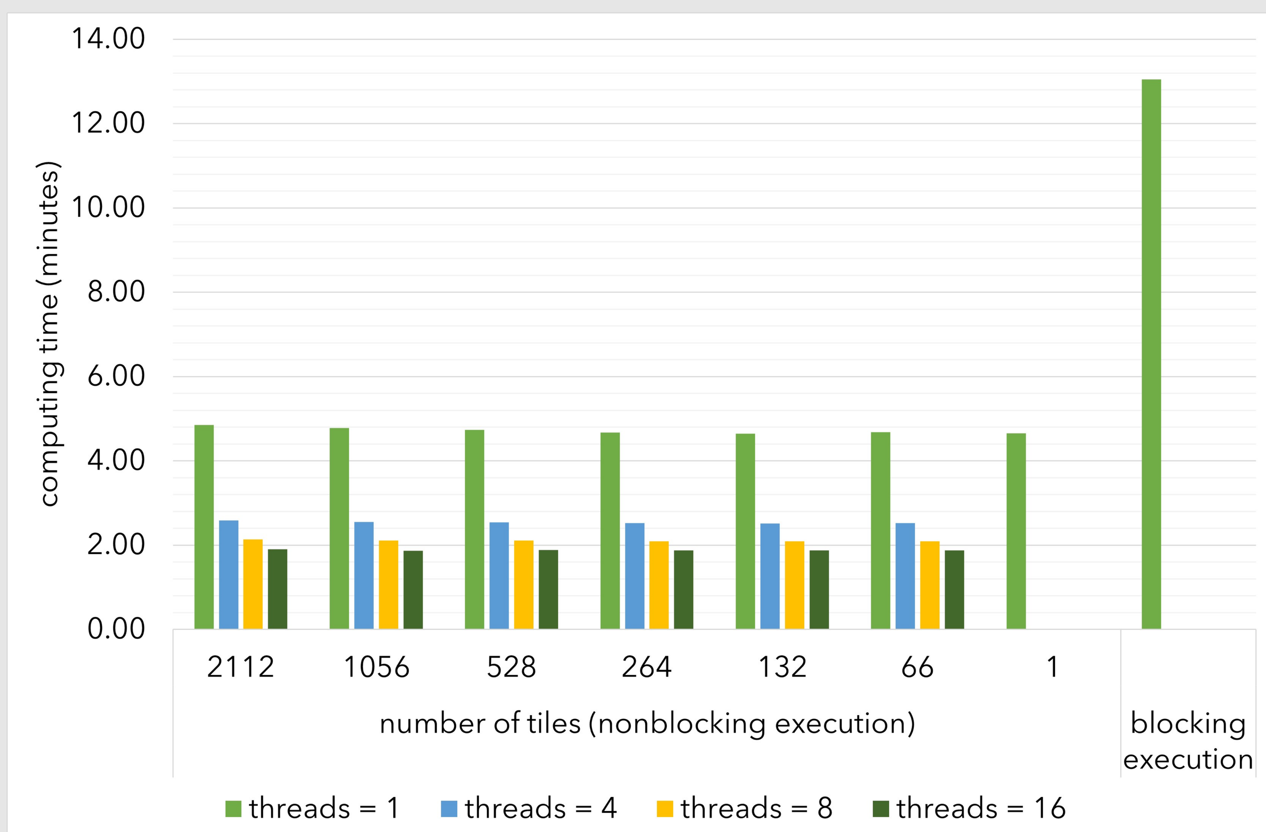

Figure 2. Computing time comparison for nonblocking and blocking execution.

Conclusions and future work

The single-threaded-single-tile execution of `mxm` in the nonblocking mode outperforms the computing time of the blocking mode due to reusage of data on cache memory. Similarly, the multi-threaded nonblocking execution leads to further speed-ups due to parallel execution. For future research, it may be worth

- testing the performance of a program that utilizes level-2 and level-3 operations in the nonblocking execution mode
- implementing distinct algorithms to compute the SpM-SpM product