Contents

Chapter 1

Introduction

1.1 Parallel sparse matrix-vector multiplication

Matrices are one of the most important mathematical objects, as they can be used to represent a wide variety of data in many scientific disciplines: they can encode the structure of a graph, define Markov chains with finitely many states, or maybe represent linear combinations of quantum states or also the behaviour of electronic components.

In most real-world computations, the systems considered are usually of very large size and involve **sparse** matrices, because the analyzed variables are usually connected to a limited number of others (for example, a very large graph in which each node has just a handful of incident edges); therefore, the matrices involved have the vast majority of entries equal to 0.

More formally, let us consider a matrix of size $m \times n$ with N nonzeros. We say that the matrix is sparse if $N \ll mn$.

One of the most fundamental operations performed in these real-world computations is the sparse matrix-vector multiplication: in which we compute

$$u := Av \tag{1.1}$$

where A denotes our $m \times n$ sparse matrix, v denotes a $n \times 1$ dense vector, and u the resulting $m \times 1$ vector.

Computing this quantity following the definition of matrix-vector multiplication, i.e. with the sum

$$u_i = \sum_{j=0}^{n-1} a_{ij} v_j \qquad 0 \leq i < m$$

requires $\mathcal{O}(n^2) = \mathcal{O}(mn)$ operations; this is not very efficient if we have a sparse matrix: if we perform the multiplications only on the nonzero elements, we obtain an algorithm with running time $\mathcal{O}(N)$, and by definition of sparsity we have that $N \ll mn$.

As already previously mentioned, the systems considered are very large, with sparse matrices with thousands (even millions) of rows and columns and millions of nonzeros; for such big instances, even a running time of $\mathcal{O}(N)$ might be non-negligible, especially since sparse matrix-vector multiplications are usually just a part of a bigger algorithm, and need to be performed several times.

It is a very important goal then to be able to perform such computations in the least amount of time possibile: however, as there is a natural tradeoff between power consumption and the speed of the processing units [1], it is not feasible to rely only on very fast CPUs, but rather focus on parallelism and employ a large number of them with lower processing speed (with low energy requirements).

To describe an efficient way of performing parallel sparse matrix-vector multiplications, we follow the approach described in [2]: before the actual computation takes place, the sparse matrix is distributed among the p processors, creating a **partitioning** of the set of the nonzeros: A is split into $A_0, \ldots, A_{p-1}$ disjoint subsets. Moreover, also the input vector v and the output vector u are distributed among the p processors (note that their distribution might not necessarily, and usually it is not, the same).

Figure 1.1 shows an example of such distribution.

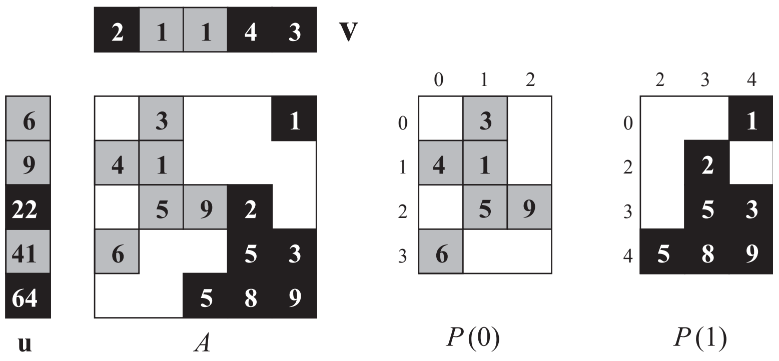

Figure 1.1: Example of a possible distribution among 2 processors of a 5x5 matrix, and the input and output vectors, as taken from [2, Fig. 4.3]

After this distribution, each processor has then to compute its local contribution toward the matrix-vector multiplication: to do so, it requires the appropriate vector component, which might have been assigned to another processor during the previous data distribution phase; if this is the case, some communication is required. After the processor has all the required vector components, it starts computing all its local distribution, once this computation step has finished, such local contributions are to their appropriate "owner" (according to the distribution of u).

These three phases for each processor (whose identity is denoted by the letter s) are summarized in Algorithm 1.1.

In reality there is also a fourth phase, in which each processor sums up all the contributions received in phase (2) for all of its own components of u; this is a very small sum with negligible computational cost and for this reason it has been omitted from the algorithm.

Our parallel algorithm, which follows the Bulk Synchronous Parallel model[3], consists of one communication superstep, followed by a computation superstep and finally another communication superstep.

1.2 Hypergraph model and current literature

1.3 Medium-grain model

Input: A_s, the local part of the vector v
Output: The local part of the vector u

```
$I_s := \{i | a_{ij} \in A_s\}$
$J_s := \{j | a_{ij} \in A_s\}$

(0)                                                          ▷ Fan-out
    for all $j \in J_s$ do
        Get $v_j$ from the processor that owns it.
    end for

(1)                         ▷ Local sparse matrix-vector multiplication
    for all $i \in I_s$ do
        $u_{is} := 0$
        for all $j$ such that $a_{ij} \in A_s$ do
            $u_{is} = u_{is} + a_{ij} v_j$
        end for
    end for

(2)                                                          ▷ Fan-in
    for all $i \in I_s$ do
        Send $u_{is}$ to the owner of $u_i$.
    end for
```

Algorithm 1.1: Parallel sparse matrix-vector multiplication.

Bibliography

[1] Jan Rabaey. *Digital integrated circuits : a design perspective.* Prentice Hall, 1996. ISBN: 0131786091 (cited on page 2).

[2] Rob. H. Bisseling. *Parallel Scientific Computation: A structured approach using BSP and MPI.* Oxford University Press, 2004 (cited on page 2).

[3] Leslie G. Valiant. "*A bridging model for parallel computation*", in: *Commun. ACM* 33.8 (Aug. 1990), pp. 103–111. ISSN: 0001-0782. DOI: `10.1145/79173.79181`. URL: `http://doi.acm.org/10.1145/79173.79181` (cited on page 2).