# 1 Parallel Implementations

We can implement our algorithms for SVD on various parallel models of computing including the PRAM and the mesh. In this section we present some of these implementations.

A shared memory model (also called the Parallel Random Access Machine (PRAM)) is a collection of RAMs working in synchrony where communication takes place with the help of a common block of shared memory [2, 1]. For instance if processor $i$ wants to communicate with processor $j$ it can do so by writing a message in memory cell $j$ which can then be read by processor $j$.

More than one processor may want to either write into or read from the same cell at the same time. Depending on how these conflicts are handled, a PRAM can further be classified into three. An Exclusive Read and Exclusive Write (EREW) PRAM does not permit concurrent reads or concurrent writes. A Concurrent Read and Exclusive Write (CREW) PRAM allows concurrent reads but not concurrent writes. Finally, a Concurrent Read and Concurrent Write (CRCW) PRAM permits both concurrent reads and concurrent writes. In the case of a CRCW PRAM, it is essential to have a mechanism for handling write conflicts, since the processors trying to write at the same time in the same cell can possibly have different data to write and we should determine which data gets written. This is not a problem in the case of concurrent reads since the data read by different processors will be the same. In a Common-CRCW PRAM, concurrent writes are permissible only if the processors trying to access the same cell have the same data to write. In an Arbitrary-CRCW PRAM, if more than one processor tries to write in the same cell at the same time, an arbitrary one of them succeeds. In a Priority-CRCW PRAM, processors have assigned priorities. Write conflicts are resolved using these priorities.

A fixed connection machine (or a fixed connection network) can be represented as a directed graph whose nodes correspond to processing elements and whose edges correspond to communication links [3, 1]. If an edge connects two processors, they can communicate in a unit step. Two processors not connected by an edge can communicate by sending a message along a path that connects the two processors. Each processor in a fixed connection machine is a RAM. Examples of fixed connection machines include the mesh, the hypercube, the star graph, etc. An $n \times n$ mesh is a $n \times n$ grid where each grid point corresponds to a processor and each edge corresponds to a (bidirectional link).

## 1.1 Some Fundamental Parallel Algorithms

In this section we describe some basic algorithms that will be employed in the parallel implementation of our SVD algorithms. These algorithms can be found in relevant texts (such as [1, 2, 3]).

Let $\oplus$ be any associative unit-time computable binary operator defined in some domain $\Sigma$. Given a sequence of $n$ elements $k_1, k_2, \ldots, k_n$ from $\Sigma$, the problem of *prefix computation* is to compute $k_1, k_1 \oplus k_2, k_1 \oplus k_2 \oplus k_3, \ldots, k_1 \oplus k_2 \oplus \cdots \oplus k_n$.

**Lemma 1.1** *Prefix computation on a sequence of $n$ elements can be performed in $O(\log n)$ time using $n$ CREW PRAM processors.*

**Lemma 1.2** *Prefix computation on a sequence of $n^2$ elements (located one element per node) can be performed in $O(n)$ time on a $n \times n$ mesh.*

**Lemma 1.3** [The slow-down lemma] *Any PRAM algorithm that runs in time $T$ using $P$ processors can be simulated on a $P'$-processor machine in time $O\left(\frac{PT}{P'}\right)$, for any $P' \leq P$.* $\square$

**Lemma 1.4** *Any sequence of arbitrary elements of length $n$ can be sorted in $O(\log n)$ time using $n$ EREW PRAM processors.* $\square$

**Lemma 1.5** *Any sequence of arbitrary elements of length $n$ can be sorted in $O(n)$ time using on a $n \times n$ mesh..* $\square$

## 1.2 Implementation on the PRAM

Consider an imput matrix $A$ of size $n \times n$. Note that in our basic one-sided SVD algorithm a basic step consists of orthagonalizing a pair of columns $i$ and $j$. After applying each of these operations the original matrix changes. Let $M$ stand for the matrix that we have at any given stage in the algorithm. Orthagonalizing the columns $i$ and $j$ of $M$ can be thought of as premultiplying the matrix $M$ with another matrix $R_{i,j}$. $R_{i,j}$ is such that only four of the entries in this matrix are nonzero. This multiplication can be done in $O(n)$ time sequentially. If we have a PRAM, this can be done in $O(1)$ time using $n$ processors. The steps involved in each sweep of the algorithm and the corresponding parallel bounds are given below.

1. For every pair of columns perform a dot product computation. For a single pair, the dot product can be computed in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors. Thus this step can be completed in $O(\log n)$ time using a total of $\frac{n^3}{\log n}$ CREW PRAM processors.

2. Sort the $\binom{n}{2}$ dot product values and choose the top $\frac{\binom{n}{2}}{k}$ values. This can be done in $O(\log n)$ time using $n^2$ CREW PRAM processors.

3. For each of the pivot elements we have to compute a rotation matrix. For one element, this can be done in $O(1)$ time. Thus this step can be completed in $O(1)$ time using $n^2$ processors.

4. After having computed all the $\frac{\binom{n}{2}}{k}$ rotation matrices we have to multiply all of them. One possible way of doing this is to multiply each one of these matrices at a time with a cumulative product. Multiplying a rotation matrix with a cumulative produce matrix (that can be possibly full) can be done in $O(1)$ time using $n$ processors. Thus this step can be completed in $O(n^2)$ time using $n$ CREW PRAM processors. Alternatively, we can multiply all of these matrices like in a prefix computation. In this case each matrix multiplication can be between two possibly full matrices. Thus this step can be completed in $O(\log^2 n)$ time using $\frac{n^5}{\log n}$ processors.

We get the following Theorem:

**Theorem 1.1** *We can run the JTS algorithm in parallel such that: 1) The run time is $O(S \log^2 n)$ using $\frac{n^5}{\log n}$ CREW PRAM processors; or 2) The run time is $O(Sn^2)$ using $n$ CREW PRAM processors. Here $S$ is the number of sweeps.*

## 1.3 Implementation on the Mesh

We can run the JTS algorithm on the mesh as well. Here again there are four steps illustrated in the previous section (for the PRAM). In this case, step 1 can be completed in $O(n)$ time on an $n \times n$ mesh. Sorting in step 2 can also be completed in $O(n)$ time on an $n \times n$ mesh. Step 3 can be completed in $O(1)$ time. Step 4 can be completed in $O(n^2)$ time. As a result, we get the following Theorem:

**Theorem 1.2** *We can run the JTS algorithm on an $n \times n$ mesh in $O(Sn^2)$ time. Here $S$ is the number of sweeps.*

## References

[1] E. Horowitz, S. Sahni, and S. Rajasekaran, *Computer Algorithms*, Silicon Press, 2008.

[2] J. Já Já, *Parallel Algorithms: Design and Analysis*, Addison-Wesley Publishers, 1992.

[3] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan-Kaufmann Publishers, 1992.