A parallel algorithm for Gaussian elimination over finite fields

Stephen Linton, Gabriele Nebe, Alice Niemeyer, Richard Parker, Jon Thackray

March 7, 2017

Lehrstuhl D für Mathematik, RWTH Aachen University, 52056 Aachen, Germany

E-mail address: nebe@math.rwth-aachen.de

70 York St. Cambridge CB1 2PY, UK

E-mail address: richpark54@hotmail.co.uk

1 Introduction

Already employed for solving equations by hand in China over 2000 years ago [**?**], the Gaussian elimination method has become an invaluable tool in many areas of science. When computing machines became available, this algorithm was one of the first to be implemented on a computer. In general the algorithm takes as input a matrix defined over a ring and transforms this matrix to a matrix in row echelon form. It can be employed for several different purposes and computer implementations can be tailored to suit the intended application. Various different versions of the Gaussian elimination algorithm can be envisaged, namely computing the rank of a matrix, computing a row echelon form or a reduced row echelon form of a matrix, or computing one of these echelon forms together with transformation matrices. Often one of these versions of the Gaussian elimination algorithm lies at the heart for many other algorithms for solving problems in a broad range of areas and their overall performance is often dictated by the performance of the underlying Gaussian elimination algorithm. Thus an implementation of a Gaussian elimination algorithm is required to display exceptional performance.

Since their invention, computers have become faster and more powerful every year. Yet, for over a decade this increase in computing power is no longer due to faster CPUs but rather to the number of different processors an individual computer has paired with the amount of cache available to the processors, that is memory accessible to these processors at very high speed. It is therefore paramount that modern algorithms

are tailored to modern computers. In particular this means that they need to be able to perform computations in parallel and store the data for the current computations readily in cache.

With the advance of parallel computers comes the need to design a parallel algorithm to perform Gaussian elimination on a matrix. Such a parallel algorithm would immediately result in immense speedups of higher level algorithms calling the Gaussian elimination without having to introduce parallelism to these algorithms themselves. When designing a parallel Gaussian elimination algorithm it is important to keep the applications of the algorithms in mind. Several versions of a parallel Gaussian elimination algorithm have been described when working over the field of real or complex numbers, see for example [**?**]. In this paper we describe a parallel version of the Gauss elimination algorithm which, given a matrix with entries in a finite field, computes a reduced row echelon form together with the transformation matrices. We note that when working over a finite field we need not be concerned with numerical accuracy and the selection of suitable pivot elements. In particular, we can always choose the first non-zero element of a given row as our pivot element, ensuring that all entries to the left of a pivot are known to be zero. Moreover, we need not be concerned with producing elements in a field which require more and more memory to store them. Avoiding to produce very large field elements would again complicate pivot selection, making it more of an art than a science. Thus our main concern is to design a parallel algorithm which makes optimal use of modern parallel computers.

We assume an underlying shared memory computational model in which we have access to k different processors, each of which can run a job independently from any other. Moreover, all processors have access to two different internal memory locations called Cache 1 and Cache 2. We assume that the access to Cache 1 is extremely fast, whereas access to Cache 2 is slower, yet still faster than access to data stored on a hard disk. The aim of our parallel algorithm is to divide the necessary computational work into smaller jobs and schedule these to run on the k processors. In order to achieve real speed-ups in this manner we have to ensure that the data required to run these jobs is stored in Cache 1. This in turn calls for a very careful organisation of the jobs and the data so that different jobs do not interfere with each other by either requesting different data to be loaded into cache or by overwriting each others data. We will address these issues in Section 2.1.

We envisage that we are given a very large matrix over a finite field for which we need to compute a reduced row echelon form together with a transformation matrix. In fact, the matrix will be so large, that it does not fit into the computer's cache and has to be stored on hard disk. In order to deal with such matrices efficiently it is clear that they will have to be subdivided into smaller pieces of data. Several approaches to achieve this already exist for finite fields. Most previous work subdivides the input matrix by considering batches of full length rows. Mention Lübeck and ...

Here we take a different approach. We subdivide our very large input matrix into smaller submatrices, called *blocks*, by subdividing both rows and columns. While at

first glance this might seem less natural it has the advantage of being able to keep more rows (albeit with fewer columns) in cache memory so that one avoids the memory wall (What is this?).

While subdividing a very large matrix into smaller submatrices is essential to designing a parallel algorithm, the size of these blocks cannot become too small. In practice the block size needs to be large enough to tap into the speed gain produced by highly efficient parallel implementations of a matrix multiplication algorithm yet small enough to ensure we obtain sufficient speed up from running many jobs concurrently during the Gaussian elimination algorithm. Highly efficient parallel implementations of matrix multiplication over finite fields exist (see []).

Having chosen such a block size in such an optimal range, we can assume that a job receives as input a single block together with some additional data about the block. The algorithms we describe perform one of the jobs to be run concurrently and thus restrict attention to this one block. We thereby gain an improvement in the same order of magnitude as the number of blocks.

The parallel Gauss algorithm is designed with three distinct environments in view, although we have only implemented the first so far.

The first (and original) target is to use a single machine with multiple cores with the matrix in shared memory. Here the objective is to subdivide the overall task into many subtasks that can run concurrently.

The second target is to use a single machine where the matrix does not fit in memory, but does fit on disk. Here the objective is to subdivide the matrix so that each piece fits into memory.

The third target is where several computers (probably each with multiple cores) are to work on a single problem simultaneously. Here the objective is again to subdivide the overall task into many subtasks that can run concurrently on different computers with access to the same central disk.

2 Preliminaries

2.1 Computational Model

Our aim is to describe a parallel algorithm. Such an algorithm needs to be running many computations simultaneously. We achieve this by having a fixed number of independent *threads of execution*, or threads for short, each of which can run one job at any time. A module, called *thread farm*, is charged with scheduling the execution of the individual tasks. We found it simplified the description of the algorithms somewhat to collect related data items into *data packages*, for example $\mathbf{A}^j_{ik}, \mathbf{B}^i_{jk}$ etc. In our work, a task can start as soon its input data packages are ready. As some tasks may in turn consist of several jobs each of which may only require some elements of a data package, higher concurrency can be achieved by allowing such jobs to start when these elements are ready. The implementation in Meataxe64 [**?**] achieves this higher concurrency,

however for ease of exposition we omit these details. Similarly, some gain in efficiency can be achieved by giving the thread farm guidance as to which jobs are urgent and which are less so. This aspect also is ignored in this paper.

The parallel Gauss algorithm is described in Section 3 as a program called the *Chief* charged with defining the tasks and the data packages they work on. The Chief is described as a sequential program and the reader should be warned that the order of execution may bear little resemblance to this program. The Chief only manipulates data packages and calls procedures that we call tasks, for which the reader finds a more or less specific description in Section 4. The tasks themselves are specified in terms of jobs also described in Section 4.

For the parallelisation it is essential to know which input data is required for the task. We distinguish data packages referring to the same package for different times by adding superscripts to the name of the data packages. In practice, these are written to a new register, and freed by the thread farm, whenever all jobs having this data package as input are finished.

2.2 Gaussian elimination

This subsection describes the Gaussian elimination process in a way we hope is familiar to the reader, but in our notation. Given a matrix H with entries in a field $\mathbb{F}$ the output of the Gauss algorithm consists of matrices M, K and R and permutation matrices P_ρ and P_γ such that

$$\begin{pmatrix} M & 0 \\ K & 1 \end{pmatrix} P_\rho H P_\gamma = \begin{pmatrix} -1 & R \\ 0 & 0 \end{pmatrix}. \tag{1}$$

The matrices P_ρ and P_γ perform row, respectively column, permutations on the input matrix H such that the top left-hand part of the resulting matrix $P_\rho H P_\gamma$ is invertible (with inverse M) with the same rank as H. It should be noticed that the permutation matrices P_ρ and P_γ in Equation (1) are not uniquely defined. All that matters is that they put the pivotal rows and columns into the top left-hand corner of the echelonised matrix. We therefore choose to only specify the sets of row and column numbers containing pivotal elements. This means we have to apply a permutation to the rows during the course of the algorithm to ensure that our pivots remain located in columns with increasing indices. We formalise how we store these permutation matrices in the following definition.

Definition 2.1. *To a subset $\rho \subseteq \{1, \ldots, \alpha\}$ we associate two* 0/1 *matrices*

$$\rho \in \mathbb{F}^{|\rho| \times \alpha} \text{ and } \overline{\rho} \in \mathbb{F}^{(\alpha - |\rho|) \times \alpha}.$$

We call ρ the row-select *matrix and $\overline{\rho}$ the* row-nonselect *matrix associated to the set ρ. To a subset $\gamma \subseteq \{1, \ldots, \beta\}$ we associate two* 0/1 *matrices*

$$\gamma \in \mathbb{F}^{\beta \times |\gamma|} \text{ and } \overline{\gamma} \in \mathbb{F}^{\beta \times (\beta - |\gamma|)}$$

We call γ the column-select *matrix and $\overline{\gamma}$ the* column-nonselect *matrix associated to the set γ.*

We also note that for a matrix $H \in \mathbb{F}^{\alpha\times\beta}$ and $\rho \subseteq \{1,\ldots,\alpha\}$ and $\gamma \subseteq \{1,\ldots,\beta\}$ the matrix $\rho \times H \in \mathbb{F}^{|\rho|\times\beta}$ consists of those rows of H whose indices lie in ρ (retaining the ordering) and the matrix $H \times \gamma \in \mathbb{F}^{\alpha\times|\gamma|}$ consists of those columns of H whose indices lie in γ.

Remark 2.2. *Let $H \in \mathbb{F}^{\alpha\times\beta}$ be of rank r. Then the echelonisation algorithm will produce sets $\rho \subseteq \{1,\ldots,\alpha\}$ and $\gamma \subseteq \{1,\ldots,\beta\}$ of cardinality $|\gamma| = |\rho| = r$ and matrices $M \in \mathbb{F}^{r\times r}$, $K \in \mathbb{F}^{(\alpha-r)\times r}$, $R \in \mathbb{F}^{r\times(\beta-r)}$ such that*

$$\begin{pmatrix} M & 0 \\ K & 1 \end{pmatrix} \begin{pmatrix} \rho \\ \overline{\rho} \end{pmatrix} H \begin{pmatrix} \gamma & \overline{\gamma} \end{pmatrix} = \begin{pmatrix} -1 & R \\ 0 & 0 \end{pmatrix}$$

We will refer to these matrices as

$$(M, K, R, \rho, \gamma) := \mathrm{ECH}(H).$$

3 A parallel Gauss algorithm

Let $\mathbb{F}$ be a finite field and $C \in \mathbb{F}^{m\times n}$ a huge matrix of rank r. We describe a parallel version of the well-known GAUSS algorithm, which computes matrices R, a transformation matrix $\mathcal{T} = \begin{pmatrix} \mathcal{M} & 0_{r\times(m-r)} \\ \mathcal{K} & 1_{(m-r)\times(m-r)} \end{pmatrix}$ and subsets $\varrho \subseteq \{1,\ldots m\}$ and $\Upsilon \subseteq \{1,\ldots,n\}$, both of cardinality r, such that

$$\begin{pmatrix} \mathcal{M} & 0_{r\times(m-r)} \\ \mathcal{K} & 1_{(m-r)\times(m-r)} \end{pmatrix} \begin{pmatrix} \varrho \\ \overline{\varrho} \end{pmatrix} C \begin{pmatrix} \Upsilon & \overline{\Upsilon} \end{pmatrix} = \begin{pmatrix} -1_{r\times r} & R \\ 0 & 0 \end{pmatrix} \tag{2}$$

is in row echelon form.

Rather than calling the echelonisation algorithm ECH for our huge input matrix C we divide this matrix into smaller blocks. We choose positive integers a_i, b_j such that

$$\sum_{i=1}^{a} a_i = m, \ \sum_{j=1}^{b} b_j = n$$

and our algorithm copies the block-submatrices of the input matrix C into data packages $\mathbf{C}_{ij}.\mathrm{C} \in \mathbb{F}^{a_i\times b_j}$, called *blocks*, and performs tasks (as described in Section 4) on these smaller blocks.

We call the a matrices $\mathbf{C}_{1j}.\mathrm{C},\ldots,\mathbf{C}_{aj}.\mathrm{C}$ the j-th *block column* and the b matrices $\mathbf{C}_{i1}.\mathrm{C},\ldots,\mathbf{C}_{ib}.\mathrm{C}$ the i-th *block row* of C. Echelonising the overall matrix C thus amounts to performing an echelonisation algorithm on individual blocks and using the resulting blocks to modify others.

The result of the Gaussian elimination as well as the intermediate matrices are partitioned suitably into blocks: When the Gauss algorithm has completed, the matrix R, which occurs in Equation (2), consists of blocks and has the form

$$R = \begin{pmatrix} R_1 & Y'_{12} & \dots & \dots & Y'_{1b} \\ 0 & R_2 & Y'_{23} & \dots & Y'_{2b} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & R_{b-1} & Y'_{b-1,b} \\ 0 & \dots & \dots & 0 & R_b \end{pmatrix} \tag{3}$$

with $R_j \in \mathbb{F}^{r_j \times (b_j - r_j)}$ and $Y'_{jk} \in \mathbb{F}^{r_j \times (b_k - r_k)}$. Here $r = \sum_{j=1}^{b} r_j$ is the rank of C and for $k = 1, \dots, b$ the sum $\sum_{j=1}^{k} r_j$ is the rank of the submatrix of C consisting of the first k block columns.

The overall Gaussian elimination algorithm consists of two steps (called Step 1 and Step 3 for technical reasons). After the first step the matrix C has been transformed into an upper triangular matrix and prior to permuting columns the matrix $(-1_{r\times r}|R) \in \mathbb{F}^{r\times n}$ has the shape

$$\tilde{R} := \left(\begin{array}{ccccc} -1 \mid R_1 & X_{12} \mid Y_{12} & \dots & \dots & X_{1b} \mid Y_{1b} \\ 0 \mid 0 & -1 \mid R_2 & X_{23} \mid Y_{23} & \dots & X_{2b} \mid Y_{2b} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 \mid 0 & \dots & 0 \mid 0 & -1 \mid R_{b-1} & X_{b-1,b} \mid Y_{b-1,b} \\ 0 \mid 0 & \dots & \dots & 0 \mid 0 & -1 \mid R_b \end{array}\right), \tag{4}$$

where $Y_{jk} \in \mathbb{F}^{r_j \times (b_k - r_k)}$ and $X_{jk} \in \mathbb{F}^{r_j \times r_k}$. To simplify notation in the algorithms below we define the data packages $\mathbf{B}_{jk}.\mathrm{B} := (X_{jk}|Y_{jk}) \in \mathbb{F}^{r_j \times b_k}$ for $1 \le j < k \le b$ and $\mathbf{D}_j := (\mathbf{D}_j.\mathrm{R}, \mathbf{D}_j.\gamma)$, where $\mathbf{D}_j.\gamma \subseteq \{1, \dots, b_j\}$ is the set of indices of pivotal columns in the block column j.

During the course of the algorithm we also compute the transformation matrix $\mathcal{T} \in \mathrm{GL}_m(\mathbb{F})$ as given in Equation (2). The data packages $\mathbf{M}_{ji}.\mathcal{M}$ $(j = 1, \dots, b; i = 1, \dots, a)$ and $\mathbf{K}_{ih}.\mathcal{K}$ $(i, h = 1, \dots, a)$ record the matching status of the transformation matrix. If $\tilde{C}_{ik}$ denotes the i,k-block of the original input matrix C, then initially $\mathbf{C}_{ik}.\mathrm{C} = \tilde{C}_{ik}$ and $\mathbf{B}_{jk}.\mathrm{B}$ is empty. Likewise, initially $\mathbf{K}_{ih}.\mathcal{K}$ is the identity matrix if $i = h$ and the zero matrix otherwise and the matrix $\mathbf{M}_{ji}.\mathcal{M}$ is empty. The art of our algorithm design is that we only remember the relevant parts of both data packages $\mathbf{K}$ and $\mathbf{M}$, i.e. omit the columns and rows in the blocks that are zero or still unchanged from the identity matrix. To obtain the row select matrix we maintain further data packages $\mathbf{E}_{ij} = (\mathbf{E}_{ij}.\rho, \mathbf{E}_{ij}.\delta)$ $(1 \le i \le a, 1 \le j \le b)$ with $\mathbf{E}_{ij}.\rho \subseteq \{1, \dots, a_i\}$ is the set of indices of pivotal rows in block row i with pivot in some block column $1, \dots, j$.

During the algorithm, having handled block row i in Step 1, we have

$$\sum_{h=1}^{a} \mathbf{M}_{jh}.\mathcal{M} \times \mathbf{E}_{hi}.\rho \times \tilde{C}_{hk} = \mathbf{B}_{jk}.\mathrm{B}$$

and

$$\sum_{h=1}^{a} \mathbf{K}_{\ell h}.\mathcal{K} \times \overline{\mathbf{E}_{hi}.\rho} \times \tilde{C}_{hk} = \mathbf{C}_{\ell k}.\mathrm{C}.$$

In the end the column select matrix is the block diagonal matrix

$$\Upsilon = \mathrm{diag}(\mathbf{D}_1^a.\gamma, \ldots, \mathbf{D}_b^a.\gamma).$$

and the row select matrix is

$$\varrho = \mathrm{diag}(\mathbf{E}_{1b}.\rho, \ldots, \mathbf{E}_{ab}.\rho).$$

3.1 Step 1

Step 1 loops over the block rows. For the j-th column of the i-th block row, the algorithm calls Task ClearDown with the two data packages $\mathbf{C}_{ij}$ and $\mathbf{D}_j$ as input. Task ClearDown amalgamates the pivots in $\mathbf{D}_j.\gamma$ with the pivots in the matrix $\mathbf{C}_{ij}.\mathrm{C}$ which lie in column j to enlarge the set $\mathbf{D}_j.\gamma$ as well as a matrix of the echelon form $(-1 \mid \mathbf{D}_j.\mathrm{R})$ followed by 0 rows (up to column permutations which are remembered in $\mathbf{D}_j.\gamma$). Moreover, the task ClearDown records in its output data package $\mathbf{A}_{ij}$ the row operations performed. With the help of these data packages, the first step then propagates the same elementary row operations to the remaining blocks in block row i as well as to block row i of the transition matrix using Task UpdateRow.

Hence Step 1 assembles in the block row $\begin{pmatrix} 0 & \ldots & 0 & -1 | \mathbf{D}_j.\mathrm{R} & \mathbf{B}_{j,j+1}.\mathrm{B} & \ldots & \mathbf{B}_{jb}.\mathrm{B} \end{pmatrix}$ the rows of the original input matrix whose pivotal entries lie in block column j for $j \leq b$. These rows are then deleted from the data package $\mathbf{C}$. Thus having treated the j-th block column the matrix $\mathbf{C}$ contains no rows whose pivots lie in block columns $1, \ldots, ,$ and the rank of $\mathbf{C}$ is $r - \sum_{\ell=1}^{j} r_\ell$.

In particular, during the course of the entire algorithm the block rows $\mathbf{C}_{i,-}.\mathrm{C}$ contain fewer and fewer rows, whereas the number of rows of the block row $\mathbf{B}_{j,-}.\mathrm{B}$ increases accordingly. After completing the block column j the matrices $\mathbf{B}_{jk}.\mathrm{B}$ remain stable.

Similarly for the transformation matrix, the matrix $\mathbf{M}$ gains rows whereas $\mathbf{K}$ looses rows. However, things here are slightly more complicated due to the fact that we do not store the full transformation matrix. As we only store columns of $\mathbf{K}_{ih}.K$ that are not known to be 0 or columns of an identity matrix, we have to ensure that all these columns are present, when calling UpdateRow. The columns that are not yet present are precisely the columns that correspond to the positions of the pivot rows stored in $\mathbf{E}_{hj}.\delta$. If $i = h$ this means we need to insert into the correct positions columns of an identity matrix and if $i \neq h$ then columns of a zero matrix. This is the achieved by Task Augment.

3.2 Step 2

This intermediate step becomes necessary as we do only store the relevant parts of the transformation matrices $\mathbf{M}_{ji}.\mathcal{M}$. Before the upwards cleaning in Step 3 we need to

riffle in zero columns in $\mathbf{M}_{ji}.\mathcal{M}$ so that the number of columns in $\mathbf{M}_{ji}.\mathcal{M}$ is equal to the number of columns in $\mathbf{M}_{bi}.\mathcal{M}$ for all j.

3.3 Step 3

Then back cleaning only performs upwards row operations on the matrix from Equation (4) to eliminate the X_{jk}. The matrices Y_{jk} from Equation (4) are stored in the data packages $\mathbf{R}_{jk}$ in the algorithm CHIEF. Having cleaned block columns $b, \ldots, k-1$ the algorithm adds the X_{jk} multiple of block row k to block row j for all $j \leq k-1$ to clear block column k. The same row operations are performed on the relevant part $\mathbf{M}$ of the transformation matrix.

Algorithm 1: CHIEF

Input : $C = (C_{ik})_{i=1,\ldots,a,k=1,\ldots,b}$, where $C_{ik} \in \mathbb{F}^{a_i \times b_k}$.

Output: $R = (\mathbf{R}_{jk}^{k-j})_{j \leq k=1,\ldots,b}$, $\mathcal{M} = (\mathbf{M}_{jh}^{2a}.\mathcal{M})_{j=1,\ldots,b,h=1,\ldots,a}$,
$\mathcal{K} = (\mathbf{K}_{ih}^{(2a}.\mathcal{K})_{i,h=1,\ldots,a}$,
a row select matrix $\varrho \subseteq \{1, \ldots, m\}$, the concatenation of the
$\mathbf{E}_{ib}.\rho \subseteq \{1, \ldots, a_i\}$ $(i = 1, \ldots, a)$,
and a column select matrix $\Upsilon \subseteq \{1, \ldots, n\}$, the concatenation of the
$\mathbf{D}_j^a.\gamma \subseteq \{1, \ldots, b_j\}$ $(j = 1, \ldots, b)$, such that

$$\begin{pmatrix} \mathcal{M} & 0 \\ \mathcal{K} & 1 \end{pmatrix} \begin{pmatrix} \varrho \\ \overline{\varrho} \end{pmatrix} C \begin{pmatrix} \Upsilon & \overline{\Upsilon} \end{pmatrix} = \begin{pmatrix} -1 & R \\ 0 & 0 \end{pmatrix}$$

Algorithm:

INITIALISATION OF DATA PACKAGES:
for *k from 1 to b* **do**
  for *i from 1 to a* **do**
    $\mathbf{C}_{ik}^1.\mathrm{C} := C_{ik}$;
    $\mathbf{M}_{ki}^0.\mathcal{M} := 0$;
  end
  $\mathbf{D}_k^0.\mathrm{R} := 0$; $\mathbf{D}_k^0.\gamma := \{\}$;
  for *j from* $k+1$ *to b* **do**
    $\mathbf{B}_{kj}^0.\mathrm{B} := 0$;
  end
end
for *i from 1 to a* **do**
  for *h from 1 to a* **do**
    $\mathbf{K}_{ih}^0.\mathcal{K} := 0$;
  end
end

Algorithm 1: CHIEF (continued)

STEP 1:
for *i from 1 to a* **do**
  for *j from 1 to b* **do**
    $(\mathbf{D}^i_j; \mathbf{A}_{ij}) :=$ CLEARDOWN$(\mathbf{C}^j_{ij}, \mathbf{D}^{i-1}_j)$;
    $\mathbf{E}_{ij} :=$EXTEND$(\mathbf{A}_{ij}, \mathbf{E}_{i,j-1})$;
    for *k from* $j+1$ *to b* **do**
      $(\mathbf{C}^{j+1}_{ik}, \mathbf{B}^i_{jk}) :=$ UPDATEROW$(\mathbf{A}_{ij}, \mathbf{C}^j_{ik}, \mathbf{B}^{i-1}_{jk})$;
    end
    for *h from* 1 *to i* **do**
      $\mathbf{K}^{2j-1}_{ih} :=$AUGMENT$(\mathbf{K}^{2(j-1)}_{ih}, \mathbf{E}_{hj}, i = h)$;
      $(\mathbf{K}^{2j}_{ih}, \mathbf{M}^i_{jh}) :=$ UPDATEROW$(\mathbf{A}_{ij}, \mathbf{K}^{2j-1}_{ih}, \mathbf{M}^{i-1}_{jh})$;
    end
  end
end
STEP 2:
for *j from 1 to b* **do**
  for *h from 1 to a* **do**
    $\mathbf{M}^{a+1}_{jh} :=$ROWLENGTHEN$(\mathbf{M}^a_{jh}, \mathbf{E}_{hj}, \mathbf{E}_{hb})$;
  end
end
STEP 3:
for *k from* 1 *to b* **do**
  $\mathbf{R}^0_{kk} :=$COPY$(\mathbf{D}^a_k)$;
end
for *k from b downto* 1 **do**
  for *j from 1 to* $k-1$ **do**
    $(\mathbf{X}_{jk}, \mathbf{R}^0_{jk}) :=$ PRECLEARUP$(\mathbf{B}^a_{jk}, \mathbf{D}^a_k)$;
    for *h from k to b* **do**
      $\mathbf{R}^{h-k+1}_{jh} :=$ CLEARUP$(\mathbf{R}^{h-k}_{jh}, \mathbf{X}_{jk}, \mathbf{R}^{h-k}_{kh})$;
    end
    for *h from 1 to a* **do**
      $\mathbf{M}^{a+h}_{jh} :=$ CLEARUP$(\mathbf{M}^{a+h-1}_{jh}, \mathbf{X}_{jk}, \mathbf{M}^{a+h-1}_{kh})$;
    end
  end
end

4 Jobs and Tasks

4.1 The jobs

In this section we describe the jobs. These are fundamental algorithms that are later used to define the tasks. Many of the jobs take as input one or more matrices. While the input and output matrices of the jobs within the global context of the parallel Gauss algorithm are blocks computed from a huge input matrix, the jobs described in this section work locally only on these matrices.

MUL This job performs a matrix **mul**tiplication. It takes as input two matrices $A \in \mathbb{F}^{\alpha\times\beta}$ and $B \in \mathbb{F}^{\beta\times\delta}$ and returns as output the matrix $A \times B \in \mathbb{F}^{\alpha\times\delta}$.

MAD This job performs a matrix **m**ultiplication followed by a matrix **ad**dition. It takes as input matrices $A \in \mathbb{F}^{\alpha\times\delta}$ and $B \in \mathbb{F}^{\alpha\times\beta}$ and $C \in \mathbb{F}^{\beta\times\delta}$ and returns the matrix $A + B \times C \in \mathbb{F}^{\alpha\times\delta}$.

CEX This job performs two **c**olumn **ex**tracts. It takes as input a matrix $H \in \mathbb{F}^{\alpha\times\beta}$ and a subset $\gamma \subseteq \{1, \ldots, \beta\}$ and returns the matrices $H \times \gamma$ and $H \times \overline{\gamma}$, consisting of all those columns of H whose indices lie in γ, respectively do not lie in γ, as described in Definition 2.1.

REX This job performs two **r**ow **ex**tracts. It takes as input a matrix $H \in \mathbb{F}^{\alpha\times\beta}$ and a subset $\rho \subseteq \{1, \ldots, \alpha\}$ and returns the matrices $\rho \times H$ and $\overline{\rho} \times H$, consisting of all those rows of H whose indices lie in ρ, respectively do not lie in ρ, as described in Definition 2.1.

ECH This job performs an **ech**elonisation as described in Remark 2.2. We will refer to the job as

$$(M, K, R, \rho, \gamma) := \mathrm{ECH}(H).$$

PVC This job takes as input two disjoint subsets $\rho_1, \rho_2 \subseteq \{1, \ldots, \alpha\}$ and returns a subset $\rho \subseteq \{1, \ldots, \alpha\}$ defined as follows. $\rho = \rho_1 \cup \rho_2 =: \{x_1, \ldots, x_r\} \subseteq \{1, \ldots, \alpha\}$ with $x_k < x_\ell$ for $k < \ell$ and records in $u \in \{0, 1\}^r$ whether $x_\ell \in \rho_1$ by setting $u_\ell = 0$ or $x_\ell \in \rho_2$ by setting $u_\ell = 1$. We refer to this job as

$$(\rho, u) := \mathrm{PVC}(\rho_1, \rho_2).$$

RRF This job performs a row riffle. The input consists of a bit string $u \in \{0, 1\}^r$ and two matrices $B \in \mathbb{F}^{\alpha\times\beta}$ and $C \in \mathbb{F}^{\gamma\times\beta}$ with $\alpha + \gamma = r$, where the number of 0s in u is α and the number of 1s in u is γ. The job returns the new matrix $A \in \mathbb{F}^{r\times b}$ whose rows are the rows of B and C combined according to u. In some sense this is the inverse of row extract.

CRZ Similarly to row riffles we also need column riffles, but we only need to riffle in the zero matrix (CRZ) or the identity matrix (CRI).

4.2 The tasks

We now describe the tasks on which our Gaussian elimination algorithm depends. As mentioned above, a task receives data packages as input and returns data packages as output. A task can start, as soon as its input data packages are ready.

Task 1: EXTEND

Input : $\mathbf{A} := (A, M, K, \rho', E, \lambda)$, $\mathbf{E} = (\rho, \delta)$ with $\rho \subset \{1, \ldots, \alpha\}$, δ a riffle.
Output: $\mathbf{E}$.

(PVC): $(\mathbf{E}.\rho, \mathbf{E}.\delta) := \mathrm{PVC}(\mathbf{E}.\rho, \mathbf{A}.\rho')$;

Task 2: AUGMENT

Input : $\mathbf{K} \in \mathbb{F}^{\alpha\times\beta}$, $\mathbf{E} = (\rho, \delta)$ and a flag.
Output: $Y \in \mathbb{F}^{\alpha\times\beta'}$, where $\beta' = \beta + z$ and $z = |\delta^{-1}(\{1\})|$.

if *flag = false* **then**
| (CRZ): $Y :=$ column riffle of $\mathbf{K}$ and $0_{\alpha\times z}$ using $\mathbf{E}.\delta$
end
if *flag = true* **then**
| (CRI): $Y :=$ column riffle of $\mathbf{K}$ and $I_\alpha \times \delta^{-1}(\{1\})$ using $\mathbf{E}.\delta$
end

Task 3: ROWLENGTHEN

Input : $\mathbf{M} \in \mathbb{F}^{\alpha\times g_1}$, $(\mathbf{E}_1).\rho \subseteq (\mathbf{E}_2).\rho \subseteq \{1, \ldots, \alpha\}$ of cardinalities say g_1, g_2 with $g_1 \leq g_2$.
Output: $\mathbf{M} \in \mathbb{F}^{\alpha\times g_2}$.

(MKR): $\lambda \subseteq \{0,1\}^{g_2}$ with 0s corresponding to the elements in $\mathbf{E}_1.\rho$ and 1s corresponding to the elements in $\mathbf{E}_2.\rho \backslash \mathbf{E}_1.\rho$.;
(CRZ): $\mathbf{M} :=$ column riffle of $\mathbf{M}$ and 0 using λ;

Task 4: CLEARUP

Input : $\mathbf{R} \in \mathbb{F}^{\alpha\times\beta}$, $\mathbf{X} \in \mathbb{F}^{\alpha\times\gamma}$, $\mathbf{M} \in \mathbb{F}^{\gamma\times\beta}$.
Output: $\mathbf{R} \in \mathbb{F}^{\alpha\times\beta}$.

(MAD): $\mathbf{R} := \mathbf{R} + \mathbf{X} \times \mathbf{M}$;

Task 5: PRECLEARUP

Input : $\mathbf{B} \in \mathbb{F}^{\alpha\times\beta}$, $\mathbf{D}$, with $\mathbf{D}.\gamma \subseteq \{1, \ldots, \beta\}$ of cardinality g.
Output: $\mathbf{X} \in \mathbb{F}^{\alpha\times g}$, $\mathbf{R} \in \mathbb{F}^{\alpha\times(\beta-g)}$.

(CEX): $\mathbf{X} := \mathbf{B} \times \mathbf{D}.\gamma$; $\mathbf{R} := \mathbf{B} \times \overline{\mathbf{D}.\gamma}$;

Task 6: COPY

Input : **D**, with **D**.R $\in \mathbb{F}^{\alpha\times\beta}$.
Output: **R** $\in \mathbb{F}^{\alpha\times\beta}$.

(CPY): **R** := **D**.R;

4.2.1 Task CLEARDOWN

Task CLEARDOWN works on block columns. Suppose that $j \in \{1,\ldots,b\}$ and CLEARDOWN works on block column j which contains b_j columns. Task CLEARDOWN assumes that block column j truncated after row $i-1$ is in row echelon form and the aim of task CLEARDOWN is to replace the block column j truncated after row i by its row echelon form.

Task CLEARDOWN takes two data packages **C** and **D** as input. The first data package **C** is the block $\mathbf{C}_{ij}$.C which is the block in the i-th block row of block column j. The second data set **D** contains two data elements. The data element **D**.R is a matrix such that block column j truncated after block row $i-1$ is in row echelon form $(-1 \mid \mathbf{D}.\mathrm{R})$ followed by 0 rows. The data element $\mathbf{D}.\gamma \subseteq \{1,\ldots,b_j\}$ contains indices of the pivots assembled in block column j truncated after block row $i-1$.

The task produces two data packages **A** and **D** as outputs. The data elements stored in the data package **A** are required to propagate row operations performed during the call to Task UPDATEROW to other blocks in block row i. The data elements stored in the data package **D** are required for a subsequent call to CLEARDOWN for the block $\mathbf{C}_{i+1,j}$ in block column j.

We begin by partitioning the input block **C** according to $\mathbf{D}.\gamma$ into pivotal and non pivotal columns $\mathbf{C} = (\mathbf{A}.\mathrm{A} \mid A')$. Using the rows of the matrix $(-1 \mid \mathbf{D}.\mathrm{R})$ we can reduce **C** to $(0 \mid H')$ where $H' = A' + \mathbf{A}.\mathrm{A} \times \mathbf{D}.\mathrm{R}$. The next step is to call job ECH to echelonise H' and obtain

$$(\mathbf{A}.\mathrm{M}, \mathbf{A}.\mathrm{K}, R, \mathbf{A}.\rho', \gamma') := \mathrm{ECH}(H'),$$

where $\mathbf{A}.\rho'$ is the set of pivotal rows of H' and γ' the set of pivotal columns.

As block column j truncated after block row $i-1$ is in row echelon form $(-1_{r\times r} \mid \mathbf{D}.\mathrm{R})$ followed by 0 rows, we now wish to determine a new remnant matrix $\hat{R}$ (which will become the new **D**.R) such that block column j truncated after block row i is in row echelon form $(-1_{(r+r')\times(r+r')} \mid \hat{R})$ followed by 0 rows. To achieve this, the we have to add the r' pivots of H' to $-1_{r\times r}$ and reduce **D**.R according to $(-1_{r'\times r'} \mid R)$. This amounts to first separating the columns of **D**.R into those containing pivot entries of H' and those that do not, i.e. writing $\mathbf{D}.\mathrm{R} = (\mathbf{A}.\mathrm{E} \mid R')$ with the help of the row select and row non-select matrices γ' and $\overline{\gamma'}$. We then use the rows of the matrix $(-1_{r'\times r'} \mid R)$ to reduce **D**.R to $(0 \mid R' + \mathbf{A}.\mathrm{E} \times \mathbf{D}.\mathrm{R})$. The new set $\mathbf{D}.\gamma$ of all pivotal columns of block column j truncated after block row i is now obtained by combining the old set $\mathbf{D}.\gamma$ and γ'. We record in $\mathbf{A}.\lambda$ the information which of these indices came

from the r' pivots of H'. Finally, the new remnant $\hat{R}$ is obtained by interleaving the rows of $R' + \mathbf{A}.\mathrm{E} \times R$ with the rows of R according to $\mathbf{A}.\lambda$ and storing the resulting matrix as the new $\mathbf{D}.\mathrm{R}$.

The following pseudo code details Task ClearDown:

Task 7: ClearDown

Input : $\mathbf{C} \in \mathbb{F}^{\alpha \times \beta}$, $\mathbf{D}.\gamma \subseteq \{1, \ldots, \beta\}$ of cardinality r, $\mathbf{D}.\mathrm{R} \in \mathbb{F}^{r \times (\beta - r)}$;

Output: $\mathbf{D}.\mathrm{R} \in \mathbb{F}^{(r+r') \times (\beta - r - r')}$, $\mathbf{D}.\gamma \subseteq \{1, \ldots, \beta\}$ of cardinality $r + r'$ and $\mathbf{A} = (A, M, K, \rho', E, \lambda)$ where $A \in \mathbb{F}^{\alpha \times r}$, $M \in \mathbb{F}^{r' \times r'}$, $E \in \mathbb{F}^{r \times r'}$, $K \in \mathbb{F}^{(\alpha - r') \times r'}$, $\rho' \subseteq \{1, \ldots, \alpha - r\}$ of cardinality r', $\lambda \in \{0,1\}^{r+r'}$.

(CEX): $\mathbf{A}.\mathrm{A} := \mathbf{C} \times \mathbf{D}.\gamma$; $A' := \mathbf{C} \times \overline{\mathbf{D}.\gamma}$;
(MAD): $H := A' + \mathbf{A}.\mathrm{A} \times \mathbf{D}.\mathrm{R}$;
(ECH): $(\mathbf{A}.\mathrm{M}, \mathbf{A}.\mathrm{K}, R, \mathbf{A}.\rho', \gamma') := \mathrm{ECH}(H)$;
(CEX): $\mathbf{A}.\mathrm{E} := \mathbf{D}.\mathrm{R} \times \gamma'$, $R' := \mathbf{D}.\mathrm{R} \times \overline{\gamma'}$;
(MAD): $R' := R' + \mathbf{A}.\mathrm{E} \times R$;
(PVC): $(\mathbf{D}.\gamma, \mathbf{A}.\lambda) := \mathrm{PVC}(\mathbf{D}.\gamma, \gamma')$;
(RRF): $\mathbf{D}.\mathrm{R} := \mathrm{RRF}(\mathbf{A}.\lambda, R', R)$;

4.2.2 Task UpdateRow

Given $i \in \{1, \ldots, a\}$, the Task UpdateRow works on block $\mathbf{C} = \mathbf{C}_{ik}.\mathrm{C}$ in block row i and block column k. It takes as input data packages $\mathbf{A}$, $\mathbf{C}$ and $\mathbf{B}$, where the data package $\mathbf{A}$ encodes the necessary information computed by ClearDown when transforming an earlier block in the same block row i into echelon form.

The same row operations that were performed on this earlier block now need to be performed on $\mathbf{C}$. This subroutine also assembles in the matrix $\mathbf{B}$ the rows in block column k whose pivotal entry lies in block column j for $j + 1 \leq k \leq b$. The new data package $\mathbf{C}$ returned by Tasks UpdateRow then is equal to the transformed input matrix $\mathbf{C}$ with these rows deleted.

The following pseudo code details Task UpdateRow:

Task 8: UpdateRow

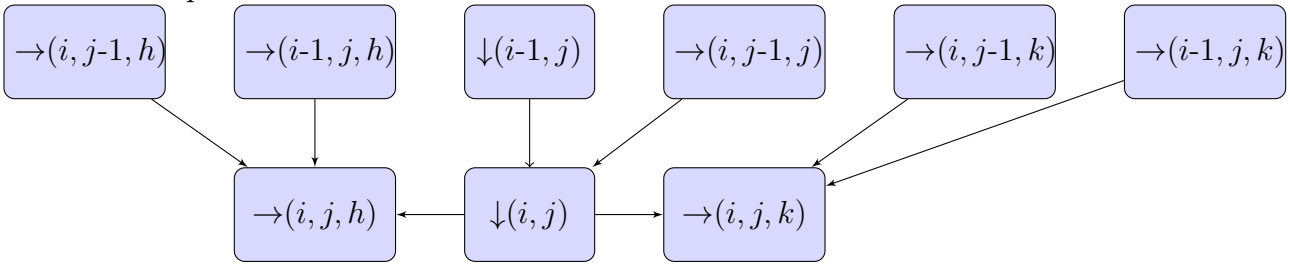
Input : $\mathbf{A} := (A, M, K, \rho', E, \lambda)$, $\mathbf{C} \in \mathbb{F}^{\alpha\times\beta}$, $\mathbf{B} \in \mathbb{F}^{r\times\beta}$.
Output: $\mathbf{C} \in \mathbb{F}^{(\alpha-r')\times\beta}$, $\mathbf{B} \in \mathbb{F}^{(r+r')\times\beta}$.

if $\mathbf{B}$ *equals NULL* **then**
  (REX): $V := \mathbf{A}.\rho' \times \mathbf{C}$; and $W := \overline{\mathbf{A}.\rho'} \times \mathbf{C}$;
  (MUL): $\mathbf{B} := \mathbf{A}.\mathrm{M} \times V$;
else
  (MAD): $Z := \mathbf{C} + \mathbf{A}.\mathrm{A} \times \mathbf{B}$;
  (REX): $V := \mathbf{A}.\rho' \times Z$; and $W := \overline{\mathbf{A}.\rho'} \times Z$;
  (MUL): $X := \mathbf{A}.\mathrm{M} \times V$;
  (MAD): $S := \mathbf{B} + \mathbf{A}.\mathrm{E} \times X$;
  (RRF): $\mathbf{B} := \mathrm{RRF}(\lambda, S, X)$;
end
(MAD): $\mathbf{C} := W + \mathbf{A}.\mathrm{K} \times V$;

5 Concurrency analysis

To measure the degree of concurrency of our algorithm we assign costs to each of the tasks. To the tasks Extend, Augment, RowLengthen, PreClearUp, and Copy which to not perform any time consuming operations we assign cost 0. The cost of a block matrix multiplication (possibly followed by an addition) of a block matrix of size $\alpha\times\beta$ is assumed to be $\mu = \alpha^2\beta$ and for the echelonisation (with transformation matrix) $2\alpha^2(\alpha+\beta) =: \mu'$. Then the cost of task ClearUp is μ, the cost of ClearDown is $2\mu+\mu'$ and the one of task UpdateRow is 4μ. We abbreviate the call to ClearDown with data packages depending on i, j by $\downarrow(i, j)$ and similarly UpdateRow by $\rightarrow(i, j, k)$ and ClearUp by $\uparrow(j, h, k)$.

Ignoring all tasks of cost 0 Step 1 only involves the tasks ClearDown and UpdateRow. The graph of task dependencies decomposes naturally into layers according to the value of $i+j$. The following picture displays the local task dependencies in Step 1 for layers $i + j - 1$ and $i + j$, where $k = j + 1, \ldots, b$ and $h = 1, \ldots, i$. The $\rightarrow$ with parameter h are performed on the transformation matrix.



Recall that a critical path in a task dependency graph is the longest directed path between any pair of start node and finish node. Its length is weighted by the cost of the nodes along the path. The following theorem is apparent from the structure of the

task dependency graph.

Theorem 5.1. *The weighted length of a critical path in the task dependency graph of Step 1 is*

$$(a+b)(4\mu + (2\mu + \mu')).$$

Step 3 only involves the task ClearUp of non-zero cost μ. The data package $\mathbf{R}_{jh}$ is only changed by the data packages below in the same column so $h-j$ times. Therefore the weighted length of a critical path in Step 3 is $\max((b-1)\mu, a\mu)$.

To determine the average degree of concurrency we divide the cost of the sequential Gaussalgorithm (with transformation matrix) applied to the $m \times n$-matrix C (for simplicity assumed to be $2m^2(n+m)$) by the weighted length of a critical path. For simplicity we assume that $m = n$, $a = b$ and that all blocks are of the same size $\alpha \times \beta$ with $\alpha = \beta$. In particular $\mu' = 4\mu$.

Remark 5.2. *Under the assumptions above the average degree of concurrency of algorithm* Chief *is* $\frac{4}{21}a^2$. *To achieve this degree of concurrency we need* $\binom{a}{2}$ *processors.*