# GPU Computing and Accelerators: Part V

# Open Computing Language (OpenCL)

## Main Message

The abstraction for the programming and hardware models are very similar to the CUDA concepts. Mainly OpenCL delivers slightly more flexible implementations due to vendor independence and uses slightly different vocabulary for the single ingredients of the concept.

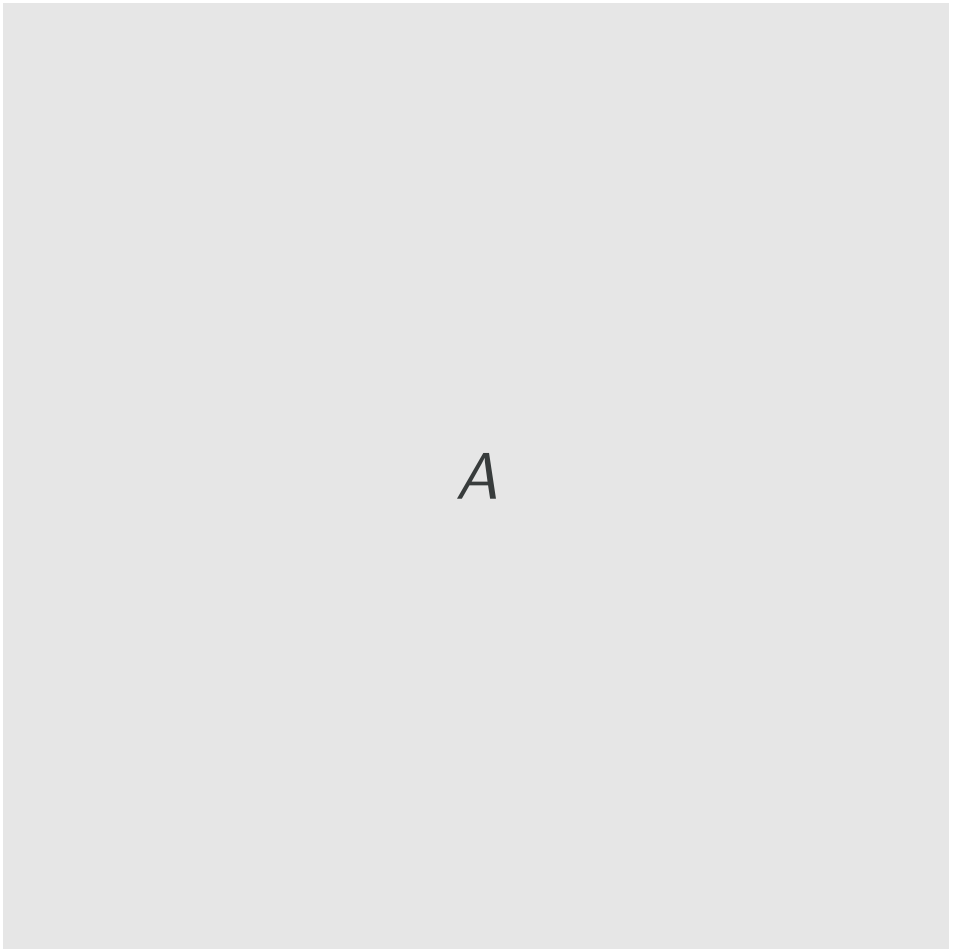
| CUDA | OpenCL |
|---|---|
| thread | (Work) item |
| block | (Work) group |
| streaming multiprocessor | compute unit |
| (CUDA) processor | processing unit |

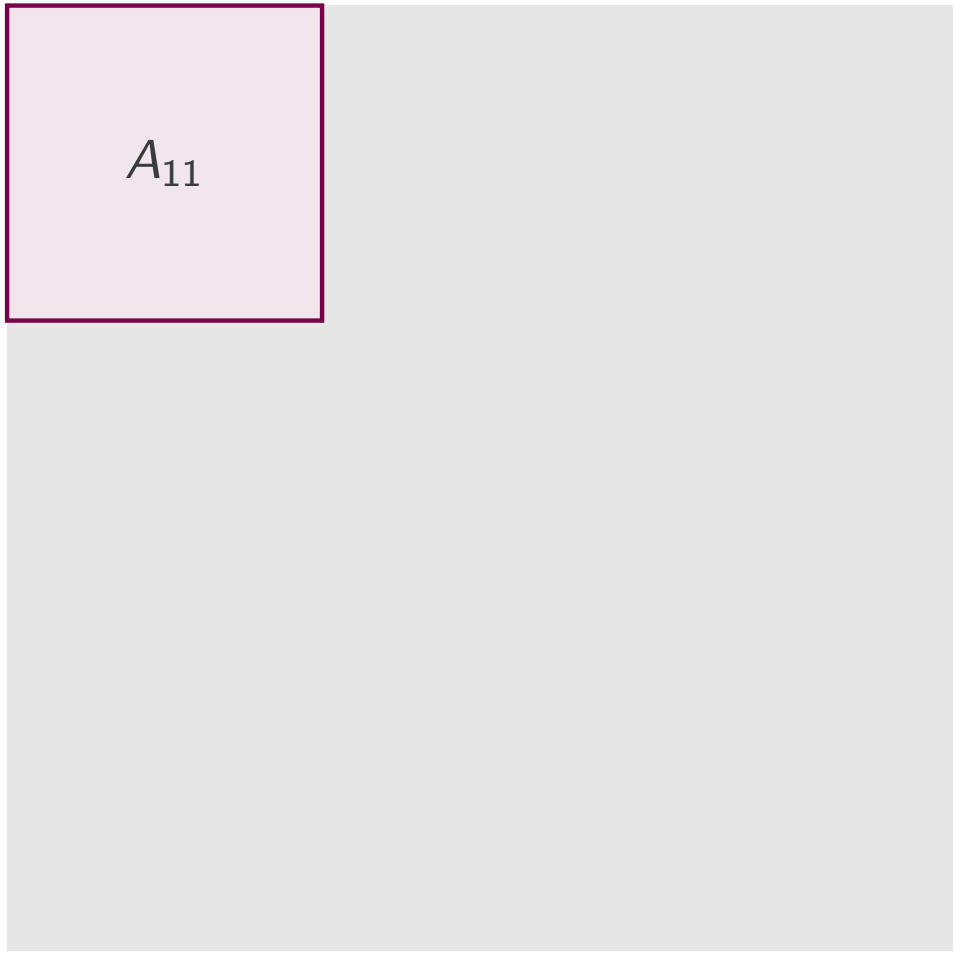
Table: A short CUDA to OpenCL dictionary

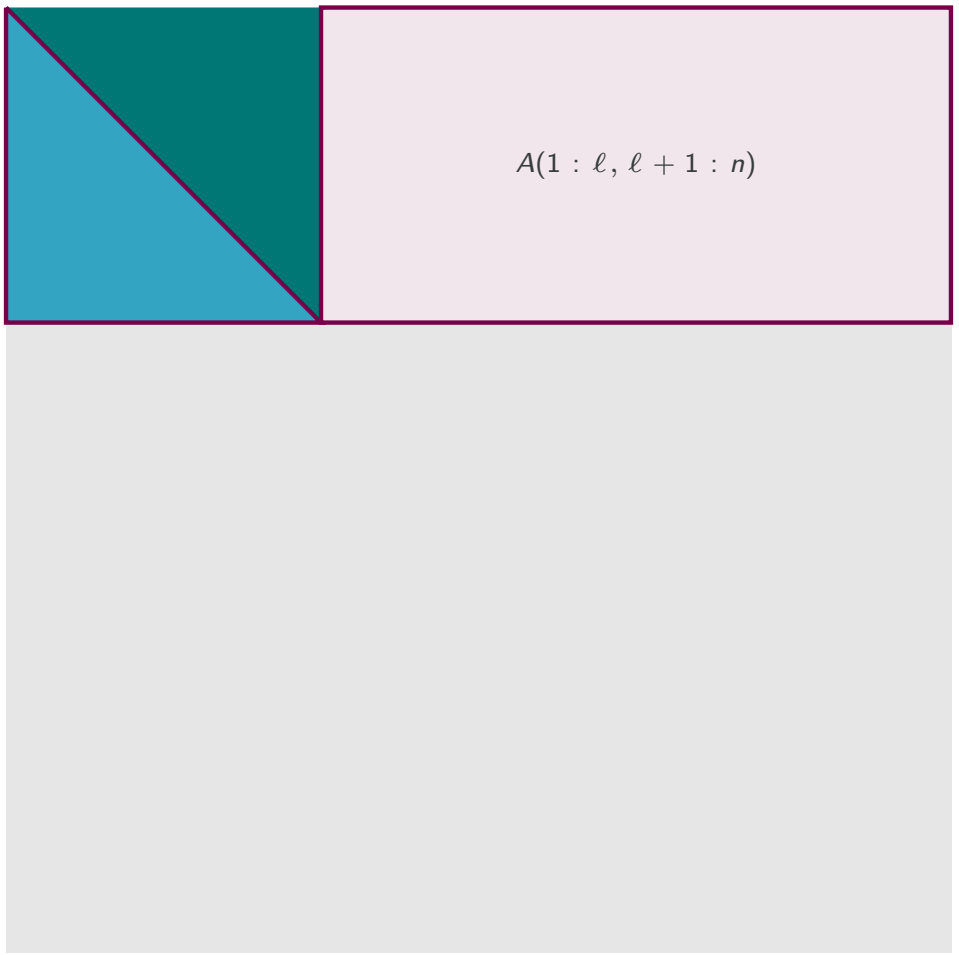
**Algorithm 6:** Gaussian elimination – Block outer product formulation

**Input**: $A \in \mathbb{R}^{n \times n}$ allowing LU decomposition, $r$ prescribed block size
**Output**: $A = LU$ with $L, U$ stored in $A$

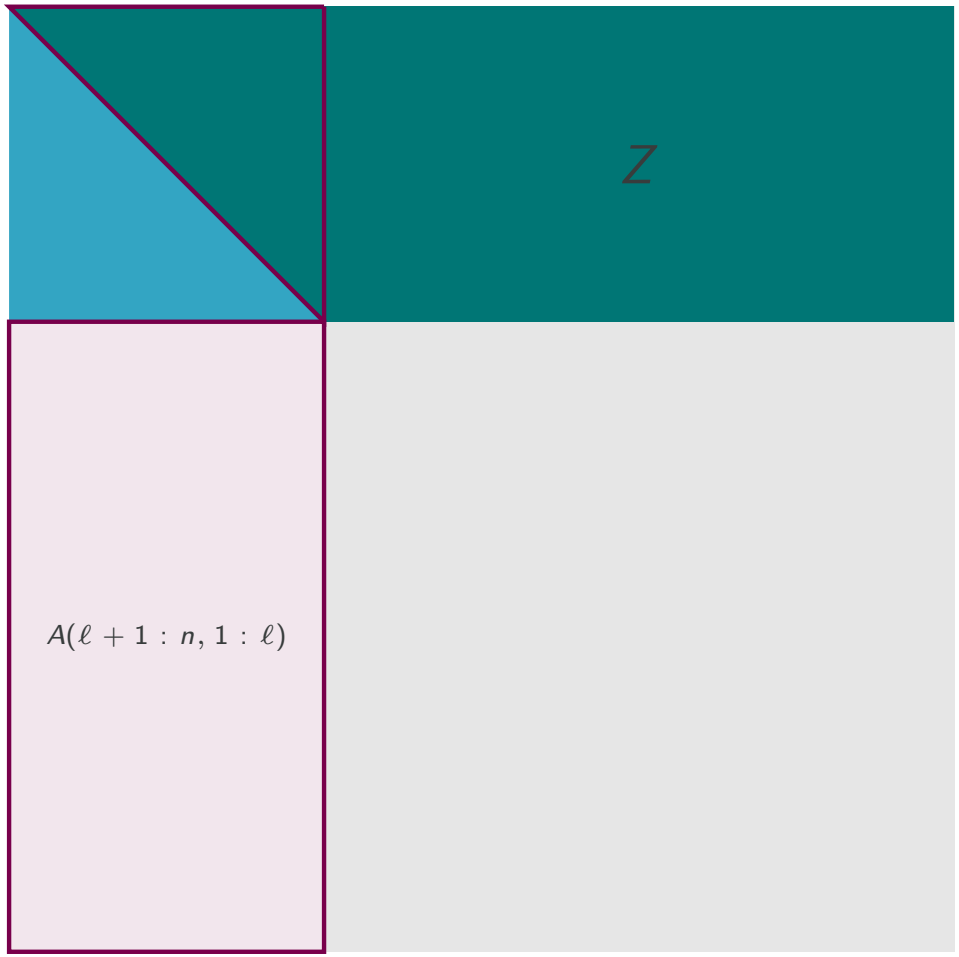
$k = 1$;
**while** $k \leq n$ **do**
  $\ell = \min(n, k + r - 1)$;
  Compute $A(k : \ell, k : \ell) = \tilde{L}\tilde{U}$ via Algorithm 7;
  Solve $\tilde{L}Z = A(k : \ell, \ell + 1 : n)$ and store $Z$ in $A$;
  Solve $W\tilde{U} = A(\ell + 1 : n, k : \ell)$ and store $W$ in $A$;
  Perform the rank-r update:
  $A(\ell + 1 : n, \ell + 1 : n) = A(\ell + 1 : n, \ell + 1 : n) - WZ$;
  $k = \ell + 1$;

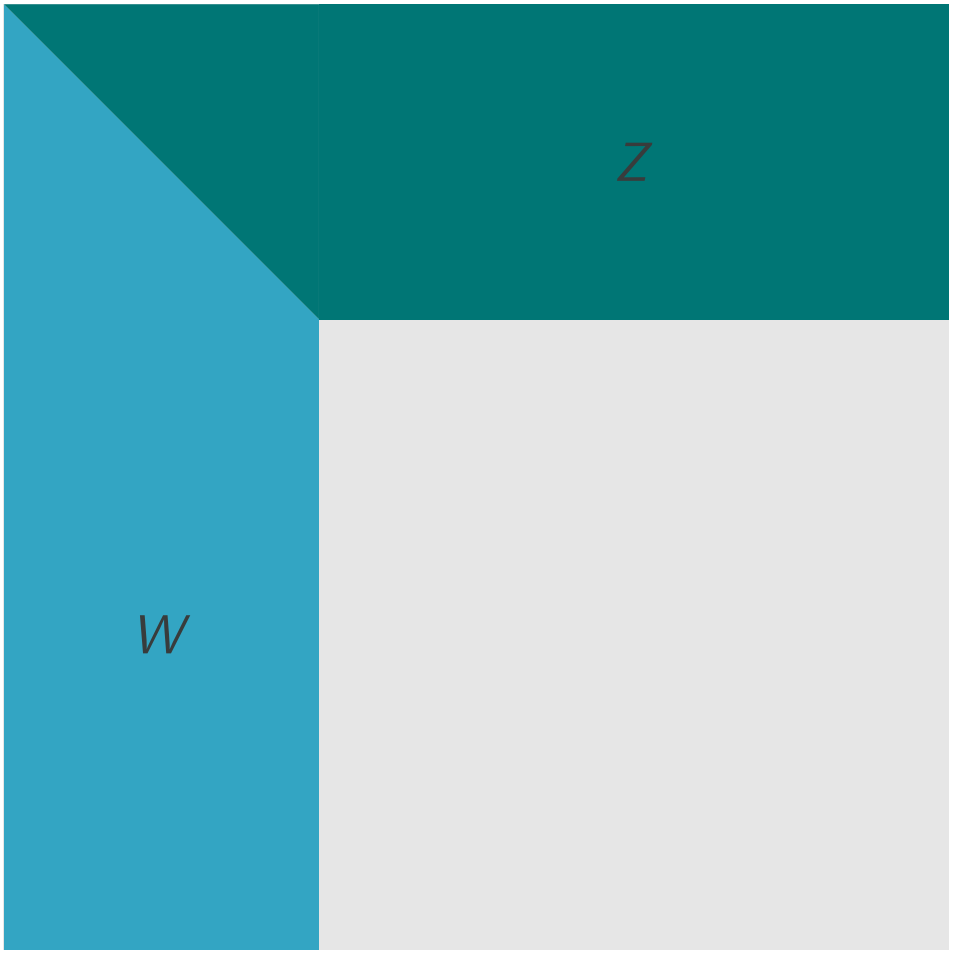

A

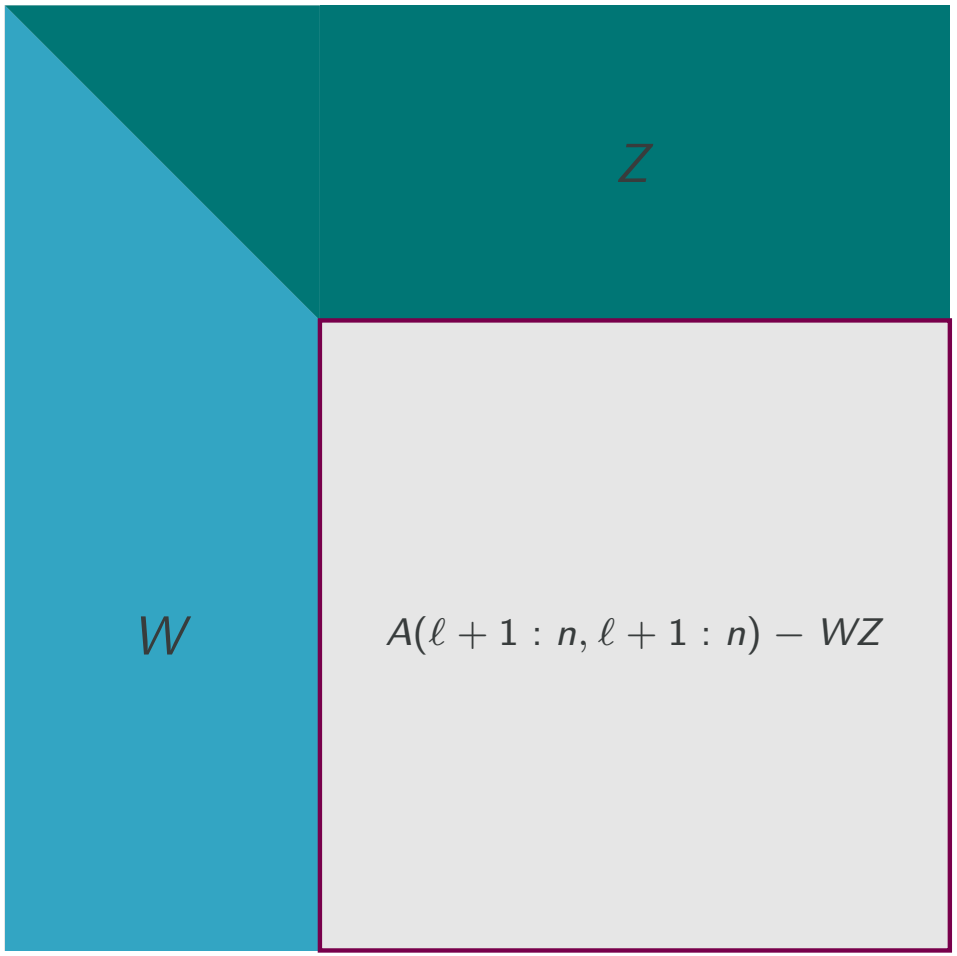

$A_{11}$

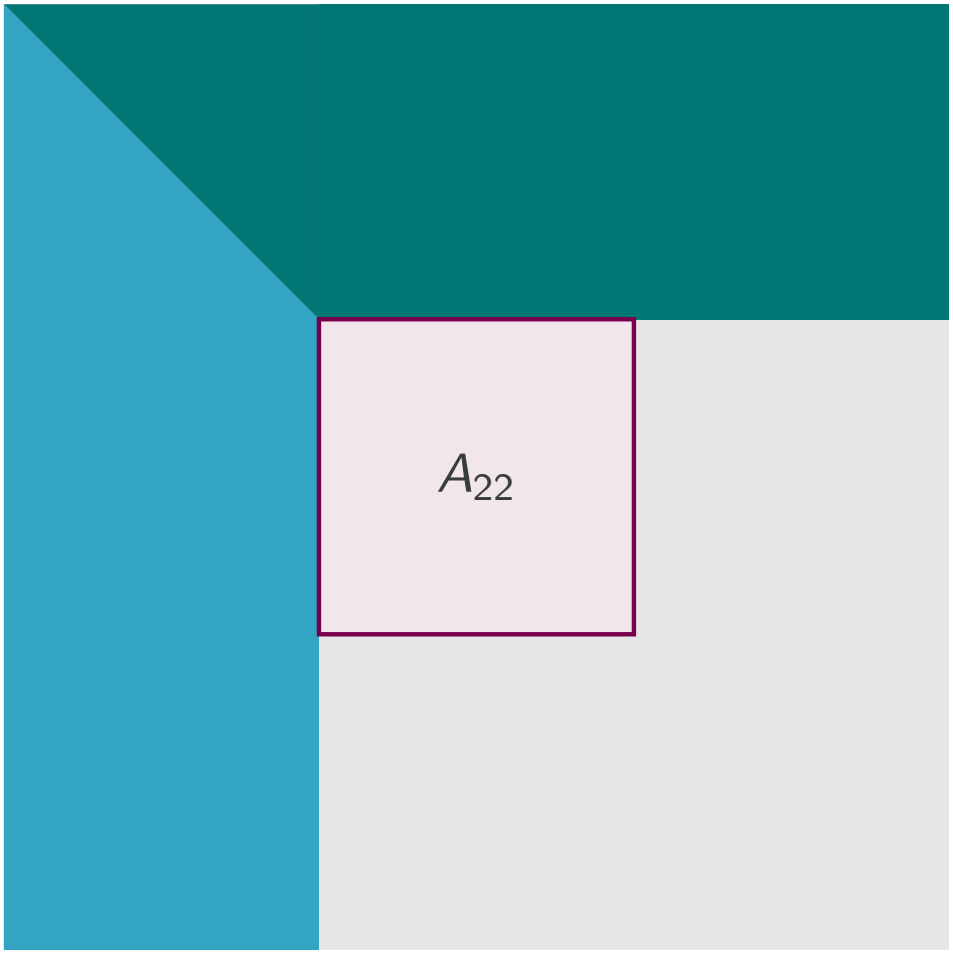

$A(1:\ell, \ell+1:n)$


Z


Z
$A(\ell+1:n, 1:\ell)$


Z
W


Z
W
$A(\ell+1:n,\ell+1:n)-WZ$


$A_{22}$

# Hybrid CPU-GPU Linear System Solvers

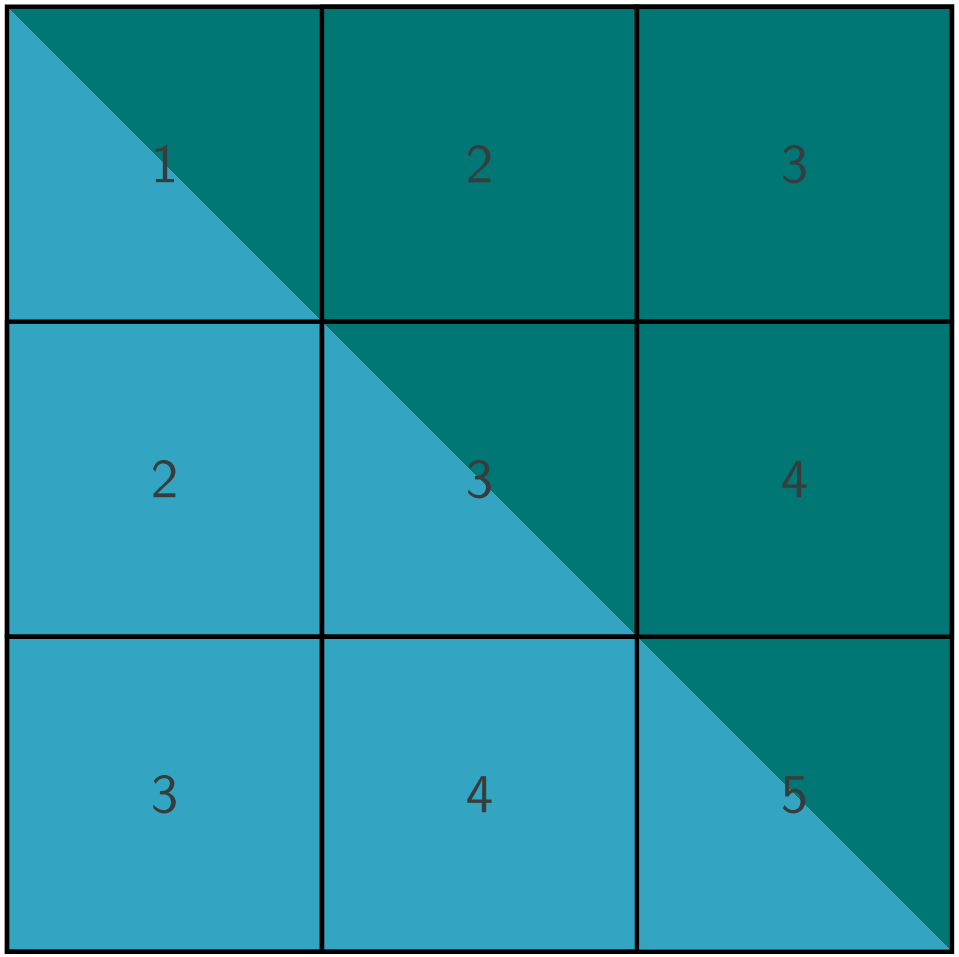
## The block outer product LU decomposition revisited

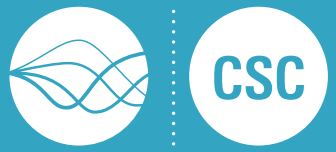
# Hybrid CPU-GPU Linear System Solvers

## The block outer product LU decomposition revisited

# Hybrid CPU-GPU Linear System Solvers

## The block outer product LU decomposition revisited


1
2
3
2
3
4
3
4
5



# Hybrid CPU-GPU Linear System Solvers

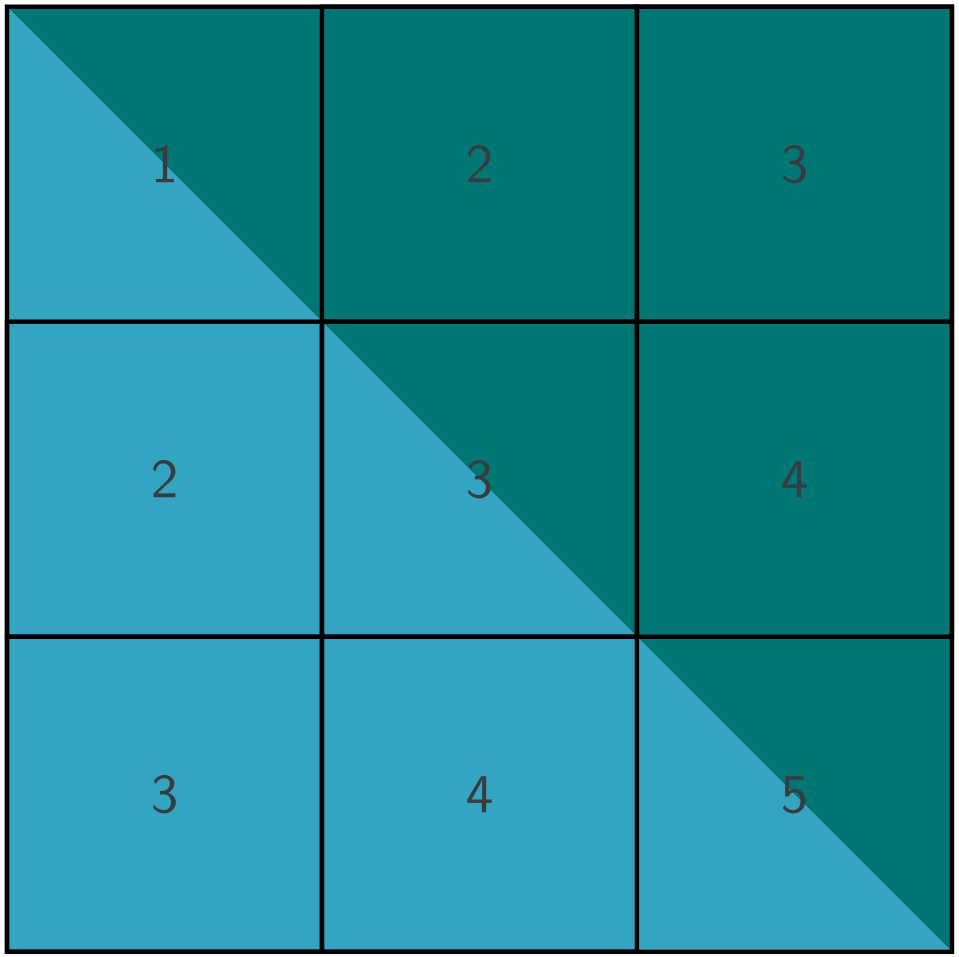
## The block outer product LU decomposition revisited

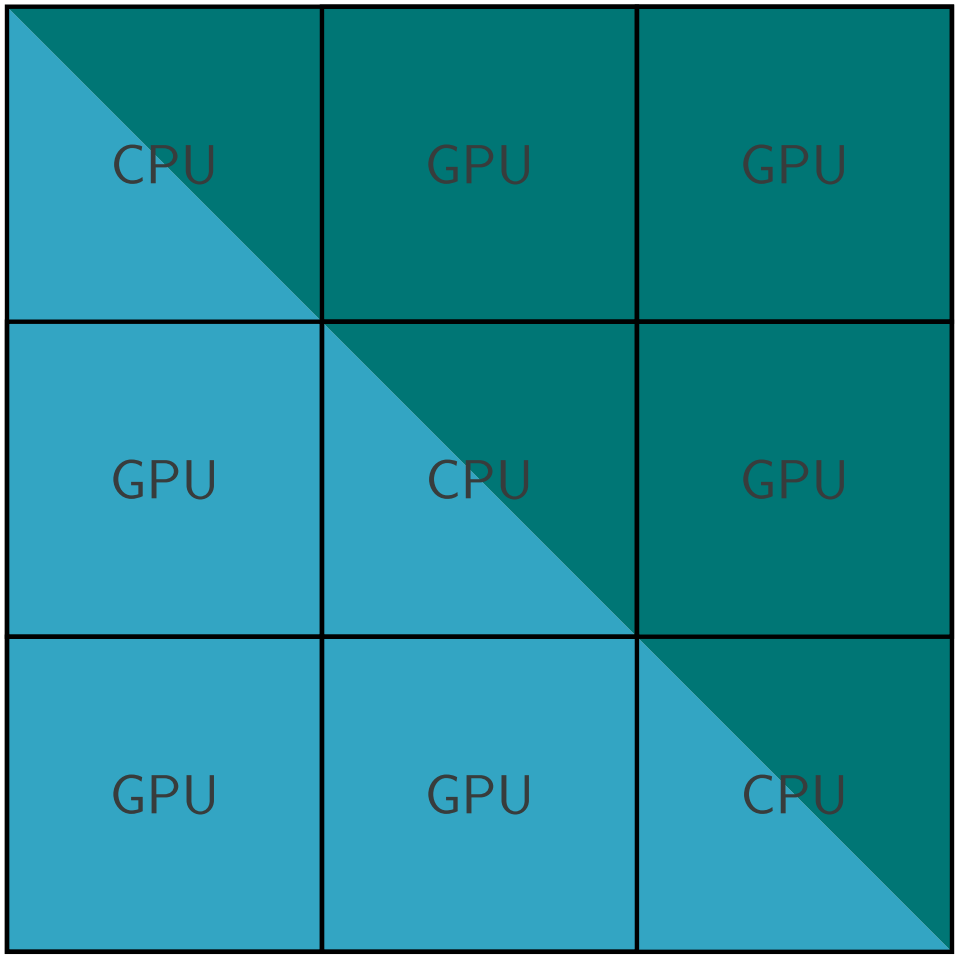
The central question for the hybrid CPU/GPU version of the algorithm now is where to execute the single steps of the algorithm compared to the DAG scheduled version.

### Requirements

- Keep data transfers between host and device limited
- optimize usage of both host and device features
- assume that the entire matrix fits into the device memory.

The assumption on the matrix size may be loosened but will then lead to a completely different algorithm.


1
2
3
2
3
4
3
4
5


CPU
GPU
GPU
GPU
CPU
GPU
GPU
GPU
CPU

In each outer iteration step perform the leading $r \times r$ blocks LU decomposition

**Algorithm 6:** Conjugate Gradient Method

**Input**: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n$
**Output**: $x = A^{-1}b$

$p_0 = r_0 = b - Ax_0, \ \alpha_0 = \|r_0\|_2^2;$
**for** $m = 0, \ldots, n-1$ **do**
  **if** $\alpha_m \neq 0$ **then**
    $v_m = Ap_m;$
    $\lambda_m = \frac{\alpha_m}{(v_m, p_m)};$
    $x_{m+1} = x_m + \lambda_m p_m;$
    $r_{m+1} = r_m - \lambda_m v_m;$
    $\alpha_{m+1} = \|r_{m+1}\|_2^2;$
    $p_{m+1} = r_{m+1} + \frac{\alpha_{m+1}}{\alpha_m} p_m;$
  **else**
    STOP;

There are mainly two observations we can draw from the algorithm.

1. The single steps need to be executed mainly sequentially
2. basically all operations are vector operations.

There is not much to distribute between host and device. To exploit the devices vector features all operations should be executed on the device. In case the matrix can not be stored in device memory completely it may be beneficial to use streams to split the operation into chunks that can be stored and operate on those streams in a round robin fashion.

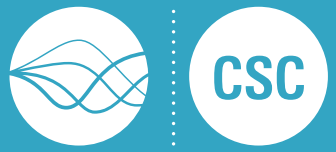
## Basic Idea

- Very similar to iterative linear solvers based on Krylov subspaces.
- Main ingredient is to use the basis of the subspace to project the eigenvalue problem to a much smaller space and solve it with dense methods there, i.e. $A \in \mathbb{R}^{n \times n}$ large and sparse $U \in \mathbb{R}^{m \times n}$, $m \ll n$ orthogonal, then

$$\underbrace{UAU^T}_{m \times m} x = \lambda x$$

is an $m$-dimensional dense eigenproblem.

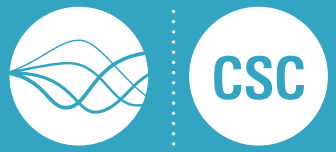
Here one can offload the solution of the small eigenvalue problem to the host, while the device keeps extending the basis further. The host can then decide whether the approximation is good enough, or the extension is required and the computation needs to continue.



# Relevant Software and Libraries

## The CUDA Related Libraries

- **CUDA Math** provides basically all math functions in `math.h` as device functions.
- **CUBLAS** the CUDA deice based implementation of BLAS
- **CUFFT** CUDA based Fast Fourier Transforms, i.e., divide and conquer based computation of Fourier transforms of complex and real valued data sets.
- **CURAND** The CURAND library provides facilities that focus on the simple and efficient generation of high-quality pseudorandom and quasirandom numbers.
- **CUSPARSE** Vector-vector and matrix-vector operations where at least one participant is sparse.
- **Thurst** A C++ template library based on the Standard Template library (STL) for minimal effort implementation of parallel programs.

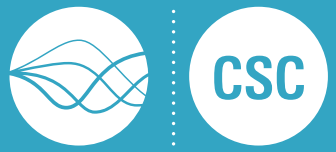


# Relevant Software and Libraries

## Matrix Algebra on GPU and Multicore Architectures (MAGMA)[21]

*"The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.*
*The MAGMA research is based on the idea that, to address the complex challenges of the emerging hybrid environments, optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, we aim to design linear algebra algorithms and frameworks for hybrid manycore and GPU systems that can enable applications to fully exploit the power that each of the hybrid components offers."*

[21] `http://icl.cs.utk.edu/magma/index.html`

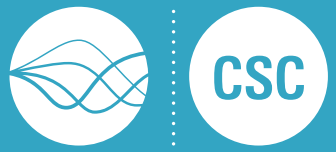


# Relevant Software and Libraries

## Formal Linear Algebra Methodology Environment (FLAME)[22]

*"The objective of the FLAME project is to transform the development of dense linear algebra libraries from an art reserved for experts to a science that can be understood by novice and expert alike. Rather than being only a library, the project encompasses a new notation for expressing algorithms, a methodology for systematic derivation of algorithms, Application Program Interfaces (APIs) for representing the algorithms in code, and tools for mechanical derivation, implementation and analysis of algorithms and implementations."*

[22] `http://www.cs.utexas.edu/~flame/web/`

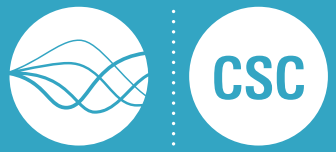


# Relevant Software and Libraries

## CUSP[23]

*"Cusp is a library for sparse linear algebra and graph computations on CUDA. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!"*

[23] https://github.com/cusplibrary



# Relevant Software and Libraries

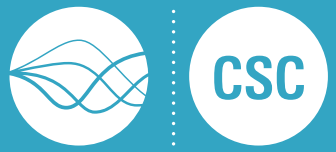
## CUSP[23]

*"Cusp is a library for sparse linear algebra and graph computations on CUDA. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!"*

Matrix formats:

- Coordinate (COO)
- Compressed Sparse Row (CSR)
- Diagonal (DIA)
- ELL (ELL)
- Hybrid (HYB)

[23] `https://github.com/cusplibrary`



# Relevant Software and Libraries

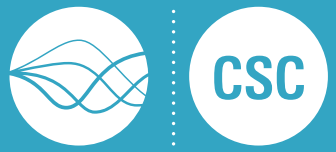
## CUSP[23]

*"Cusp is a library for sparse linear algebra and graph computations on CUDA. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!"*

More Features:

- Format conversion
- Dense Arrays
- File I/O (Matrix Market format)

[23] `https://github.com/cusplibrary`



# Relevant Software and Libraries

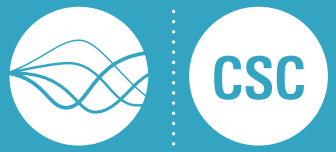
## CUSP[23]

*"Cusp is a library for sparse linear algebra and graph computations on CUDA. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!"*

Supported Iterative Solvers:

- Conjugate-Gradient (CG)
- Biconjugate Gradient (BiCG)
- Biconjugate Gradient Stabilized (BiCGstab)
- Generalized Minimum Residual (GMRES)
- Multi-mass Conjugate-Gradient (CG-M)
- Multi-mass Biconjugate Gradient stabilized (BiCGstab-M)

[23] `https://github.com/cusplibrary`



# Relevant Software and Libraries

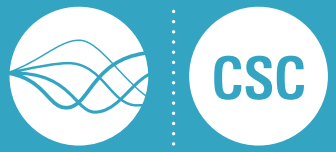
## CUSP[23]

*"Cusp is a library for sparse linear algebra and graph computations on CUDA. Cusp provides a flexible, high-level interface for manipulating sparse matrices and solving sparse linear systems. Get Started with Cusp today!"*

Preconditioners:

- Algebraic Multigrid (AMG) based on Smoothed Aggregation
- Approximate Inverse (AINV)
- Diagonal

[23] `https://github.com/cusplibrary`



# Relevant Software and Libraries

## CULA tools[24]

*"CULA is a set of GPU-accelerated linear algebra libraries utilizing the NVIDIA CUDA parallel computing architecture to dramatically improve the computation speed of sophisticated mathematics."*

They have separate packages for sparse and dense operation. The libraries are however commercial.

Besides those, there are many scientific computing packages that support GPU operations in one way or the other. Also python has packages for both CUDA (pyCUDA) and OpenCL (pyOpenCL) and MATLAB supports (basically dense only) operation on CUDA devices.

[24] `http://www.culatools.com`