Karlsruher Institut für Technologie

Bitte Logo über
\grouplogo{}
festlegen.

Mixed Precision Randomized Preconditioners for Regression Solvers on GPUs

Vasileios Georgiou | 16. May 2024



Introduction

1. Evaluate performance of randomized methods.
2. Mixed precision computations.
3. Software.

Why randNLA and mixed precision computing?

- Introduces noise to input data anyway.
- Benefits applications that require low accuracy.
- Can we improve quality of solutions?

Goals:

- Describe the numerical package
- Demonstrate some examples with dense and sparse matrices
- Showcase implicit regularization behavior



	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	

Randomized Numerical Linear Algebra: A Perspective on the Field With an Eye to Software

Riley Murray and James Demmel and Michael W. Mahoney and N. Benjamin Erichson and Maksim Melnichenko and Osman Asif Malik and Laura Grigori and Piotr Luszczek and Michal Derezinski and Miles E. Lopes and Tianyu Liang and Hengrui Luo and Jack Dongarra

EECS Department, University of California, Berkeley

Technical Report No. UCB/EECS-2023-19

February 22, 2023

http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-19.pdf

Introduction

1. Develop a module for randomized regression solvers on GPUs. randLS: https://github.com/vasilisge0/randLS/
2. Solvers
   - Implemented for dense and sparse matrices:
     - Preconditioned LSQR (right preconditioner)
     - Preconditioned FGMRES (split preconditioner)
   - Compare with
     1. QR
     2. QR on normal equations
3. Sketches:
   1. Gaussian (dense)
   2. k-countsketch (sparse)

   It should be possible to sketch both sparse and dense matrices.

$$x^* = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \left\| b - A x \right\|_2$$

Preconditioners for LSQR and FGMRES

Find $x^* = \arg\min \|Ax - b\|_2$ based on the implicit orthogonalization of the augmented matrix $\begin{bmatrix} I & A \\ A^\top & 0 \end{bmatrix}$

$$x^* \text{ is obtained from solving } \begin{bmatrix} I & A \\ A^\top & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Preconditioned LSQR assumes $n \times n$ preconditioner matrix M to solve

$$\begin{bmatrix} I & (AM^{-1}) \\ (AM^{-1})^\top & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

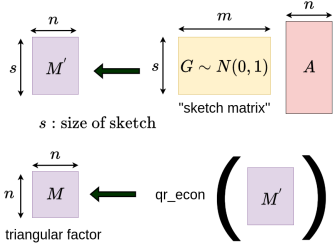
Preconditioned FGMRES also solves above equation using the arnoldi procedure for orthogonalization.

Randomized Preconditioning

Based on a “sketch” operation, a random transform to map input $m \times n$ matrix A to $s \times n$ matrix (SA)

- Early work by [Rokhlin, Tygert]
- Blendenpik outperforms dense qr codes [Avron, Maymounkov, Toledo]
- LSRN [Meng, Saunders, Mahoney]

Ideally sketching preserves the singular values of A to some extend.

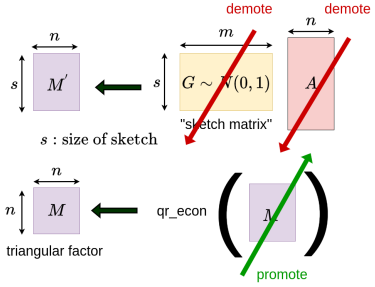


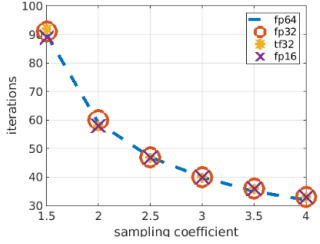
Randomized Preconditioning

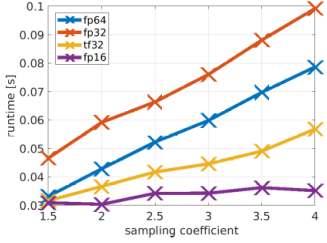
Based on a “sketch” operation, a random transform to map input $m \times n$ matrix A to $s \times n$ matrix (SA)

- Early work by [Rokhlin, Tygert]
- Blendenpik outperforms dense qr codes [Avron, Maymounkov, Toledo]
- LSRN [Meng, Saunders, Mahoney]

Ideally sketching preserves the singular values of A to some extend.







Mixed precision LSQR

Algorithm 2 Mixed precision preconditioned LSQR

Input: matrix $\boldsymbol{A}$, initial solution $\mathbf{x}_0$, right-hand side $\boldsymbol{b}$, tolerance `tol`, maximum number of iterations `maxiter`, preconditioner $\boldsymbol{M}$. The operations marked in blue color are performed in solver_precision, red is used for solver_internal and green for precond_apply_precision.
Output: solution $\mathbf{x}$, relative residual `relres`

procedure $[\mathbf{x}, \texttt{RELRES}] = \text{LSQR}(\boldsymbol{A}, \mathbf{x}_0, \boldsymbol{b}, \texttt{tol}, \texttt{maxiter}, \boldsymbol{M})$
  $\beta = \|\boldsymbol{b}\|_2, \mathbf{u} = \boldsymbol{b}/\beta$
  $\mathbf{v} = (M^{-1})\backslash(\boldsymbol{A}^{T}\mathbf{u})$
  $\alpha = \|\mathbf{v}\|_2, \mathbf{v} = \mathbf{v}/\alpha$
  $\mathbf{w} = \mathbf{v}$
  $\bar{\phi} = \beta, \bar{\rho} = \alpha$, `iter` $= 0$
  while (1) **do**
    $\mathbf{u} = \boldsymbol{A}(\boldsymbol{M}\backslash\mathbf{v}) - \alpha\mathbf{u}$
    $\beta = \|\mathbf{u}\|_2, \mathbf{u} = \mathbf{u}/\beta$
    $\mathbf{v} = \boldsymbol{M}^{T} \backslash (\boldsymbol{A}^{T}\mathbf{u}) - \beta v$
    $\alpha = \|\mathbf{v}\|_2, \mathbf{v} = \mathbf{v}/\alpha$
    $\rho = \sqrt{\bar{\rho}^2 + \beta^2}$
    $c = \bar{\rho}/\rho, s = \beta/\rho$
    $\theta = s \cdot \alpha, \bar{\rho} = -c \cdot \alpha$
    $\phi = c \cdot \bar{\phi}, \bar{\phi} = s \cdot \bar{\phi}$
    $\mathbf{x} = \mathbf{x} + \boldsymbol{M}\backslash((\phi/\rho)\mathbf{w})$
    $\mathbf{w} = \mathbf{v} - (\theta/\rho)\mathbf{w}$
    $\mathbf{r} = \boldsymbol{b} - \boldsymbol{A}\mathbf{x}$, `relres` $= \|\mathbf{r}\|_2/\|\boldsymbol{b}\|_2$, `iter` $+= 1$
    if ((`iter` == `maxiter`)||(`relres` < `tol`)) **then**
      break
    end if
  end while
end procedure

Algorithm 3 IR procedure

Input: matrix $\boldsymbol{A}$, initial solution $\mathbf{x}_0$, right-hand
tolerance `tol`, maximum number of iterations ma
preconditioner $\boldsymbol{M}$. The operations marked in bl
are performed in solver_precision, red is u
solver_internal and green for precond_apply_pre
Output: solution $\mathbf{x}$, relative residual `relres`

procedure $[x] = \text{IR}(\boldsymbol{A}, \mathbf{x}_0, \boldsymbol{b}, \boldsymbol{M})$
  $\mathbf{r} = \boldsymbol{b} - \boldsymbol{A}\mathbf{x}_0$
  $\mathbf{x} = \mathbf{x}_0$
  while (1) **do**
    $\mathbf{t} = \text{lsqr}(\boldsymbol{A}, \mathbf{x}, \mathbf{r}, \boldsymbol{M}, \text{config})$
    $\mathbf{x} = \mathbf{x} + \mathbf{t}$
    $\mathbf{r} = \boldsymbol{b} - \boldsymbol{A}\mathbf{x}$
  end while

- vtype : main storage precision
- vtype_internal_i : for generating operators
- vtype_apply : for applying operators
- vtype_refine : for accumulating output

Mixed precision implementations

operation	sparse matrix precision	dense matrix precision
precond	fp64, fp32	fp64, fp32
precond_internal0	fp64, fp32, fp16	fp64, fp32, tf32, fp16
precond_internal1	fp64, fp32, fp16	fp64, fp32
precond_apply	fp64, fp32, fp16	fp64, fp32, tf32, fp16
solver	fp64, fp32	fp64, fp32
solverinternal	fp64, fp32, fp16	fp64, fp32, tf32, fp16
ir (accumulator)	fp64, fp32	fp64, fp32

Tabelle: Precisions used by the preconditioner, solver and the iterative refinement procedure.

Example: FGMRES-solver

```
using dense = rls::matrix::Dense<rls::CUDA, double>;
using sparse = rls::matrix::Sparse<rls::CUDA, double, int>;
auto context = rls::share(rls::Context<rls::CUDA>::create());
auto input_mtx = filename_matrix;
auto input_rhs = filename_rhs;
auto mtx =
    rls::share(rls::matrix::Sparse<rls::CUDA, double, int>::create(
        context, input_mtx));
auto rhs = rls::share(
    rls::matrix::Dense<rls::CUDA, double>::create(context, input_rhs));
auto sol = rls::share(dense::create(context, dim2(mtx->get_size()[1], 1)));
sol->zeros();
std::shared_ptr<rls::Preconditioner> precond;


/* Preconditioner generation */

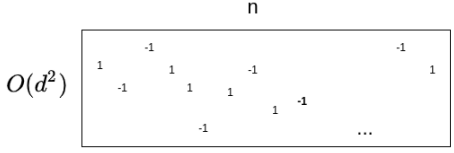
size_t k = 1;
dim2 sketch_size = {
    static_cast<int>(std::ceil(sampling_coeff * mtx->get_size()[1])),
    mtx->get_size()[0]};
using sketch_type =
    rls::CountSketch<rls::CUDA, double, vtype_precond_internal,
                     int>;
auto sketch = rls::share(sketch_type::create(context, k, sketch_size));
std::shared_ptr<rls::preconditioner::Config> precond_config;
using precond_config_type =
    rls::preconditioner::generalized_split::Config
        <double, vtype_precond_internal, vtype_precond_apply, int>;
precond_config = rls::share(precond_config_type::create(sampling_coeff));
precond = rls::preconditioner::GeneralizedSplit<rls::CUDA, double,
    vtype_precond_internal, vtype_precond_apply, int>::create(mtx, sketch, precond_config);
```

```
/* Solver construction */

std::shared_ptr<rls::Solver<rls::CUDA>> ir;
using Solver = rls::Solver<rls::CUDA>;
using SolverLogger = rls::solver::iterative::Logger;
using SolverConfig = rls::solver::iterative::Config;
std::shared_ptr<rls::Solver<rls::CUDA>> solver;
std::shared_ptr<SolverConfig> solver_config;
std::cout << "\n\n";
std::cout << "       matrix: " << matrix << '\n';
std::cout << "sampling_coeff: " << sampling_coeff << '\n';
std::cout << "     tolerance: " << tol << '\n';
auto input_true_sol = filename_true_sol;
auto input_noisy_sol = filename_noisy_sol;
using Fgmres = rls::solver::Fgmres
    <rls::CUDA, vtype_solver, vtype_solver_internal, vtype_precond_apply, double, int>;
using FgmresConfig = rls::solver::iterative::FgmresConfig
    <vtype_solver, vtype_solver_internal, vtype_precond_apply, double, int>;
solver_config = FgmresConfig::create();
solver_config->set_precond(precond);
solver_config->set_tolerance(tol);
solver_config->set_stagnation_tolerance(1e-3);  // <= sqrt(gemv precision)
solver_config->set_restarts(restarts);
solver_config->set_min_iterations(miniter);
solver_config->set_iterations(maxiter);
solver_config->set_filename_relres(filename_relres);
solver_config->set_filename_true_error(input_true_sol,
                                       filename_true_error);
solver_config->set_filename_noisy_error(input_noisy_sol,
                                        filename_noisy_error);
solver_config->set_filename_stagnation(filename_stagnation);
solver_config->set_filename_similarity(filename_true_similarity,
                                       filename_noisy_similarity);
sol->zeros();
solver =
    rls::share(Fgmres::create(context, solver_config, mtx, sol, rhs));
ir = rls::solver::Ir<rls::CUDA, int>::create(solver, restarts);
ir->run();
auto logger = ir->get_logger();
logger.write_history();
}
```

Countsketch

- specialized sketch-apply operation
- very sparse
- requires larger embedding dimension than gaussian



We will mostly use coutsketch for sparse and dense matrices. Essential to improve runtime. For all our solvers we use a similar configuration of 100 restarts, 20 minimum iterations, 500 maximum iterations per restart. Tolerance is set to 1e-14, restarts are scheduled automatically.

Comparisons for sampling coefficient = 2.0

id	dataset	dimensions	AR	$\kappa(A)$
1	mesh_deform	$234,023 \times 9,393$	25	$1,166$
2	Delor64K	$1,785,345 \times 10,000$	178	
3	sprand_03	$10,000 \times 100$	100	$1e5$
4	sparse_large	$100,000 \times 500$	200	$1e5$
5	sparse_large_2	$100,000 \times 1,000$	100	$1e7$
6	Rucci_1_0	$1,997,885 \times 1,000$	$1,997$	$1e5$

matrix	dimensions	AR	gaussian	CS-1	CS-100	(CS-1, gaus)	$A'A$
mesh_deform	$234,093 \times 9,393$	25	-	**0.0074**	0.2019	0.2290	0.0100
delor64K	$1,785,093 \times 10,000$	178	-	0.0106	0.2009	0.2572	**0.0093**
sprand_03	$10,000 \times 100$	100	0.0365	**0.0103**	0.0178	0.0144	0.0118
sparse_large	$100,000 \times 500$	200	0.0874	**0.0190**	0.5964	0.0111	0.0280
sparse_large2	$100,000 \times 1000$	100	0.2259	0.0373	1.5759	0.0282	**0.0109**
rucci1_0	$1,997,885 \times 1000$	$1,997$	-	0.0044	0.1956	0.0053	**0.0111**

Tabelle: sketch-apply runtimes.

Preconditioned LSQR

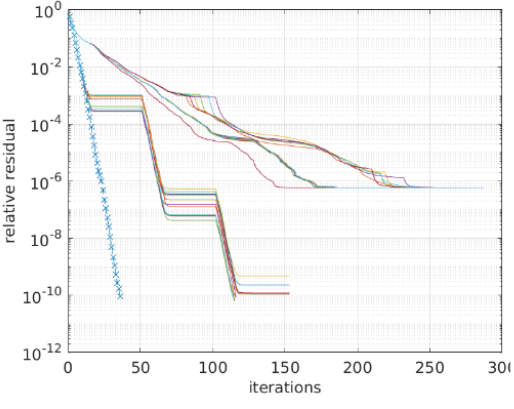
matrix	dimensions	aspect ratio	no precond	sc=2.0	sc=2.5	sc=3.0	sc=3.5
mesh_deform	$234,093 \times 9,393$	25	**1**, **887**	116	93	80	74
delor64K	$1,785,093 \times 10,000$	178	**305**	211	169	150	142
sprand_03	$10,000 \times 100$	100	**106**	80	61	59	49
sparse_large	$100,000 \times 500$	200	**765**	84	73	56	51
sparse_large2	$100,000 \times 1000$	100	**1909**	108	64	57	53
rucci1_0	$1,997,885 \times 1000$	$1,997$	**362**	68	58	51	47

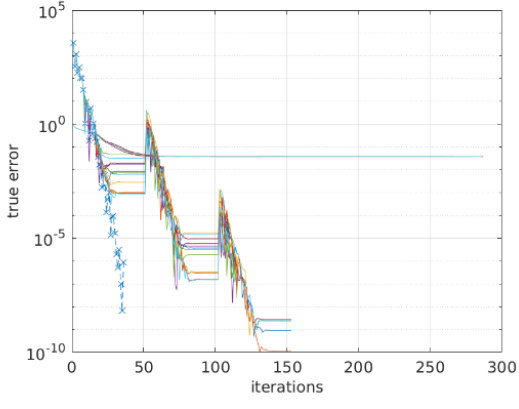
Tabelle: Convergence to $1e-14$ relative residual norm for different sampling coefficients.

Runtimes

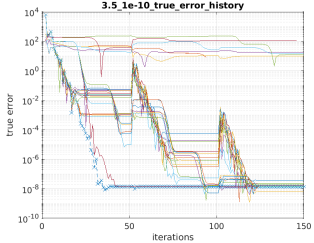
	cusolver	geqrsv	SPQR	cusolver-ne	SPQR-ne	plsqr
mesh_deform	-	174	0.70	0.89	0.92	1.30
Delor64K	0.81	0.73	0.17	0.41	0.23	161
sprand_03	0.13	0.24	0.04	0.08	4.60	**0.09**
sparse_large	31.7	0.59	3.25	0.56	101	**0.20**
sparse_large_2	121	270	8.52	3.16	475	**0.38**
Rucci_1_small	16.2	0.10	0.15	0.10	0.94	0.38

Mixed precision preconditioned LSQR example, convergence

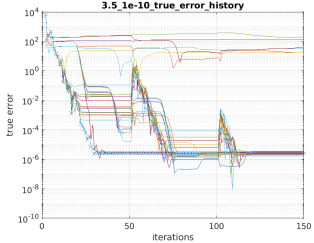




Adding noise to rhs, noise levels = [1e-10 1e-8 1e-5 1e-3]

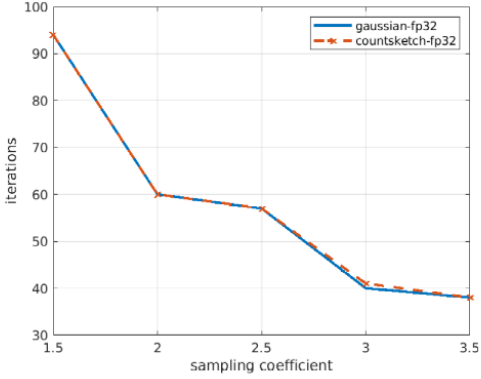


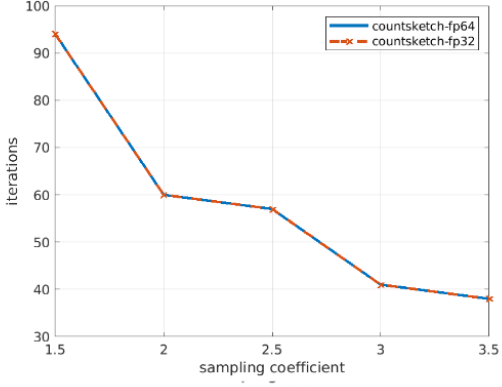




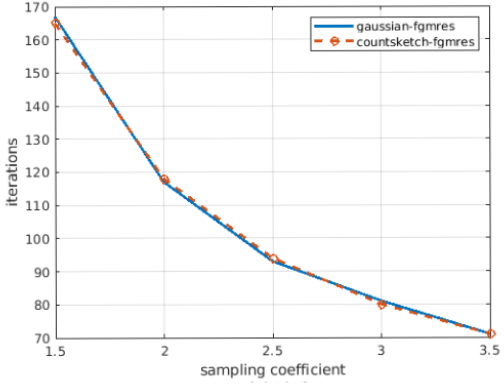


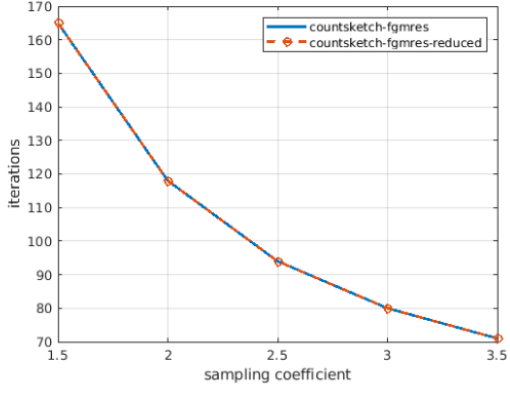
Lsqr countsketch example





Fgmres countsketch example





Recent implementation / Sketch and solve - IR

```
void run_sketch_and_solve_instance(rls::test::instance_config_t parameters)
{
    /* Defines types of matrices and inner solver configuration. */
    using dense = rls::matrix::Dense<rls::CUDA, double>;
    using sparse = rls::matrix::Sparse<rls::CUDA, double, int>;
    using SolverConfig = rls::solver::iterative::Config;
    using QrConfig = rls::solver::iterative::QrConfig<double, int>;
    std::shared_ptr<SolverConfig> solver_config;
    solver_config = QrConfig::create();

    /* Setup filenames. */
    std::string offset = "0";
    std::string filename_offset = "_" + offset + ".mtx";
    auto filename_matrix = parameters.filename_matrix;
    auto filename_rhs = parameters.filename_rhs;
    auto filename_xtrue = parameters.filename_xtrue;
    auto filename_xnoisy = parameters.filename_xnoisy;
    auto filename_relres = parameters.filename_relres + filename_offset;
    auto filename_true_error = parameters.filename_true_error + filename_offset;
    auto filename_stagnation = parameters.filename_stagnation + filename_offset;
    auto filename_noisy_error = parameters.filename_noisy_error + filename_offset;
    auto filename_true_similarity = parameters.filename_true_similarity + filename_offset;
    auto filename_noisy_similarity = parameters.filename_noisy_similarity + filename_offset;

    /* Setup solver convergence configurations. */
    //solver_config->set_precond(precond);
    solver_config->set_tolerance(parameters.tol);
    solver_config->set_restarts(parameters.restarts);
```

```

    /* Setup output filenames. */
    solver_config->set_filename_relres(filename_relres);
    solver_config->set_filename_true_error(filename_xtrue, filename_true_error);
    solver_config->set_filename_noisy_error(filename_xnoisy, filename_noisy_error);
    solver_config->set_filename_stagnation(filename_stagnation);
    solver_config->set_filename_similarity(filename_true_similarity, filename_noisy_similarity);

    /* Initialize matrix, right-hand side, solution vector and run solver. */
    auto context = rls::share(rls::Context<rls::CUDA>::create());
    auto mtx = rls::share(rls::matrix::Dense<rls::CUDA, double>::create(context, parameters.filename_matrix));
    auto rhs = rls::share(rls::matrix::Dense<rls::CUDA, double>::create(context, parameters.filename_rhs));
    auto sol = rls::share(dense::create(context, dim2(mtx->get_size()[1], 1)));
    auto context_ref = rls::share(rls::Context<rls::CPU>::create());
    double sampling_coeff = parameters.sampling_coeff;
    dim2 sketch_size_0 = {
        static_cast<int>(std::ceil(sampling_coeff * mtx->get_size()[1])),
        mtx->get_size()[0]};
    auto solver = rls::share(rls::solver::Qr<rls::CUDA, double, int>::create(context, solver_config));
    auto sketch = rls::share(rls::GaussianSketch<rls::CUDA, double, double, magma_int_t>::create(context, sketch_size_0));
    auto outer_solver = rls::share(
            rls::solver::SketchAndSolve<rls::CUDA, double, double, double, double, int>::create(context, mtx, sol, rhs, sol
    std::shared_ptr<rls::Solver<rls::CUDA>> ir;
    ir = rls::solver::Ir<rls::CUDA, magma_int_t>::create(outer_solver, solver_config->get_restarts());
    ir->run();
```

Conclusions and future work

Overview:

- Design and implementation of mixed precision preconditioned iterative solvers
- Preconditioner generation in reduced precision
- Implementation of sparse sketches

Current issues:

1. Difficulties outperforming state of the art direct QR solvers.
2. More experiments required as matrices used are extremely sparse.

Future work:

- Randomly preconditioned dense least squares show promise.
- Implementation of an explicit regularization method to better showcase implicit regularization.

This is joint work with Hartwig Anzt (TUM), Christos Boutsikas (Purdue), Ieva Dauzizkaite (Charles University), Petros Drineas (Purdue), Erin Carson (Charles university).