Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library
2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

MCS 572 Lecture 22
Introduction to Supercomputing
Jan Verschelde, 12 October 2016

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library
2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

solving $A\mathbf{x} = \mathbf{b}$ with the QR factorization

To solve an overconstrained linear system $A\mathbf{x} = \mathbf{b}$
of m equations in n variables, where $m > n$,
we factor A as a product of two matrices, $A = QR$:

- Q is an orthogonal matrix: $Q^H Q = I$,
  the inverse of Q is its Hermitian transpose Q^H;
- R is upper triangular: $R = [r_{i,j}]$, $r_{i,j} = 0$ if $i > j$.

Solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving $Q(R\mathbf{x}) = \mathbf{b}$:

1. Multiply $\mathbf{b}$ with Q^T.
2. Apply backward substitution: $R\mathbf{x} = Q^T\mathbf{b}$.

The solution $\mathbf{x}$ minimizes $||b - A\mathbf{x}||_2^2$ (least squares).

Householder reflectors

For $\mathbf{v} \neq 0$: $H = I - \dfrac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$ is a Householder transformation.

By its definition, $H = H^T = H^{-1}$.

For some $\mathbf{x} \in \mathbb{R}^n$, let $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$:

$$\mathbf{v} = \mathbf{x} - ||\mathbf{x}||_2 \mathbf{e}_1 \quad \Rightarrow \quad H\mathbf{x} = ||\mathbf{x}||_2 \mathbf{e}_1 .$$

With H we eliminate, reducing A to an upper triangular matrix.

Householder QR: Q is a product of Householder reflectors.

The LAPACK `DGEQRF` on an m-by-n matrix produces:

$$H_1 H_2 \cdots H_n = I - VTV^T$$

where each H_i is a Householder reflector matrix with associated vectors $\mathbf{v}_i$ in the columns of V and T is an upper triangular matrix.

Householder QR factorization

For an m-by-n matrix $A = [a_{i,j}]$ with columns $\mathbf{a}_j$:

for $k = 1, 2, \ldots, n$ do
  $\alpha_k := -\text{sign}(a_{k,k})\sqrt{a_{k,k}^2 + \cdots + a_{m,k}^2}$;
  $\mathbf{v}_k = [0 \cdots 0 \; a_{k,k} \cdots a_{m,k}]^T - \alpha_k \mathbf{e}_k$;
  $\beta_k := \mathbf{v}_k^T \mathbf{v}_k$;
  if $\beta_k \neq 0$ then
    for $j = k, k+1, \ldots, n$ do
      $\gamma_j := \mathbf{v}_k^T \mathbf{a}_j$;
      $\mathbf{a}_j := \mathbf{a}_j - 2\frac{\gamma_j}{\beta_k}\mathbf{v}_k$;
    end for;
  end if;
end for.

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library
2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

tiled matrices

Divide the m-by-n matrix A into b-by-b blocks,
$m = b \times p$ and $n = b \times q$.

Then we consider A as an $(p \times b)$-by-$(q \times b)$ matrix:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,q} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p,1} & A_{p,2} & \cdots & A_{p,q} \end{bmatrix},$$

where each $A_{i,j}$ is an b-by-b matrix.

A crude classification of memory hierarchies distinguishes between registers (small), cache (medium), and main memory (large).

To reduce data movements, we want to keep data in registers and cache as much as possible.

QR factorization on a tiled matrix

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} = Q \begin{bmatrix} R_{1,1} & R_{1,2} \\ 0 & R_{2,2} \end{bmatrix}$$

We proceed in three steps.

1. Perform a QR factorization:

$$\begin{array}{l} A_{1,1} = Q_1 B_{1,1} \\ (Q_1, B_{1,1}) := \mathrm{QR}(A_{1,1}) \end{array} \quad \text{and} \quad \begin{array}{l} A_{1,2} = Q_1 B_{1,2} \\ B_{1,2} = Q_1^T A_{1,2} \end{array}$$

2. Coupling square blocks:

$$(Q_2, R_{1,1}) := \mathrm{QR}\left(\begin{bmatrix} B_{1,1} \\ A_{2,1} \end{bmatrix} \right) \quad \text{and} \quad \begin{bmatrix} R_{1,2} \\ B_{2,2} \end{bmatrix} := Q_2^T \begin{bmatrix} R_{1,2} \\ A_{2,2} \end{bmatrix}.$$

3. Perform another QR factorization: $(Q_3, R_{2,2}) := \mathrm{QR}(B_{2,2})$.

tiled QR factorization

for $k = 1, 2, \ldots, \min(p, q)$ do

  DGEQRT($A_{k,k}$, $V_{k,k}$, $R_{k,k}$, $T_{k,k}$); -- $V_{k,k}, R_{k,k}, T_{k,k} := \mathrm{QR}(A_{k,k})$

  for $i = k+1, \ldots, p$ do

    DLARFB($A_{k,j}$, $V_{k,k}$, $T_{k,k}$, $R_{k,j}$); -- $R_{k,j} := (I - V_{k,k} T_{k,k} V_{k,k}^T) A_{k,j}$

  end for;

  for $i = k+1, \ldots, p$ do

    DTSQRT($R_{k,k}$, $A_{i,k}$, $V_{i,k}$, $T_{i,k}$); -- $(V_{i,k}, T_{i,k}, R_{k,k}) := \mathrm{QR}\left(\begin{bmatrix} R_{k,k} \\ A_{i,k} \end{bmatrix}\right)$

    for $j = k+1, \ldots, p$ do

      DSSRFB($R_{k,j}$, $A_{i,j}$, $V_{i,k}$, $T_{i,k}$); -- $\begin{bmatrix} R_{k,j} \\ A_{i,j} \end{bmatrix} := (I - V_{i,k} T_{i,k} V_{i,k}^T) \begin{bmatrix} R_{k,j} \\ A_{i,j} \end{bmatrix}$

    end for;

end for.

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library

2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

compiling `example_dgeqrs`

The directory `examples` in
`/home/jan/Downloads/plasma-installer_2.8.0/build/plasma_2.8.0`
contains `example_dgeqrs`, an example for solving an overdetermined linear system with a QR factorization.

Compiling with a `makefile`:

```
$ make example_dgeqrs
gcc -O2 -DADD_ -I../include -I../quark \
-I/home/jan/Downloads/plasma-installer_2.8.0/install/include \
-c example_dgeqrs.c -o example_dgeqrs.o \
gfortran  example_dgeqrs.o -o example_dgeqrs -L../lib \
-lplasma -lcoreblasqw  -lcoreblas -L../quark -lquark \
-L/home/jan/Downloads/plasma-installer_2.8.0/install/lib \
-lcblas \
-L/home/jan/Downloads/plasma-installer_2.8.0/install/lib \
-llapacke \
-L/home/jan/Downloads/plasma-installer_2.8.0/install/lib \
-ltmg -llapack \
-L/home/jan/Downloads/plasma-installer_2.8.0/install/lib \
-lrefblas  -lpthread -lm
$
```

defining a makefile

```
PLASMA=/home/jan/Downloads/plasma-installer_2.8.0
PLASMALIB=\
/home/jan/Downloads/plasma-installer_2.8.0/build/plasma_2.8.0/lib

example_dgeqrs:
        gcc -O2 -DADD_ -I$(PLASMA)/build/include \
            -I$(PLASMA)/build/plasma_2.8.0/quark \
            -I$(PLASMA)/install/include \
            -c example_dgeqrs.c -o example_dgeqrs.o
        gfortran  example_dgeqrs.o -o example_dgeqrs \
            -L$(PLASMALIB) -lplasma -lcoreblasqw  -lcoreblas \
            -L$(PLASMA)/build/plasma_2.8.0/quark -lquark  \
            -L$(PLASMA)/install/lib -lcblas \
            -L$(PLASMA)/install/lib -llapacke \
            -L$(PLASMA)/install/lib -ltmg -llapack \
            -L$(PLASMA)/install/lib -lrefblas  -lpthread -lm
```

running `example_dgeqrs`

Running `example_dgeqrs` with dimension 4000.

```
]$ time ./example_dgeqrs
-- PLASMA is initialized to run on 1 cores.
============
Checking the Residual of the solution
-- ||Ax-B||_oo/((||A||_oo||x||_oo+||B||)_oo.N.eps) = 2.829153e-02
-- The solution is CORRECT !
-- Run of DGEQRS example successful !

real    0m42.172s
user    0m41.965s
sys     0m0.111s
$
```

running for different number of cores

Result of `time example_dgeqrs`, for dimension 4000:

p	real	user	sys	speedup
1	42.172	41.965	0.111	
2	21.495	41.965	0.154	1.96
4	11.615	43.322	0.662	3.63
8	6.663	46.259	1.244	6.33
16	3.872	46.793	2.389	10.89

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library

2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

an optimization problem

Let A be a positive definite matrix: $\forall \mathbf{x} : \mathbf{x}^T A\mathbf{x} \geq 0$ and $A^T = A$.
The optimum of

$$q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} \text{ is at } A\mathbf{x} - \mathbf{b} = \mathbf{0}.$$

For the exact solution $\mathbf{x}$: $A\mathbf{x} = \mathbf{b}$ and an approximation $\mathbf{x}_k$,
let the error be $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}$.

$$\begin{aligned} ||\mathbf{e}_k||_A^2 = \mathbf{e}_k^T A\mathbf{e}_k &= (\mathbf{x}_k - \mathbf{x})^T A(\mathbf{x}_k - \mathbf{x}) \\ &= \mathbf{x}_k^T A\mathbf{x}_k - 2\mathbf{x}_k^T A\mathbf{x} + \mathbf{x}^T A\mathbf{x} \\ &= \mathbf{x}_k^T A\mathbf{x}_k - 2\mathbf{x}_k^T\mathbf{b} + c \\ &= 2q(\mathbf{x}_k) + c \end{aligned}$$

$\Rightarrow$ minimizing $q(\mathbf{x})$ is the same as minimizing the error.

the conjugate gradient method

The CG method is similar to the steepest descent method.

$\mathbf{x}_0 = \mathbf{0}$; $\mathbf{r}_0 = \mathbf{b}$; $\mathbf{p}_0 := \mathbf{r}_0$
for $k = 0, 1, 2 \ldots$ do

$$\alpha_k := \frac{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{p}_{k-1}^T A \mathbf{p}_{k-1}} \qquad \textit{step length}$$

$$\mathbf{x}_k := \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_{k-1} \qquad \textit{update solution}$$

$$\mathbf{r}_k := \mathbf{r}_{k-1} - \alpha_k A \mathbf{p}_{k-1} \qquad \textit{residual}$$

$$\beta_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}} \qquad \textit{improvement of step}$$

$$\mathbf{p}_k := \mathbf{r}_k + \beta_k \mathbf{p}_{k-1} \qquad \textit{compute search direction}$$

For large sparse matrices, CG is much faster than Cholesky.

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library
2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

spaces spanned by matrix-vector products

Input: right hand side vector $\mathbf{b} \in \mathbb{R}^n$;
some algorithm to evaluate $A\mathbf{x}$.
Output: approximation for $A^{-1}\mathbf{b}$.

Example ($n = 4$), consider $\mathbf{y}_1 = \mathbf{b}$, $\mathbf{y}_2 = A\mathbf{y}_1$, $\mathbf{y}_3 = A\mathbf{y}_2$, $\mathbf{y}_4 = Ay_3$, stored in the columns of $K = [\mathbf{y}_1\ \mathbf{y}_2\ \mathbf{y}_3\ \mathbf{y}_4]$.

$$
\begin{aligned}
AK &= [A\mathbf{y}_1\ A\mathbf{y}_2\ A\mathbf{y}_3\ A\mathbf{y}_4] = [\mathbf{y}_2\ \mathbf{y}_3\ \mathbf{y}_4\ A^4\mathbf{y}_1] \\
&= K \begin{bmatrix} 0 & 0 & 0 & -c_1 \\ 1 & 0 & 0 & -c_2 \\ 0 & 1 & 0 & -c_3 \\ 0 & 0 & 1 & -c_4 \end{bmatrix}, \quad \mathbf{c} = -K^{-1}A^4\mathbf{y}_1.
\end{aligned}
$$

$\Rightarrow AK = KC$, where C is a companion matrix.

Krylov subspace methods

Set

$$K = [\mathbf{b} \;\; A\mathbf{b} \;\; A^2\mathbf{b} \;\; \cdots \;\; A^k\mathbf{b}].$$

Compute

$$K = QR.$$

Then the columns of Q span the *Krylov subspace*.

Look for an approximate solution of $A\mathbf{x} = \mathbf{b}$
in the span of the columns of Q.

We can interpret this as a modified Gram-Schmidt method,
stopped after k projections of $\mathbf{b}$.

parallel matrix-vector products

The conjugate gradient and Krylov subspace methods rely mostly on matrix-vector products.

Parallel implementations involve the distribution of the matrix among the processors.

For general matrices, tiling scales best.

For specific sparse matrices, the patterns of the nonzero elements will determine the optimal distribution.

Parallel Numerical Linear Algebra

1. QR Factorization
   - solving overconstrained linear systems
   - tiled QR factorization
   - using the PLASMA software library

2. Conjugate Gradient and Krylov Subspace Methods
   - linear system solving and optimization
   - Krylov subspace methods
   - using the PETSc toolkit

selecting an example

PETSc (see Lecture 20) provides a directory of examples `ksp` of Krylov subspace methods. We will take `ex3` of `/home/jan/Downloads/petsc-3.7.4/src/ksp/ksp/examples/tests`

To get the makefile, we compile in the examples and then copy the compilation and linking instructions.

the makefile on kepler

```
PETSC_DIR=/home/jan/Downloads/petsc-3.7.4
PETSC_ARCH=arch-linux2-c-debug

ex3:
        $(PETSC_DIR)/$(PETSC_ARCH)/bin/mpicc -o ex3.o -c -fPIC -Wall \
        -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -g3 \
        -fno-inline -O0   -I/usr/local/petsc-3.4.3/include \
        -I/usr/local/petsc-3.4.3/arch-linux2-c-debug/include \
        -D__INSDIR__=src/ksp/ksp/examples/tests/ ex3.c
        $(PETSC_DIR)/$(PETSC_ARCH)/bin/mpicc -fPIC -Wall -Wwrite-strings \
        -Wno-strict-aliasing -Wno-unknown-pragmas -g3 -fno-inline -O0 \
        -o ex3 ex3.o \
        -Wl,-rpath,/usr/local/petsc-3.4.3/arch-linux2-c-debug/lib \
        -L/usr/local/petsc-3.4.3/arch-linux2-c-debug/lib  -lpetsc \
        -Wl,-rpath,/usr/local/petsc-3.4.3/arch-linux2-c-debug/lib \
        -lflapack -lfblas -lX11 -lpthread -lm \
        -Wl,-rpath,/usr/gnat/lib/gcc/x86_64-pc-linux-gnu/4.7.4 \
        -L/usr/gnat/lib/gcc/x86_64-pc-linux-gnu/4.7.4 \
        -Wl,-rpath,/usr/gnat/lib64 -L/usr/gnat/lib64 \
        -Wl,-rpath,/usr/gnat/lib -L/usr/gnat/lib -lmpichf90 -lgfortran \
        -lm -Wl,-rpath,/usr/lib/gcc/x86_64-redhat-linux/4.4.7 \
        -L/usr/lib/gcc/x86_64-redhat-linux/4.4.7 -lm -lmpichcxx -lstdc++ \
        -ldl -lmpich -lopa -lmpl -lrt -lpthread -lgcc_s -ldl
        /bin/rm -f ex3.o
```

running the example

```
$ time /home/jan/Downloads/petsc-3.7.4/arch-linux2-c-debug/bin/\
mpiexec -n 16 /tmp/ex3 -m 200
Norm of error 0.000622192 Iterations 162

real 0m2.243s user 0m8.169s sys 0m5.753s
$ time /home/jan/Downloads/petsc-3.7.4/arch-linux2-c-debug/bin/\
mpiexec -n 8 /tmp/ex3 -m 200
Norm of error 0.000387934 Iterations 131

real 0m1.829s user 0m7.546s sys 0m1.738s
$ time /home/jan/Downloads/petsc-3.7.4/arch-linux2-c-debug/bin/\
mpiexec -n 4 /tmp/ex3 -m 200
Norm of error 0.00030938 Iterations 105

real 0m4.238s user 0m14.402s sys 0m0.944s
$
```

suggested reading

- E. Agullo, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, J. Langou, H. Ltaief, P. Luszczek, and A. YarKhan. **PLASMA Users' Guide. Parallel Linear Algebra Software for Multicore Architectures. Version 2.0**.
- S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang. **PETSc Users Manual. Revision 3.4**. Mathematics and Computer Science Division, Argonne National Laboratory, 15 October 2013.
- A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. **A class of parallel tiled linear algebra algorithms for multicore architectures.** *Parallel Computing* 35: 38-53, 2009.

Summary + Exercises

We looked at tiled QR factorization for multicore computers and encountered QR again in the Krylov subspace methods.

Exercises:

1. Take a 3-by-3 block matrix, apply the tiled QR algorithm step by step, indicating which block is processed in each step along with the type of operations. Draw a Directed Acyclic Graph linking steps that have to wait on each other. On a 3-by-3 block matrix, what is the number of cores needed to achieve the best speedup?
2. Using the `time` command to measure the speedup of `example_dgeqrs` also takes into account all operations to define the problem and to test the quality of the solution. Run experiments with modifications of the source code to obtain more accurate timings of the parallel QR factorization.