


Parallel Numerical Algorithms

Linear Algebra Libraries:
LAPACK and ScaLAPACK

Chris Johnson
Applications Consultant, EPCC
c.johnson@epcc.ed.ac.uk
+44 131 650 5362

Linear algebra libraries



- Linear Algebra
- Matrix types
- BLAS
- LINPACK
- LAPACK
 - LU Factorisation
- Sparse Matrices
- ScaLAPACK

Linear algebra libraries



- Linear Algebra is a well constrained problem
 - can define a small set of common operations
 - implement them robustly and efficiently in a library
 - mainly designed to be called from Fortran (see later ...)
- Often seen as the most important HPC library
 - eg LINPACK benchmark is standard HPC performance metric
 - possible to achieve performance close to theoretical peak
- Linear algebra is unusually efficient
 - LU decomposition has $O(N^3)$ operations for $O(N^2)$ memory loads

Matrix types



- Matrices generally classified as either *sparse* or *dense*
 - We will deal with sparse matrices later
- Rectangular matrices
 - correspond to different number of equations from unknowns
 - system can be either under or over-determined
- Matrices may have symmetry about the diagonal:
 - Symmetric (real matrices): $a_{ij} = a_{ji} \rightarrow \mathbf{A}^T = \mathbf{A}$
 - Hermitian (complex matrices): $a_{ij}^* = a_{ji} \rightarrow \mathbf{A}^{T*} = \mathbf{A}^\dagger = \mathbf{A}$

$$\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

Symmetric

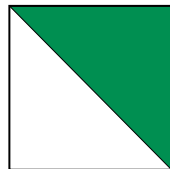
$$\begin{pmatrix} 1 & 2+i \\ 2-i & 3 \end{pmatrix}$$

Hermitian

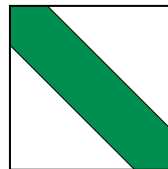
Matrices

epcc

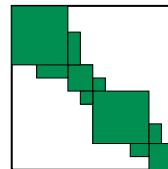
- Many matrices have a regular structure



Upper
triangular



Band
diagonal



Block
diagonal

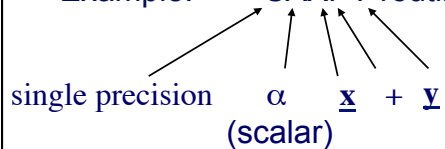
- Can exploit regular structure or symmetry for efficiency
 - eg special form of LU decomposition for tridiagonal matrices

BLAS

epcc

- Basic Linear Algebra Subprograms
 - Level 1: vector-vector operations (e.g. $\underline{x} \cdot \underline{y}$)
 - Level 2: matrix-vector operations (e.g. $\mathbf{A}\underline{x}$)
 - Level 3: matrix-matrix operations (e.g. $\mathbf{A}\mathbf{B}$)

- Example: SAXPY routine



\underline{y} is replaced “in-place” with $\alpha \underline{x} + \underline{y}$

SAXPY



- (In Fortran

```
call SAXPY(n,a,x,incx,y,incy)
...
ix = 1; iy = 1
do i = 1, n
    y(iy) = y(iy) + a * x(ix)
    ix = ix + incx; iy = iy + incy
end do
```

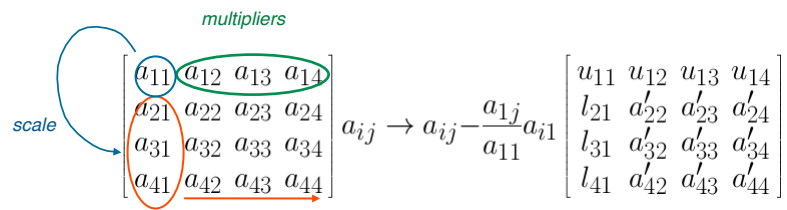
LINPACK library



- <http://www.netlib.org/linpack/>
 - LINPACK is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems.
 - The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular and tridiagonal square.
 - In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems.
 - LINPACK uses column-oriented algorithms to increase efficiency by preserving locality of reference.
- LINPACK *benchmark* means LU factorisation

Why is performance so high? LU factorisation | epcc

- Look at step in LU decomposition
 - consider an in-place decomposition
 - for each column k
 - scale the sub-diagonal column below a_{kk} by $1/a_{kk}$
 - for all columns j to the right, subtract a_{kj} times above
 - eg for $k = 1$



- These are vector ^{sweep}operations over columns
 - scaling a vector x by a (a scalar variable): $x_i \rightarrow a x_i$
 - updating a second vector y by the above: $y_i \rightarrow a x_i + y_i$

LINPACK implementation | epcc

- Based around level 1 BLAS

– eg look at LU decomposition

```

t = -1.0e0/a(k,k)
call SSCAL(n-k,t,a(k+1,k),1)
...
do j = k+1,n
  call SAXPY(n-k,t,a(k+1,k),1,a(k+1,j),1)
end do

```

- Very efficient on early HPC vector machines
 - not nearly so efficient on modern cache-based systems

LAPACK



- LAPACK is built on top of BLAS libraries
 - Most of the computation is done with the BLAS libraries
- Original goal of LAPACK was to improve upon EISPACK and LINPACK libraries to run more efficiently on shared memory and multi-layered systems
 - Spend less time moving data around!
- LAPACK attempts to use BLAS 3 instead of BLAS 1
 - matrix-matrix operations more efficient than vector-vector
- Illustrates trend to layered numerical libraries
 - allows for portable performance libraries
 - efficient implementation of BLAS 3 leads immediately to efficient implementation of LAPACK
 - porting LAPACK becomes a straightforward exercise

BLAS/LAPACK naming conventions



- Routines generally have a name of up to 6 letters, e.g. DGESV,
- Initial letter
 - S: Real
 - D: Double Precision
 - C: Complex
 - Z: Double Complex or COMPLEX*16
- For level 2 and 3 routines, 2nd and 3rd letter refers to matrix type
 - GE: matrices are general rectangular (i.e. could be unsymmetric, not necessarily square)
 - HE: (complex) Hermitian
 - SY: symmetric
 - TR: triangular
 - BD: bidiagonal
 - etc. ~30 in total
- E.g.
 - SGESV: Single prec, general matrix solver (solves $\mathbf{A} \mathbf{x} = \mathbf{b}$)

LU factorisation



- LU factorisation
 - call `SGETRF(M, N, A, LDA, IPIV, INFO)`
 - does an in-place LU factorisation of M by N matrix A
 - we will always consider the case $M = N$
 - A can actually be declared as `REAL A(NMAX,MMAX)`
 - routine operates on $M \times N$ submatrix
 - must tell the library the Leading Dimension of A , ie set `LDA=NMAX`
 - `INTEGER IPIV(N)` returns row permutation due to pivoting
 - error information returned in the integer `INFO`

Solving: Forward/backward substitution



- Forward / backward substitution
 - call `SGETRS(TRANS,N,NRHS,A,LDA,IPIV,B,LDB,INFO)`
 - expects a factored A and `IPIV` from previous call to `SGETRF`
 - solves for multiple right-hand-sides, ie B is $N \times NRHS$
 - we will only consider `NRHS=1`, ie RHS is the usual vector \underline{b}
 - solution x is returned in \underline{b} (ie original \underline{b} is destroyed)
- Options exist for precise form of equations
 - specified by character variable `TRANS`
 - 'N' (Normal), 'T' (Transpose) or 'C' (hermitian Conjugate)

$$\begin{array}{ccc} \text{N} & \text{T} & \text{C} \\ \downarrow & \downarrow & \downarrow \\ \mathbf{A} \underline{x} = \underline{b} & \mathbf{A}^T \underline{x} = \underline{b} & \mathbf{A}^+ \underline{x} = \underline{b} \end{array}$$

Sparse matrices

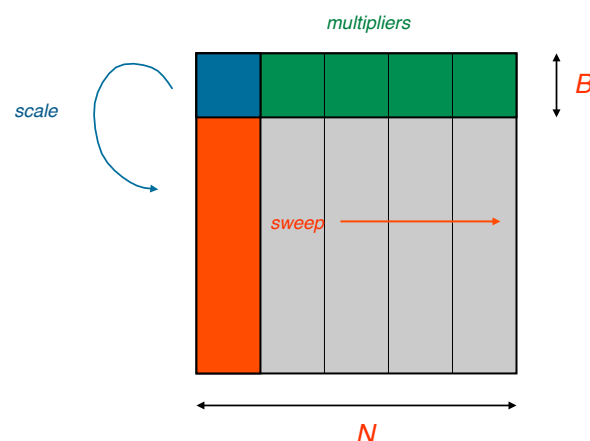
epcc

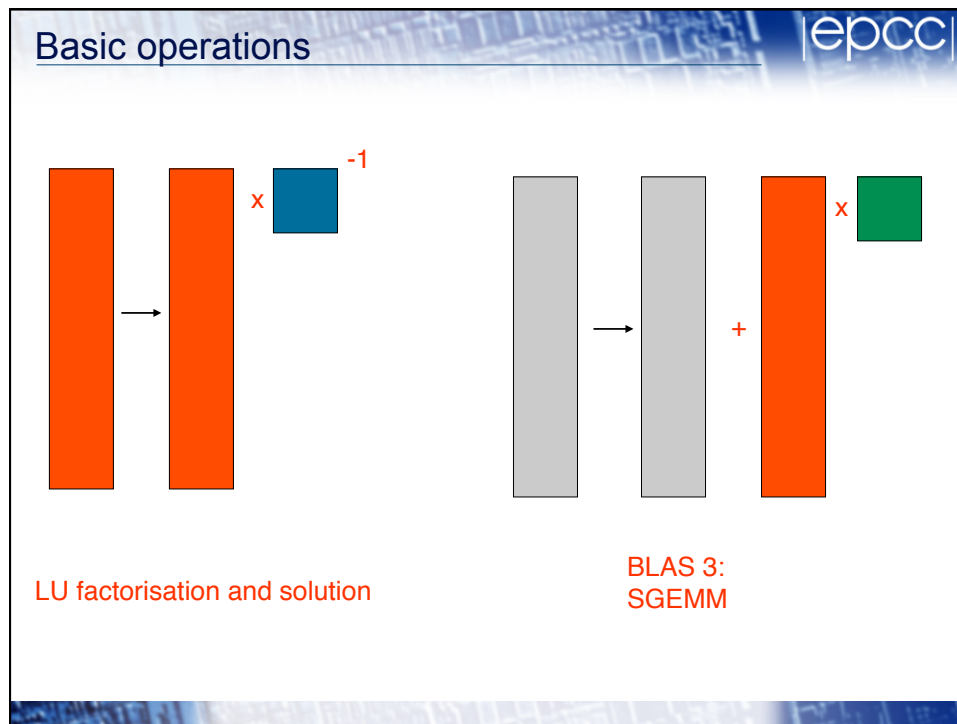
- Direct methods easiest for structured matrices
 - eg tridiagonal, pentadiagonal, block-diagonal
 - support in LAPACK for some well-structured sparse cases
- General sparse matrices
 - difficult to code efficiently due to “fill-in”
 - LU factors will have non-zero entries even where A was zero
 - some specialist libraries, eg Harwell Sparse Matrix Library
- In general, iterative methods are used
 - see later

Blocked linear algebra

epcc

- LAPACK achieves performance by *blocking*
- Operate on $B \times B$ sub-matrices and not scalars





Optimal blocking size

- Value of B for which SGEMM runs fastest
 - clearly very architecture dependent
 - both $B=1$ and $B=N$ map to original LINPACK implementation!
- Can be tackled by self optimising libraries
 - eg ATLAS library has a subset of LAPACK routines
 - this includes LU factorisation

epcc

Parallelisation



- Clear opportunities for parallelism
 - multiple independent **saxpy** or **SGEMM** operations
 - major problem is load balance, on four processors

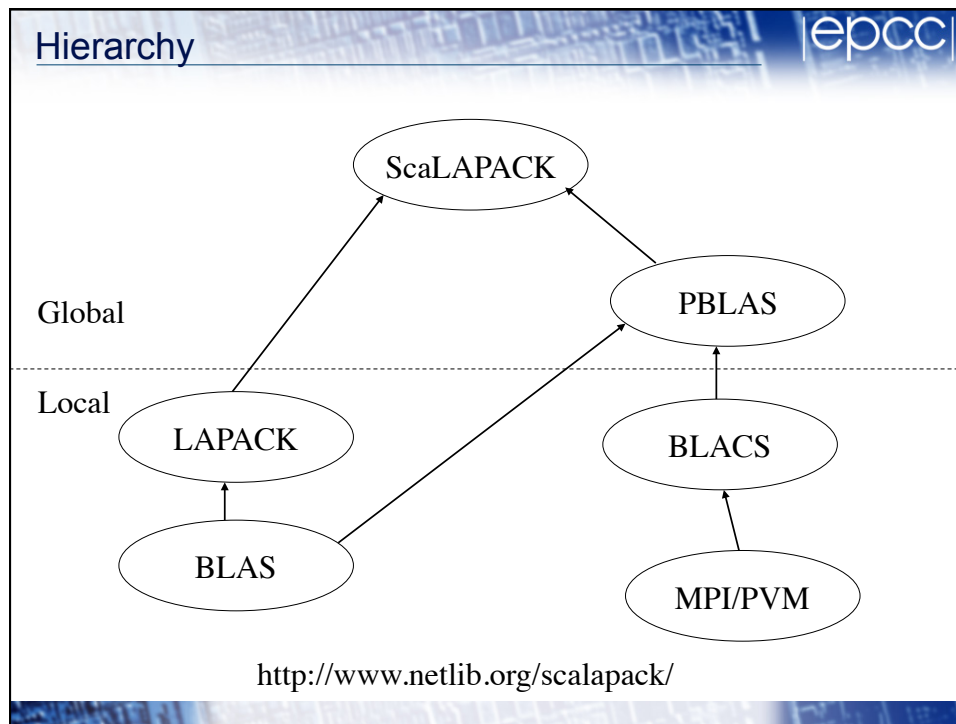


- No block distribution is appropriate
 - must use block-cyclic in at least one dimension

ScaLAPACK introduction

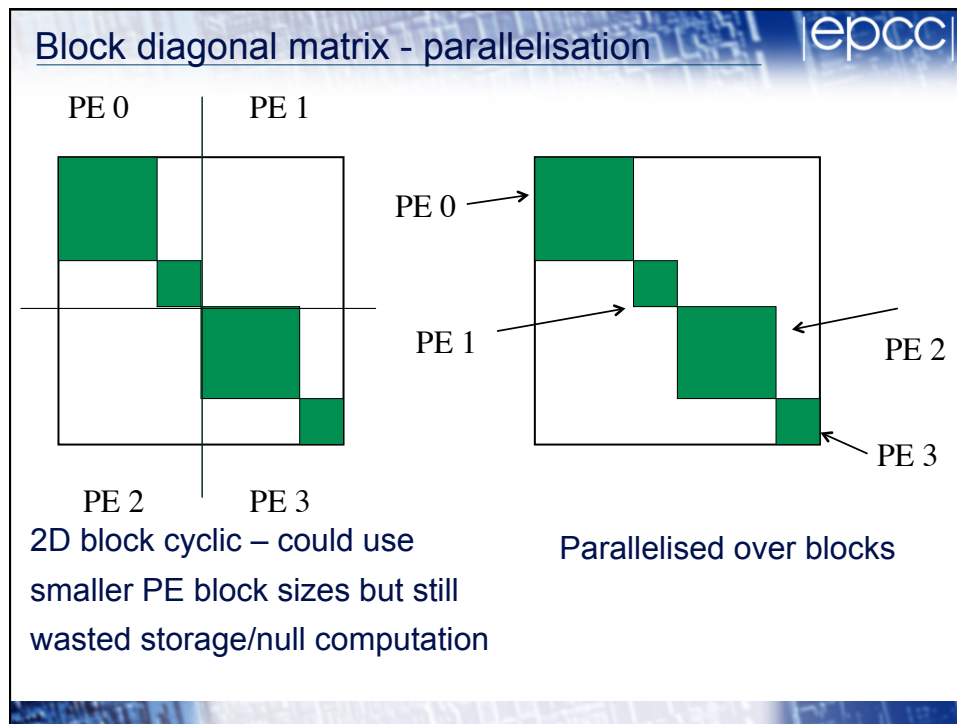


- ScaLAPACK allows us to run LAPACK-like routines *in parallel*
- ScaLAPACK routines written to resemble equivalent LAPACK routines
 - e.g. dgesv -> pdgesv
- Assumes matrices are laid out in 2D block-cyclic form
- Built on top of
 - PBLAS (distributed memory version of Level 1,2 and 3 BLAS)
 - BLACS (Basic Linear Algebra Communication Subprograms)
 - Lapack (Linear algebra library)
 - MPI/PVM/...
- As with LAPACK, written in Fortran 77 but interfaces to Fortran 90/C/C++



Parallelisation of code epcc

- ScaLAPACK is basically a parallel version of LAPACK with the introduction of a few extra routines
 - E.g. matrix transposition
 - Some LAPACK routines have been improved to help with scalability
- Large sparse matrices: use ARPACK or parallel equivalent (P_ARPACK)
- Large dense matrices: Need to consider carefully how to parallelise
 - E.g. finding eigenvalues of a large block diagonal matrix
 - With ScaLAPACK using a 2D block-cyclic distribution, most processors would have nothing to do
 - Probably best just to distribute individual blocks to processors by hand and employ LAPACK on each local block
 - Threaded BLAS implementations exist to exploit further parallelisation
 - Can also make use of accelerated libraries



ScaLAPACK

- ScaLAPACK follows similar format to MPI
 - BLACS “context” being the equivalent of an MPI communicator
 - Need several set-up and finalise routines
- Matrices/vectors completely distributed over processors and described using an *array descriptor*
 - Set up using DESCINIT
- Set up a 2D processor grid
 - Routines provided to determine processor position in grid and local matrix size
- Data distributed in a “block cyclic” fashion with block size (referred to as “blocking factor”)
- <http://acts.nersc.gov/scalapack/hands-on/datadist.html>

2D Block cyclic

epcc

- Situation can be simple – e.g. 8x8 matrix with 2x2 processor grid...

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3

- ...or can be more complicated...

2D block cyclic distribution - example

epcc

Global matrix size: 9x9

Block size: 2x2

No. procs: 6

Processor grid: 2x3

Local matrix sizes

proc 0: 5x4

proc 1: 5x3

proc 2: 5x2

proc 3: 4x4

proc 4: 4x3

proc 5: 4x2

0	0	1	1	2	2	0	0	1
0	0	1	1	2	2	0	0	1
3	3	4	4	5	5	3	3	4
3	3	4	4	5	5	3	3	4
0	0	1	1	2	2	0	0	1
0	0	1	1	2	2	0	0	1
3	3	4	4	5	5	3	3	4
3	3	4	4	5	5	3	3	4
0	0	1	1	2	2	0	0	1

ScaLAPACK: Example solving $Ax=b$



```
...
n=64; nrhs=1; nprow=2; npc=3; mb=nb=2;
...
CALL BLACS_GET(-1,0,ctxt)
CALL BLACS_GRIDINIT(ctxt, 'Row-major', nprow, npc)
CALL BLACS_GRIDINFO(ctxt, nprow, npc, myrow, mycol)
...
num_rows_local = NUMROC(n, nb, myrow, 0, nprow)
num_cols_local = NUMROC(n, nb, mycol, 0, npc)
...
Allocate(A(num_rows_local, num_cols_local), b(num_rows_local),
ipiv(num_rows_local+nb))
...
IF(MYROW.EQ.-1) Skip computation!
...
CALL DESCINIT(desca, n, n, mb, nb, rsrc, csrc, ctx, llda, info)
CALL DESCINIT(descb, n, nrhs, nb, nrhs, rsrc, csrc, ctx, ldb, info)
...
call PDGETRF(n, n, A, 1, 1, desca, ipiv, info)
call PDGETRS('N', n, 1, A, 1, 1, desca, ipiv, b, 1, 1, descb, info)
...
WRITE(*,*) ...Results.....
...
CALL BLACS_EXIT(0)
```

PLASMA and MAGMA



- **PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures)**
 - Aims to overcome “fork-join” model of BLAS
 - Uses some BLAS libraries
 - Dynamic runtime scheduling (QUARK) uses DAG (Directed Acyclic Graphs)
 - Uses small “tiles” of data: Finer grain parallelism and much more cache aware than LAPACK
- **MAGMA (Matrix Algebra on GPU and Multics Architectures)**
 - Designed for hybrid/heterogeneous architectures
 - Designed in particular for accelerators/GPUs
 - CUDA (all routines), OpenCL (some), Intel Xeon Phi (a few)
 - Some routines can use multiple GPUs
 - Algorithms avoid comms as much as possible
 - Dynamic or static scheduling

Exercise



- Code an **$A\mathbf{x}=\mathbf{b}$** solver by hand
 - Using LU factorisation followed by forward/backward substitution
 - with NO pivoting
- Replace hand coded routines with calls to LAPACK
 - library routines use pivoting
 - very well optimised (except Java!)
 - compare accuracy and performance
- Run equivalent calculation in ScaLAPACK
 - use different block sizes
 - determine best block size

Calling Fortran libraries from C



- A number of issues
 - storage order of matrices
 - calling by reference / calling by value
 - character variables ← *System Dependent*
 - subroutine names ← *System Dependent*
- C arrays are transposed w.r.t. Fortran
 - could choose to store all matrices in transpose format
 - but may simply be able to specify **TRANS= 'T'** where appropriate
- Fortran *always* expects pass-by-reference
 - must assign C constants to variables, eg **one = 1;**
 - pass the pointer **&one** to the subroutine

Calling LAPACK from C



- Fortran

```
call SGETRS (TRANS,N,NRHS,A,LDA,IPIV,B,LDB,INFO)
```

- Easiest to write a wrapper for C, e.g:

```
int sgetrs(char trans, int n, int nrhs,  
float *a, int lda, int *ipiv, float *b, int ldb)  
{  
    int info;  
    sgetrs_(&trans, &n, &nrhs, a, &lda, ipiv, b, &ldb, &info);  
    return(info);  
}  
...  
info = sgetrs('t', n, 1, &(a[0][0]), NMAX, ipiv, x, NMAX);
```

C requires the following libs when linking:

```
-llapack -lblas -lpqftnrtl -lrt
```