Numerical Algorithms for HPC

Parallel linear algebra libraries

epcc

1

Overview

- Parallel dense linear algebra library: ScaLAPACK
- PETSc library for PDEs

epcc

2

Parallel dense linear algebra libraries

- Previously we introduced the serial library **LAPACK**
- This lecture: will introduce parallel equivalent **ScaLAPACK**
- ScaLAPACK is basically a parallel version of LAPACK with a few extra routines
  - E.g. matrix transposition
  - Some LAPACK routines have been improved to help with scalability
- Large sparse matrices: instead use ARPACK or parallel equivalent (P_ARPACK)

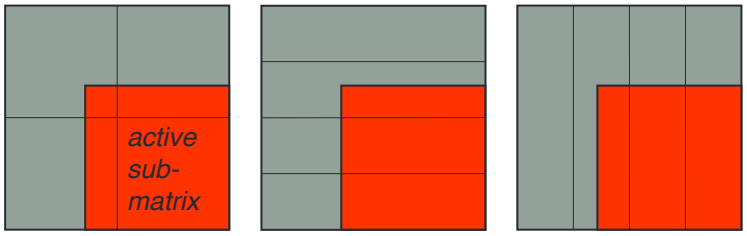
Dense Linear Algebra in parallel

- Consider LU factorisation
  - equivalently Gaussian Elimination
- Recall main operations were
  - `saxpy/daxpy` – for subtracting one row from another
  - `sgemm/dgemm` – matrix-matrix multiplication – scaling
- Algorithm works on progressively smaller sub-matrix
  - More zeros on each row as you move down the matrix

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{bmatrix} \rightarrow \begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & a'_{43} & a'_{44} \end{bmatrix} \rightarrow \dots$$

Parallelisation

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

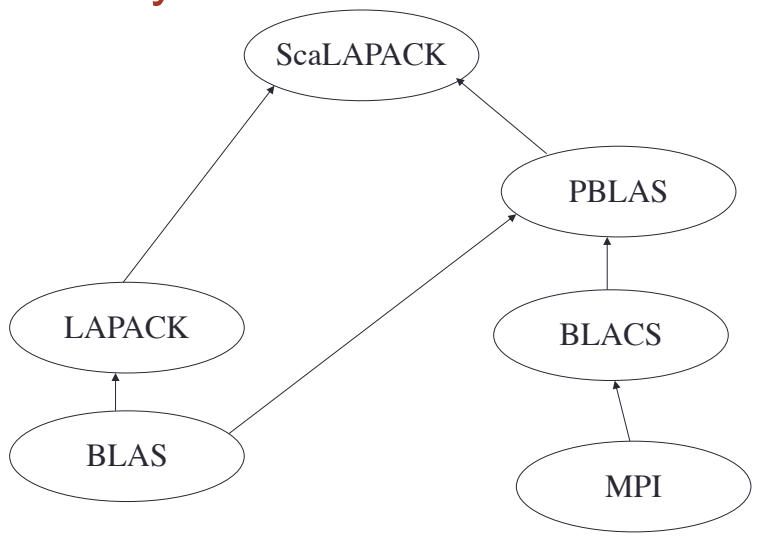


- No simple block distribution is appropriate
  - But clear we must use block-cyclic in at least one dimension

ScaLAPACK introduction

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

Hierarchy



epcc http://www.netlib.org/scalapack/

7

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 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")
  - ScaLAPACK describes this as "complicated but efficient"

epcc 8

8

2D Block cyclic

- Situation can be simple – e.g. 8×8 matrix with 2×2 processor grid…

P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1
P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1
P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1
P_0	P_0	P_0	P_0	P_1	P_1	P_1	P_1
P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3
P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3
P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3
P_2	P_2	P_2	P_2	P_3	P_3	P_3	P_3

- …or by making processor blocks smaller can be more complicated…

P_0	P_0	P_1	P_1	P_0	P_0	P_1	P_1
P_0	P_0	P_1	P_1	P_0	P_0	P_1	P_1
P_2	P_2	P_3	P_3	P_2	P_2	P_3	P_3
P_2	P_2	P_3	P_3	P_2	P_2	P_3	P_3
P_0	P_0	P_1	P_1	P_0	P_0	P_1	P_1
P_0	P_0	P_1	P_1	P_0	P_0	P_1	P_1
P_2	P_2	P_3	P_3	P_2	P_2	P_3	P_3
P_2	P_2	P_3	P_3	P_2	P_2	P_3	P_3

epcc

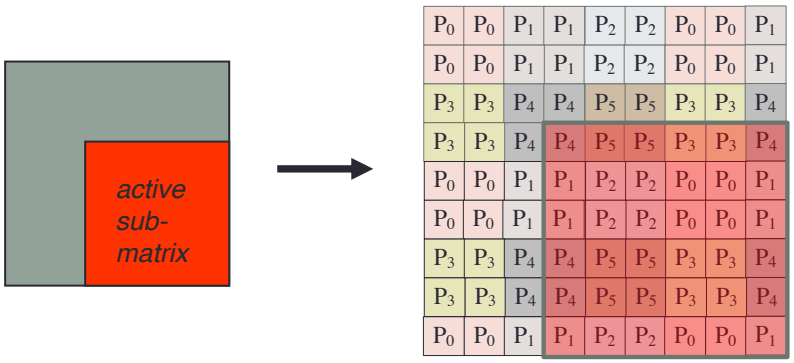
2D block cyclic distribution - example

- Global matrix size: 9×9
- Block size: 2×2
- No. processors: 6
- Processor grid: 2×3
- Local matrix sizes
- P_0: 5 x 4 = 20
- P_1: 5 x 3 = 15
- P_2: 5 x 2 = 10
- P_3: 4 x 4 = 16
- P_4: 4 x 3 = 12
- P_5: 4 x 2 = 8
- Routines exist to calculate local sizes!

P_0	P_0	P_1	P_1	P_2	P_2	P_0	P_0	P_1
P_0	P_0	P_1	P_1	P_2	P_2	P_0	P_0	P_1
P_3	P_3	P_4	P_4	P_5	P_5	P_3	P_3	P_4
P_3	P_3	P_4	P_4	P_5	P_5	P_3	P_3	P_4
P_0	P_0	P_1	P_1	P_2	P_2	P_0	P_0	P_1
P_0	P_0	P_1	P_1	P_2	P_2	P_0	P_0	P_1
P_3	P_3	P_4	P_4	P_5	P_5	P_3	P_3	P_4
P_3	P_3	P_4	P_4	P_5	P_5	P_3	P_3	P_4
P_0	P_0	P_1	P_1	P_2	P_2	P_0	P_0	P_1

epcc

2D block cyclic applied to LU factorisation



- This decomposition allows a reasonable load balance

ScaLAPACK: Example solving Ax=b

```
…
n=64; nrhs=1; nprow=2; npcol=3; mb=nb=2;
…
CALL BLACS_GET(-1,0,ctxt)
CALL BLACS_GRIDINIT(ctxt, 'Row-major', nprow, npcol)
CALL BLACS_GRIDINFO(ctxt, nprow, npcol, myrow, mycol)
…
num_rows_local = NUMROC(n, nb, myrow, 0, nprow)
num_cols_local = NUMROC(n, nb, mycol, 0, npcol)
…
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 ,nbrhs, rsrc, csrc, ctx, lldb, 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)
```


PETSc

The Portable, Extensible Toolkit for Scientific computing

https://www.mcs.anl.gov/petsc/

"PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/Tao) also contains the Tao optimization software library."

Current PETSc version is 3.16

epcc

13

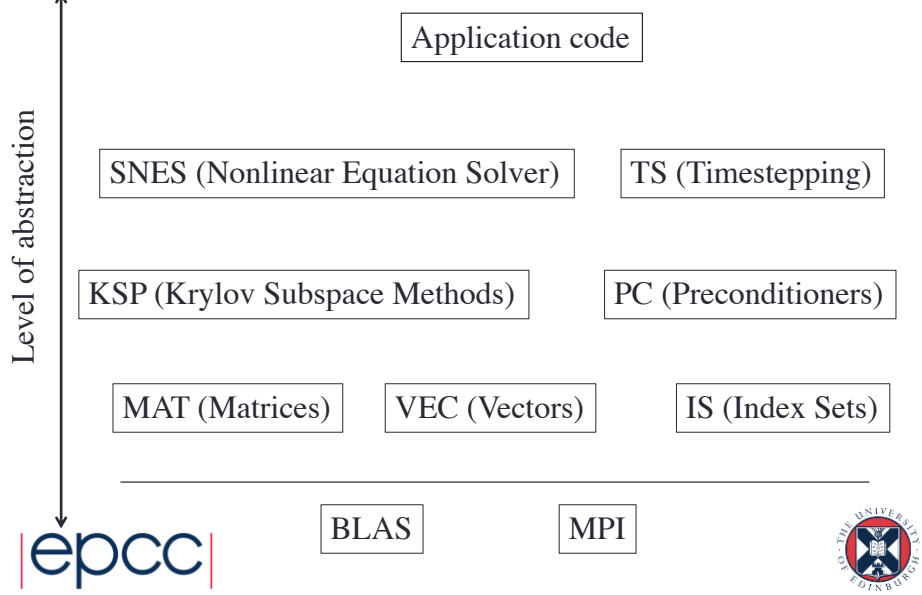
PETSc use in codes

- There are many, but notable examples
  - SLEPc - Scalable Library for Eigenvalue Problems
  - Fluidity - a finite element/volume fluids code
  - libMesh - adaptive finite element library
  - FEniCS and Firedrake - sophisticated Python based finite element simulation package
  - …

epcc

14

PETSc Hierarchy



15

Design characteristics

- Shared nothing – only distributed memory programming
- Object-oriented
- Callable from C, C++, Fortran, Python
- Same code runs in serial or parallel
- Either real or complex scalar datatype (library build option, your code doesn't look different)

16

Example

```c
#include <petsc.h>

int main(int argc, char **argv)
{
    Vec x;
    PetscInt n = 100;
    PetscScalar y;
    PetscInitialize(&argc, &argv,
                    PETSC_NULL,
                    PETSC_NULL);
    VecCreate(PETSC_COMM_WORLD, &x);
    VecSetSizes(x, PETSC_DECIDE, n);
    VecSetFromOptions(x);
    VecSetRandom(x, PETSC_NULL);
    VecDot(x, x, &y);

    PetscPrintf(PETSC_COMM_WORLD,
                "Dot product is %g\n", y);
    VecDestroy(&x);
    PetscFinalize();
    return 0;
}
```

```fortran
program main
#include "finclude/petscdef.h"
  use petsc
  implicit none
  Vec :: x
  PetscInt :: n = 100
  PetscScalar :: y
  integer :: ierr
  character(len=12) :: tmp
  call PetscInitialize(PETSC_NULL_CHARACTER, &
       ierr)
  call VecCreate(PETSC_COMM_WORLD, x, ierr)
  call VecSetSizes(x, PETSC_DECIDE, n, ierr)
  call VecSetFromOptions(x, ierr)
  call VecSetRandom(x, PETSC_NULL_OBJECT, ierr)
  call VecDot(x, x, y, ierr)

  write(tmp, '(F10.5)')y
  call PetscPrintf(PETSC_COMM_WORLD, &
       'Dot product is'//trim(tmp)//'\n', &
       ierr)
  call VecDestroy(x, ierr)
  call PetscFinalize(ierr)
end program main
```

epcc

17