Parallel Subroutine Libraries

- FFTW
- PetSc
- Scalapack, ParPack, etc.

Parallel Subroutine Libraries

- Scientific Applications can often be reduced to a combination of atomic operations. These atomic operations may be linear algebra operations, eigenvalue problems, interpolation, integration, differentiation, integral transforms, solutions of differential equations, etc.

- Libraries of subroutines for carrying out these atomic operations on sequential computers have been developed and used heavily, e.g., BLAS, LAPACK, etc.

- Parallel libraries allow scientists and engineers to solve problems without having to develop parallel algorithms for standard atomic processes.

Parallel Subroutine Libraries

• In most scientific applications, these atomic processes take up a large fraction of the CPU time required.

◦ Using parallel libraries is efficient on n processors if the atomic processes for which parallel libraries are to be used account for a fraction $f \gg 1 - 1/n$ of the total CPU time. (see Amdahl's law)

◦ If the algorithm can be rewritten so that the CPU time required on one processor is larger, but the fraction f is closer to unity, use of a large number of processors will result in improved efficiency.

FFTW: Fastest Fourier Transform in the West

- Fast Fourier Transforms.
  - Algorithm.
  - Inherent Parallelisms.
  - Shared Memory Parallelization.
  - Distributed Memory Parallelization.

Fourier Transform

- The integral for Fourier transform is approximated by a sum.

$$
\begin{aligned}
f(x) &= \int_{-\infty}^{\infty} \frac{dk}{2\pi} g(k) e^{-ikx} \simeq \int_{-k_{max}}^{k_{max}} \frac{dk}{2\pi} g(k) e^{-ikx} \\
&\simeq \frac{\Delta k}{2\pi} \sum_{k=-k_{max}}^{k_{max}} g(k) e^{-ikx} \simeq \frac{1}{NL} \sum_{-k_{max}}^{k_{max}+\Delta k} g(k) e^{-ikx} \\
&= \frac{1}{NL} \sum_{m=-N/2+1}^{N/2} g(\Delta k\, m) e^{-i\Delta k m x} \\
f(nL) &= \frac{1}{NL} \sum_{m=-N/2+1}^{N/2} g(\Delta k\, m) e^{-i2\pi m n/N} \qquad (1)
\end{aligned}
$$

Fast Fourier Transform

$$
\begin{aligned}
f(nL) &= \frac{1}{NL}\sum_{m=-N/2+1}^{-1} g(\Delta k\, m)e^{-i2\pi mn/N} \\
&\quad + \frac{1}{NL}\sum_{m=0}^{N/2} g(\Delta k\, m)e^{-i2\pi mn/N} \\
&= \frac{1}{NL}\sum_{m=N/2+1}^{N-1} g(\Delta k(m-N))e^{-i2\pi mn/N} \\
&\quad + \frac{1}{NL}\sum_{m=0}^{N/2} g(\Delta k\, m)e^{-i2\pi mn/N} \\
&= \frac{1}{NL}\sum_{m=0}^{N-1} G(\Delta k\, m)e^{-i2\pi mn/N} \qquad (2)
\end{aligned}
$$

Fast Fourier Transform

• If $N = 2^j$, with j an integer, the summation can be optimized significantly. At the first level, the summation can be split into two smaller sums.

$$\begin{aligned} f(n) &= \frac{1}{N}\sum_{m=0}^{N-1} G(m)e^{-i2\pi mn/N} \\ &= \frac{1}{N}\sum_{m=0}^{N/2-1} G(2m)e^{-i2\pi mn/(N/2)} \\ &\quad + \frac{1}{N}e^{-i2\pi n/N}\sum_{m=0}^{N/2-1} G(2m+1)e^{-i2\pi mn/(N/2)} \end{aligned} \tag{3}$$

• The sum can be divided into smaller and smaller parts, till we are left with a two element transform.

Fast Fourier Transform

• The FFT algorithm involves two steps: reshuffling of the input array, and, multiplication by phase factors as we go from two element transforms to the full N element transform.

◦ The transform can be calculated "in place", f and G do not require two arrays.

◦ The inverse transform can also be computed in a similar fashion.

◦ The shuffling required is mapping each $G(m)$ to a $G(l)$ where l is the integer obtained by bit inversion of m.

Parallelisms

• FFTs are fast enough to allow computation of one dimensional FTs of size 10^7 in a few seconds on modern day processor, hence the challenge of paralleliation is mainly for multi-dimensional systems.

• If data is to be used in both real space as well as k-space, a lot of communications are required.

Parallelisms: Shared Memory

• The core part of the FFT routine is the following loop. This can be parallelized using OpenMP on shared memory computers.

```
DO j1=0,n-1
   tmpr=wr*f(j,,j1)-wi*fi(j,j1)
   tmpi=wr*fi(j,j1)+wi*f(j,j1)
   f(j,j1)=f(i,j1)-tmpr
   fi(j,j1)=fi(i,j1)-tmpi
   f(i,j1)=f(i,j1)+tmpr
   fi(i,j1)=fi(i,j1)+tmpi
END DO
```

Parallelisms: Distributed Memory

- Domain decomposition can be achieved by dividing the rectangular cube into parallel slabs.

  - Fourier transform can be done locally along directions orthogonal to the direction in which the cube is divided.

  - Data exchange is needed for doing the Fourier transform along the direction in which the cube has been divided.

FFTW (www.fftw.org)

- Optimizes at a local level by using the cache and processor optimizations like SSE, SSE2, 3D-NOW, etc.

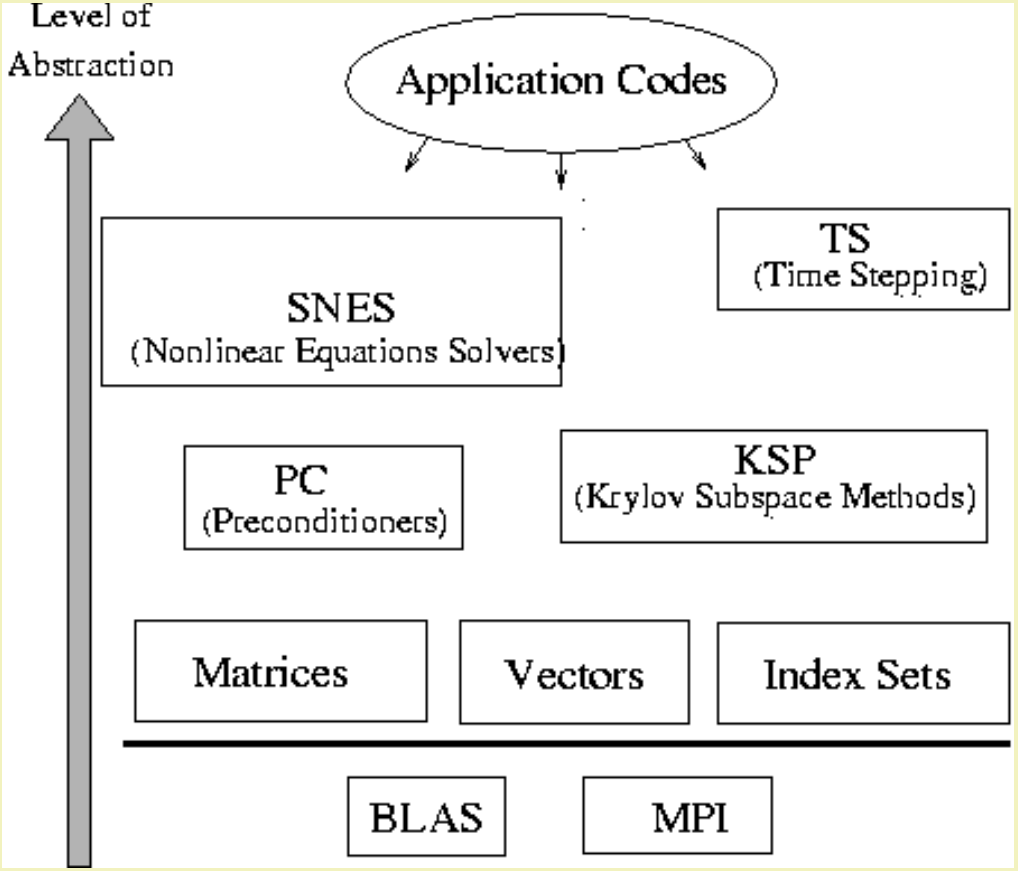
- Can make a “plan”, the fastest way of computing FFT for a given array size on the given processor. This is very useful is FFTs are to be calculated repeatedly.

- Shared memory parallelization is available using `pthreads` as well as OpenMP. (Versions 2.x, 3.0.1)

- MPI based parallelization is available in older (2.x) versions.

PETSC

- The Portable, Extensible Toolkit for Scientific Computing (PETSC) is available at http://www.mcs.anl.gov/petsc



PETSC

• **Vectors:** Provides the vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations.

• **Matrices:** A large suite of data structures and code for the manipulation of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.

• **Pre-Conditioners:** A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and (through BlockSolve95) ILU(0) and ICC(0).

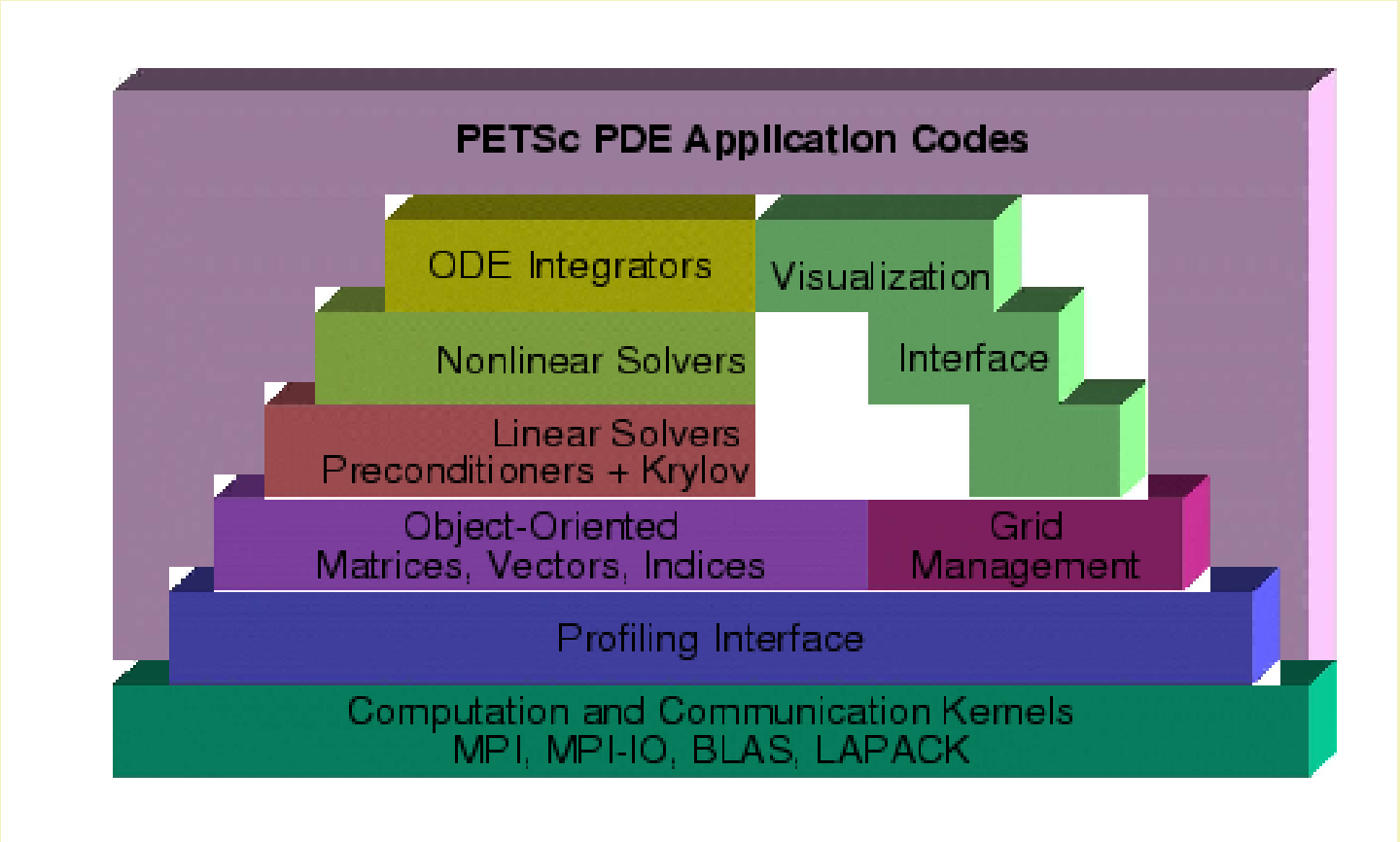
PETSC

• **Krylov Subspace Methods:** Parallel implementations of many popular Krylov subspace iterative methods. All are coded so that they are immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.

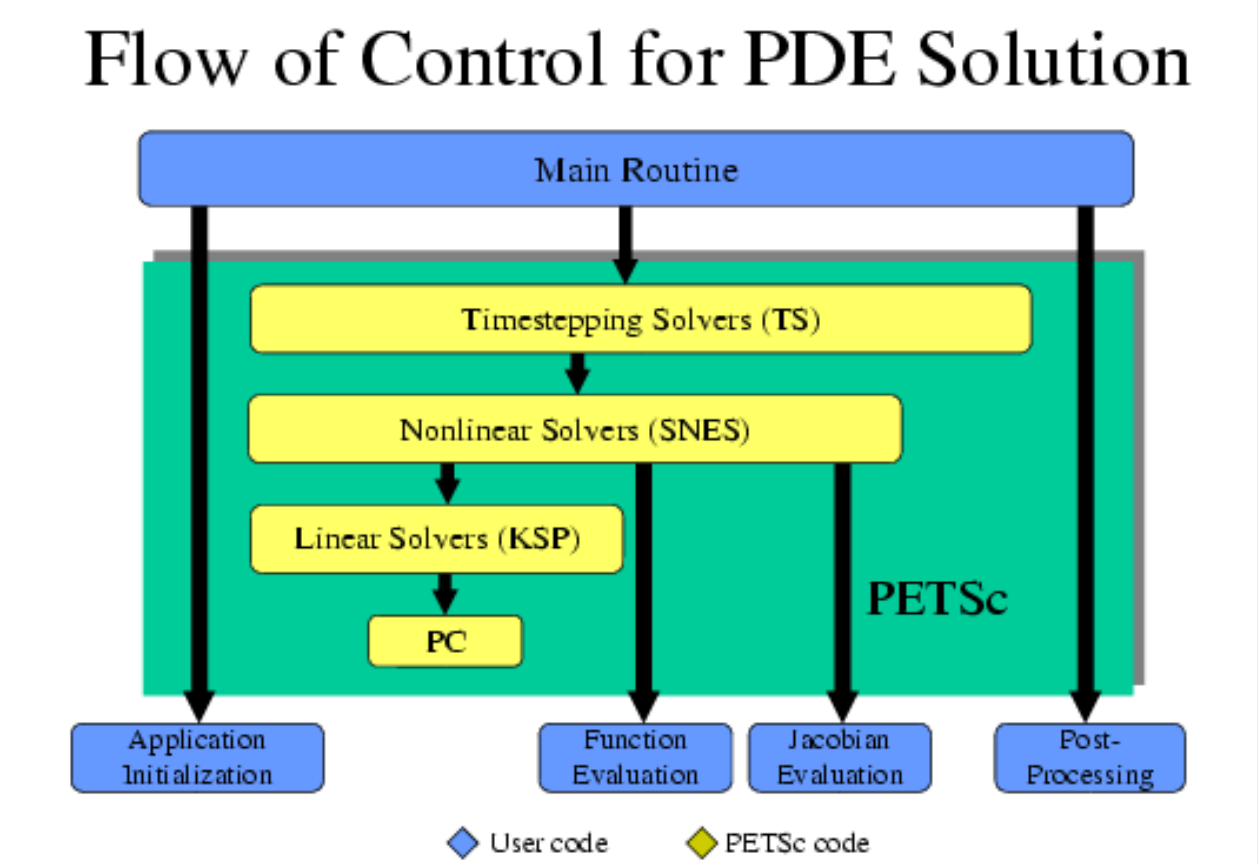
• Data-structure-neutral implementations of Newton-like methods for nonlinear systems. Includes both line search and trust region techniques with a single interface. Employs by default the above data structures and linear solvers. Users can set custom monitoring routines, convergence criteria, etc.

• **Time Stepping:** Code for the time evolution of solutions of PDEs. In addition, provides pseudo-transient continuation techniques for computing steady-state solutions.

PETSC


PETSc PDE Application Codes
ODE Integrators
Visualization
Nonlinear Solvers
Interface
Linear Solvers
Preconditioners + Krylov
Object-Oriented
Matrices, Vectors, Indices
Grid
Management
Profiling Interface
Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK

PETSC


Flow of Control for PDE Solution
Main Routine
Timestepping Solvers (TS)
Nonlinear Solvers (SNES)
Linear Solvers (KSP)
PC
PETSc
Application Initialization
Function Evaluation
Jacobian Evaluation
Post-Processing
User code
PETSc code

PETSC

- PETSC has been used in a wide variety of applications.

- A CFD code has been parallelized using PETSC and is shown to scale almost linearly up to 1024 processors on T3E.

Linear Algebra Packages

• A large number of parallel libraries are available, these are at different stages of development. These are parallel extension of the LAPACK project and are available from http://www.netlib.org

- **ScaLAPACK:** Dense and band matrix software.
- **PARPACK & ARPACK:** Large sparse eigenvalue software.
- **CAPSS & MFACT:** Sparse direct systems software.
- **ParPre:** Preconditioners for large sparse iterative solvers.