Advanced Computational Software

Scientific Libraries: Part 2

Blue Waters Undergraduate Petascale Education Program

May 29 – June 10 2011

Outline

- Quick review
- Fancy Linear Algebra libraries
  - ScaLAPACK
  -PETSc
  -SuperLU
- Fancy Differential Equation solvers
  - SUNDIALS
- Optimization libraries –TAO, SAMRAI
- Trilinos





Quick review

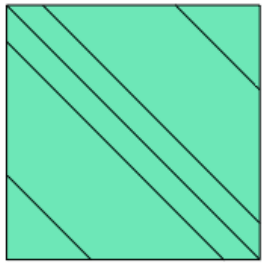
- What is the most efficient way do my mathematical operation?

  Lazy answer #2: It depends!

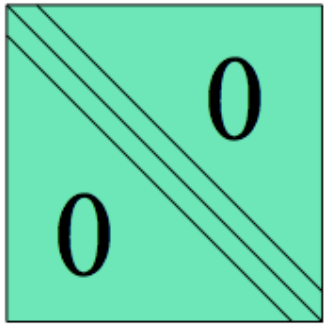
- Don't reinvent the wheel
- Read the documentation and do examples
- **Understand your problem!!**

Quick Matrix Review

- General Matrix – Nothing too special going on with it
- Banded Matrix – A lines of data going diagonally down the matrix.



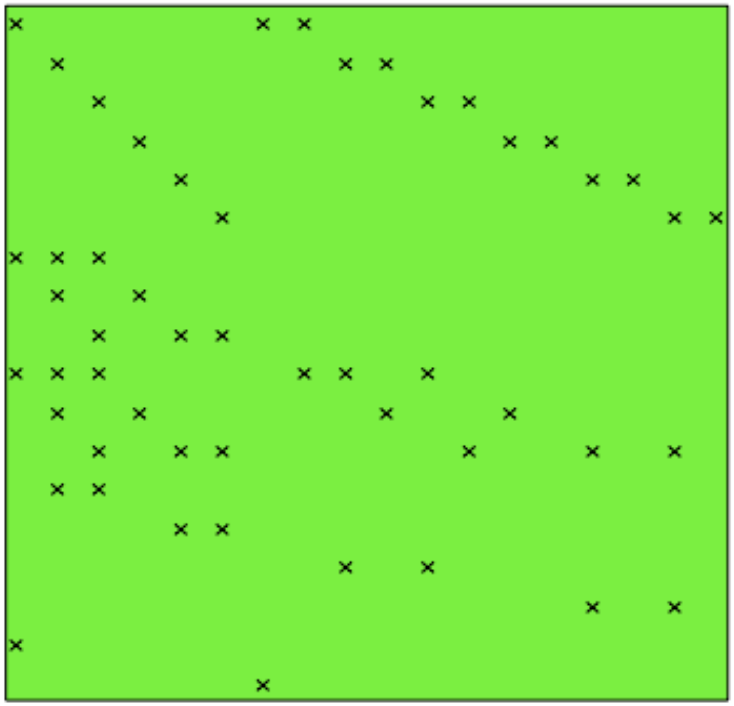
- Tri-diagonal matrix- A 3-width banded matrix



- Symmetric matrix- $a_{i,j} = a_{j,i}$

Another Quick Matrix review

- A matrix that is mostly empty. A good rule of thumb is approximately 90% - 95% empty.
- The opposite of a sparse matrix is a dense matrix.



Explanation of Lazy Answer #2

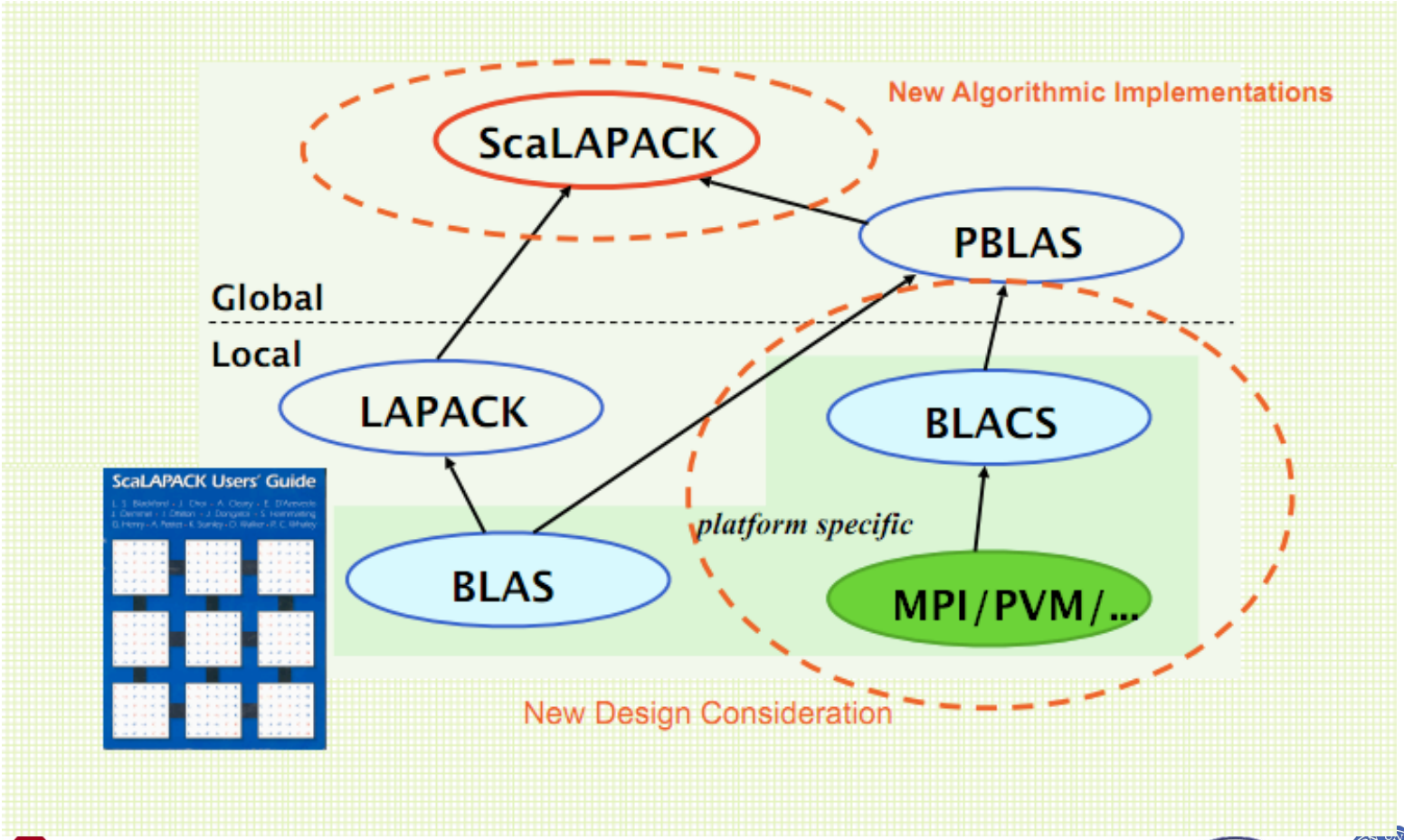
- It depends on the problem.
- If you fully understand the problem, you can narrow down what software will work best for you relatively easily.

Parallel Linear Algebra Libraries

- ScaLAPACK – Scalable LAPACK
- PETsC – Portable Extendable Toolkit for Scientific Computing
- SuperLU – Performs LU factorizations in a super way

Scalapack

- Use with dense matrices



Four Basic Steps

- 1. Initialize the process grid

```
CALL SL_INIT( ICTXT, NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
```

- 2. Distribute the matrix on the process grid – User's responsibility
  – Set array descriptors
- 3. Call ScaLAPACK routine – Read documentation
- 4. Release the process grid

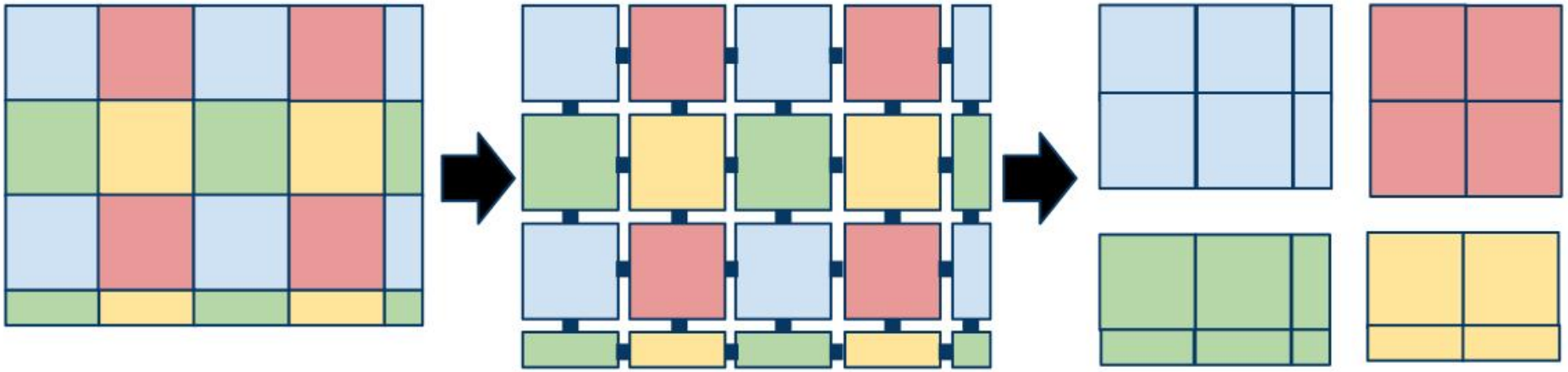
```
CALL BLACS_GRIDEXIT( ICTXT )
CALL BLACS_EXIT( 0 )
```





Data Representation

2-dimensional Block Cyclic distribution



ScaLAPACK will **NOT** distribute your data for you.
You must initialize it that way!!
Thus it is very difficult to write into existing serial code.

PETSc

- Portable Extendable Toolkit for Scientific Computing
- Programming flexible and extendable, implementation requires experimentation
- Heavily utilizes MPI
- Object oriented Matrix creation

```
MatCreate(MPI_Comm, Mat *)
MatSetSizes(Mat, int m, int n, int M, int N)
MatSetType(Mat, MatType typeName)
MatSetFromOptions(Mat)
```

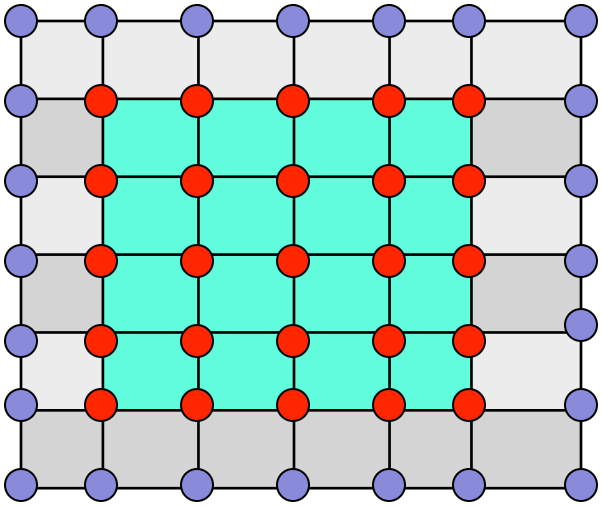


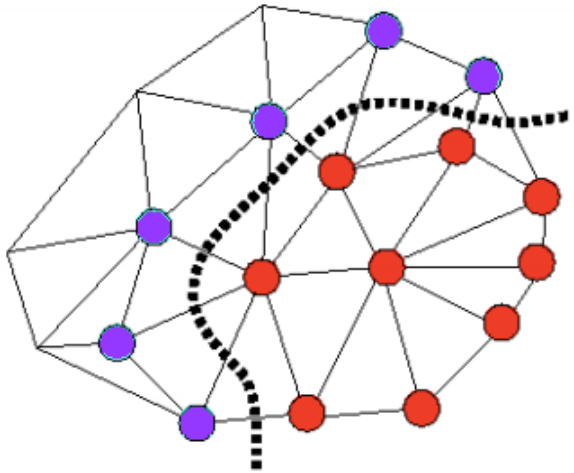


Data Representation

- Object oriented matrix storing means the user doesn't need to worry about it.
- Algorithmically stores data. User has some control.
- Ultimately the amount of control is up to the user- Stencils

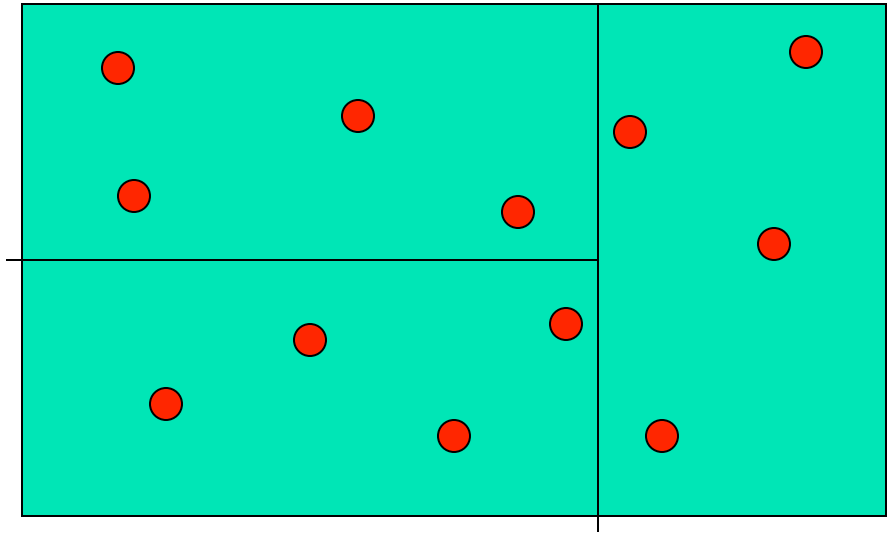
  -Stencils are a pattern of local points and ghost points





Advantages of PETSc

- Many matrix configuration options
- Object oriented approach
- Load balancing interface through Zoltan
  - -Mesh refinement
  - -Data migration
  - -Matrix Ordering



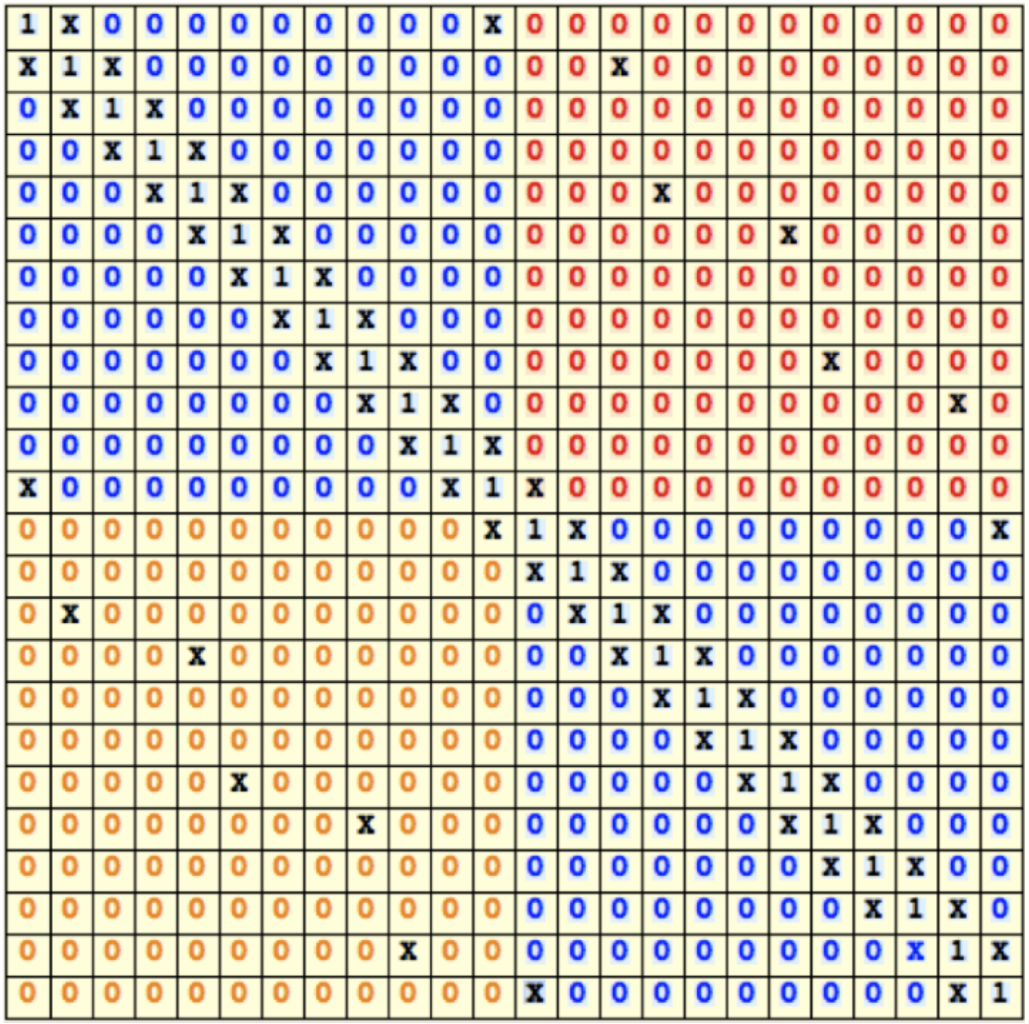
PETSc Example

- PETSc parallel Hello World
- PETSc Linear Solver

2	-1									
-1	2	-1								
	-1	2	-1							
		-1	2	-1						
			-1	2	-1					
				-1	2	-1				
					-1	2	-1			
						-1	2	-1		
							-1	2	-1	
								-1	2	

Another advantage of PETSc

- Doesn't necessarily store zeros!

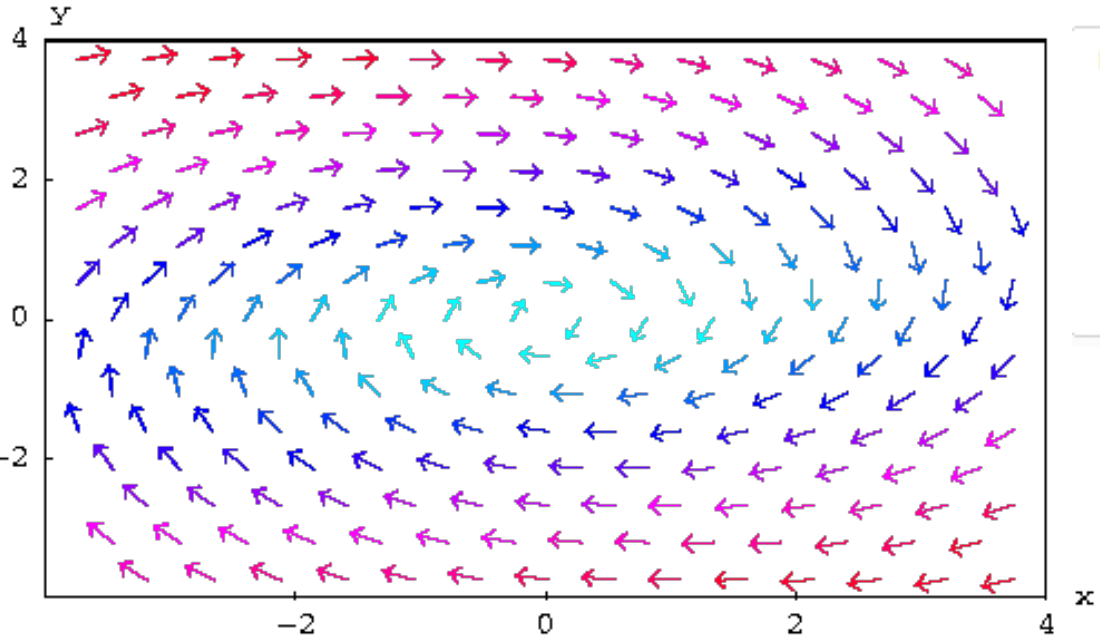


SuperLU

- Sparse direct solver and preconditioner
- Adapted into PETSc, Trilinos, and other major scientific libraries
- Different flavors for sequential, distributed and shared memory machines. Others are optimized for particular architectures

Differential Equations

$$\frac{dx}{dt} = -x + 4y \qquad \frac{dy}{dt} = -3x - y$$



Differential equation solutions:

$$x(t) = \frac{2\,c_2\,e^{-t}\sin\left(2\sqrt{3}\,t\right)}{\sqrt{3}} + c_1\,e^{-t}\cos\left(2\sqrt{3}\,t\right)$$

$$y(t) = c_2\,e^{-t}\cos\left(2\sqrt{3}\,t\right) - \frac{1}{2}\sqrt{3}\,c_1\,e^{-t}\sin\left(2\sqrt{3}\,t\right)$$

SUNDIALS

- **Su**ite of **N**onlinear and **Di**fferential/**Al**gebraic Equation **S**olvers
- Ordinary Differential Equation (ODE) and Differential Algebraic Equation (DAE) integration and solvers
- Utilizes Newton's methods for nonlinear systems
- Created with an emphasis on usability

Optimization Libraries

- SAMRAI - Structured Adaptive Mesh Refinement Application Infrastructure

  – Automatic (yet user-controlled) Adaptive mesh refinement

  – Interfaces for PETSc and SUNDIALS

  –Visualization support through VisIt

- TAO- Toolkit for Advanced Optimization

  - Solves unconstrained, bound constrained and complementary optimization problems

  -Utilizes PETSc data structures





Trilinos

- Does a little bit of everything through **packages**
- Uses Linear Algebra libraries, preconditioners, linear/non-linear solvers, Eigensolvers, Automatic Differentiation solvers, Partitioning/Load balancing packages, mesh generation, and other tools/utilities

Full list of packages available: http://trilinos.sandia.gov/packages/

I hope you like acronyms!!!

Bonuses of Trilinos

- Package interoperability

  - Packages within Trilinos are nearly guaranteed to work together

- Designed to be portable, relatively easy to install and configure
- Lots of supplementary resources

Installing Libraries

1. Learn everything about your architecture – have it written down in front of you.
2. Follow readme very carefully.
3. Run independent tests using example code.

OR

1. Have a system administrator install your library for you, then provide you with a makefile.

Module System

Currently used on most Teragrid resources

Command	Performs
`module list`	Lists the modules currently loaded
`module avail`	Lists all the modules that could be loaded
`module load`	Loads the module for later use
`module unload`	Unloads a module that is unnecessary
`module swap`	Unload one module and replace it with another
`module help`	Receive information on a particular module





What do I do now?

- Many architectures have a unique directory for the library.
- How you link libraries on one machine may not work on another, despite
- Ask your system admin for a sample makefile that runs a piece of example code

EXAMPLE:

The Cray XT6m at Colorado State University's module system.





Parallel Mathematica

- Cluster integration extremely easy, GPGPU integration nearly nonexistent. (NOTE: This is rapidly changing)

Parallelize[*expr*]
evaluates *expr* using automatic parallelization.

```
In[2]:= Parallelize[Prime[Range[10]]]

Out[2]= {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}
```

ParallelTry[*f*, {arg_1, arg_2, ...}]
evaluates $f[arg_i]$ in parallel, returning the first result received.

```
ParallelTry[
 (While[! PrimeQ[p = RandomInteger[{10^100, 10^101}]],]; p) &,
 Range[$ProcessorCount]]

38 783 649 646 337 458 800 780 981 726 428 741 990 486 114 515 015 515 670 631 518 011 716 ↓
 377 413 733 128 654 512 994 059 007 426 811
```

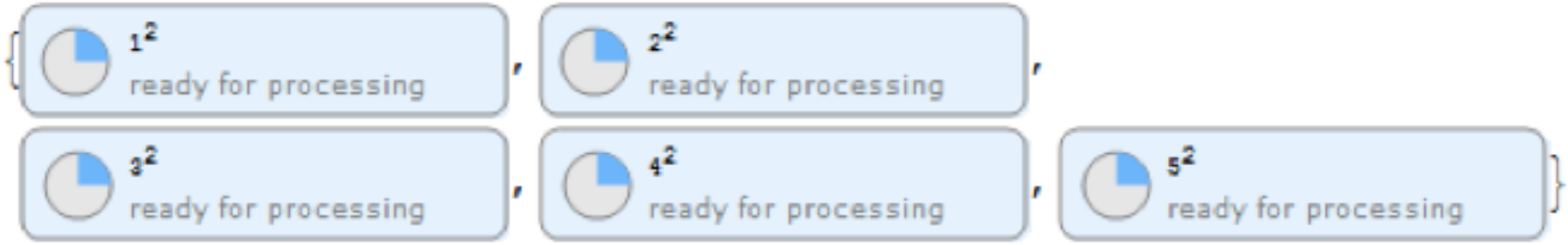
Parallel Mathematica

ParallelSubmit[*expr*]

submits *expr* for evaluation on the next available parallel kernel and returns an EvaluationObject expression representing the submitted evaluation.

```
In[4]:= pids = Function[i, ParallelSubmit[i^2]] /@ {1, 2, 3, 4, 5}
```

Out[4]=



SetSharedVariable[s_1, s_2, ...]

declares the symbols s_i as shared variables whose values are synchronized among all parallel kernels.

```
In[1]:= xs = 0; SetSharedVariable[xs];

In[2]:= ParallelEvaluate[xs++]

Out[2]= {0, 1, 2, 3}

In[3]:= xs

Out[3]= 4
```

Mathematica GPGPU

- Difficult to use at the moment

```
CUDAFunctionLoad[src, fun, argtypes, blockdim]
    loads CUDAFunction from scr and makes fun available in Mathematica.

CUDAFunctionLoad[{srcfile}, fun, argtypes, blockdim]
    loads CUDAFunction from srcfile and makes fun available in Mathematica.

CUDAFunctionLoad[{libfile}, fun, argtypes, blockdim]
    loads CUDAFunction from libfile and makes fun available in Mathematica.
```

1. Write CUDA code. Pass it as "src".
2. Fill out the function name ("fun"), arguments ("arg") and block dimension ("blockdim").
3. Compile CUDA code. It is now a callable function.

Parallel MATLAB

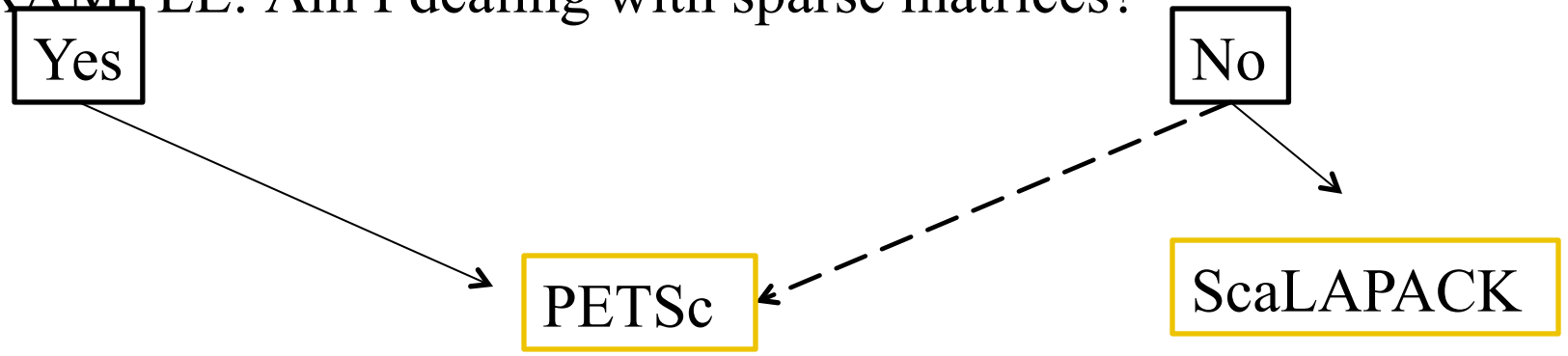
- Built-in parallel algorithms optimized for both distributed memory computing and GPGPU.
- More info here- http://www.mathworks.com/products/parallel-computing/

```
>>
>> % Create arrays that reside on the GPU
>> A = gpuArray(rand(1000, 1000));
>> b = gpuArray(rand(1000, 1));
>>
>> % Use GPU-enabled MATLAB functions
>> x_gpu = A \ b; % "\" is GPU-enabled
>>
>> f_gpu = fft(A);
>>
>> % Bring data back from GPU memory into MATLAB workspace
>> x = gather(x_gpu);
>> f = gather(f_gpu);
>>
```

Choosing your library

- The libraries and software packages presented here are not all there is, do research and ask people solving similar problems what they are using
- There is often an overlap in functionality, so figuring out which is better is a question dependent on the project

EXAMPLE: Am I dealing with sparse matrices?



Remember to do your research

- Choosing which library to use can seem like a long grueling process, but picking the right one for your application can save countless hours of time.
- Read the documentation carefully and run a couple of examples before attempting to do anything with a library you haven't used before.
- Keep someone close at hand who has used the library who can be your "lifeline" in that you can contact them when something isn't working correctly and the documentation isn't helping.