# PETSc Tutorial June 15, 2015

- Introduction of tutors
  - Barry Smith
  - Satish Balay
  - Matt Knepley
  - Jed Brown
  - Dmitry Karpeyev
  - Mark Adams
- Material to be presented
  - DAE/ODE integrators
  - Vectors and matrices
  - Linear preconditioners
  - Nonlinear solvers
  - Understanding performance

# Valgrind

Valgrind is a debugging framework

- **Memcheck**: Check for memory overwrite and illegal use
- **Callgrind**: Generate call graphs
- **Cachegrind**: Monitor cache usage
- **Helgrind**: Check for race conditions
- **Massif**: Monitor memory usage

Memcheck will catch

- Illegal reads and writes to memory
- Uninitialized values
- Illegal frees
- Overlapping copies
- Memory leaks

Let's try a simple experiment

```
# Memcheck is the default tool
valgrind --trace-children=yes --suppressions=bin/simple.supp \
  ./bin/ex5 -use_coords
# Try it for multiple processes
valgrind --trace-children=yes --suppressions=bin/simple.supp \
  $PETSC_DIR/$PETSC_ARCH/bin/mpiexec -n 2 ./bin/ex5 -use_coords
```

## We get an error!

```
==13697== Invalid read of size 8
==13697==    at 0x100005263: MyInitialGuess(AppCtx*, _p_Vec*) (myStuff.c:45)
==13697==    by 0x100004447: main (ex5.c:202)
==13697==  Address 0x103dc6fa0 is 0 bytes after a block of size 48 alloc'd
==13697==    at 0x10001ED75: malloc (vg_replace_malloc.c:236)
==13697==    by 0x1005CABC4: PetscMallocAlign(unsigned long, int, char const*, char const*, char c
==13697==    by 0x1009CC07D: VecGetArray2d(_p_Vec*, int, int, int, int, double***) (rvector.c:1739
==13697==    by 0x10030D980: DMDAVecGetArray(_p_DM*, _p_Vec*, void*) (dagetarray.c:72)
==13697==    by 0x100005102: MyInitialGuess(AppCtx*, _p_Vec*) (myStuff.c:38)
==13697==    by 0x100004447: main (ex5.c:202)
==13697==
==13697== Invalid read of size 8
==13697==    at 0x100005273: MyInitialGuess(AppCtx*, _p_Vec*) (myStuff.c:45)
==13697==    by 0x100004447: main (ex5.c:202)
==13697==  Address 0x18 is not stack'd, malloc'd or (recently) free'd
==13697==
==13698== Use of uninitialised value of size 8
==13698==    at 0x10000529D: MyInitialGuess(AppCtx*, _p_Vec*) (myStuff.c:45)
==13698==    by 0x100004447: main (ex5.c:202)
==13698==
==13698== Invalid read of size 8
==13698==    at 0x10000529D: MyInitialGuess(AppCtx*, _p_Vec*) (myStuff.c:45)
==13698==    by 0x100004447: main (ex5.c:202)
==13698==  Address 0x6f5c300000018 is not stack'd, malloc'd or (recently) free'd
```

```
# Memcheck is the default tool
valgrind --tool=massif --trace-children=yes \
  --massif-out-file=vecfem.massif \
  ./vecfem --sizes=[100,100] -ksp_rtol 1.0e-9
# Turn on stack profiling
valgrind --tool=massif --trace-children=yes \
  --massif-out-file=vecfem.massif \
  ./vecfem --stacks=yes --sizes=[100,100] -ksp_rtol 1.0e-9
# Visualize output
ms_print --threshold=10.0 vecfem.massif
```

# Correctness Debugging

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers

# An optimized build

- `$ intel-dbg/conf/reconfigure-intel-dbg.py PETSC_ARCH=intel-opt --with-debugging=0 && make PETSC_ARCH=intel-opt`
- What does `--with-debugging=1` (default) do?
  - Keeps debugging symbols (of course)
  - Maintains a stack so that errors produce a full stack trace (even SEGV)
  - Does lots of integrity checking of user input
  - Places sentinels around allocated memory to detect memory errors
  - Allocates related memory chunks separately (to help find memory bugs)
  - Keeps track of and reports unused options
  - Keeps track of and reports allocated memory that is not freed `-malloc_dump`

# Interacting with the Debugger

- Launch the debugger
  - `-start_in_debugger [gdb,dbx,noxterm]`
  - `-on_error_attach_debugger [gdb,dbx,noxterm]`
- Attach the debugger only to some parallel processes
  - `-debugger_nodes 0,1`
- Set the display (often necessary on a cluster)
  - `-display khan.mcs.anl.gov:0.0`

# Interacting with the Debugger

```
  $ ./ex6 -start_in_debugger noxterm,lldb
[0]PETSC ERROR: PETSC: Attaching lldb to ./ex6 of pid 7432
Process 7432 stopped
    frame 0: 0x00007fff8d94b48a libsystem_kernel.dylib`__se
libsystem_kernel.dylib`__semwait_signal:
->  0x7fff8d94b48a <+10>: jae    0x7fff8d94b494
    0x7fff8d94b48c <+12>: movq   %rax, %rdi
    0x7fff8d94b48f <+15>: jmp    0x7fff8d946c78
    0x7fff8d94b494 <+20>: retq
(lldb) c
Process 7432 resuming
(lldb)
Process 7432 stopped
    frame 0: 0x0000000102ecbb80 ex6`main(argc=3, args=0x000
   71     ierr = PetscBinaryRead(fd,avec,sz,PETSC_SCALAR);C
-> 72     avec[10000000] = 23;
   73     ierr = VecRestoreArray(vec,&avec);CHKERRQ(ierr);
(lldb)
```

# Time integration in PETSc

- ODE forms supported

$$G(t, x, \dot{x}) = F(t, x)$$
$$J_\alpha = \alpha G_{\dot{x}} + G_x \text{ or}$$
$$M(t)\dot{x} = F(t, x)$$
$$J_\alpha = \alpha M \text{ or}$$
$$\dot{x} = F(t, x)$$

- User provides:
  - `FormRHSFunction(ts,`$t, x, F$`,void *ctx);`
  - `FormIFunction(ts,`$t, x, \dot{x}, G$`,void *ctx);`
  - `FormIJacobian(ts,`$t, x, \dot{x}, \alpha, J, J_p$`,void *ctx);`

# Motivation for IMEX time integration

- Explicit methods are easy and accurate, but must resolve all time scales
  - reactions, acoustics, incompressibility
- Implicit methods are robust
  - mathematically good for stiff systems
  - harder to implement, need efficient solvers
- Implicit-explicit methods are fragile and complicated
  - Severe order reduction
  - Still need implicit solvers, similar complexity to implicit
  -

# Motivation for IMEX time integration

- Explicit methods are easy and accurate, but must resolve all time scales
  - reactions, acoustics, incompressibility
- Implicit methods are robust
  - mathematically good for stiff systems
  - harder to implement, need efficient solvers
- Implicit-explicit methods are fragile and complicated
  - Severe order reduction
  - Still need implicit solvers, similar complexity to implicit
  - Why bother?

# Motivation for IMEX time integration

- Explicit methods are easy and accurate, but must resolve all time scales
  - reactions, acoustics, incompressibility
- Implicit methods are robust
  - mathematically good for stiff systems
  - harder to implement, need efficient solvers
- Implicit-explicit methods are fragile and complicated
  - Severe order reduction
  - Still need implicit solvers, similar complexity to implicit
  - Very expensive non-stiff residual evaluation
  - Non-stiff components are non-smooth.
    - TVD limiters for monotone transport
    - Phase change

# IMEX time integration in PETSc

- Can have *L*-stable DIRK for stiff part *G*, SSP explicit part, etc.
- Orders 2 through 5, embedded error estimates
- Dense output, hot starts for Newton
- More accurate methods if *G* is linear, also Rosenbrock-W
- Can use preconditioner from classical “semi-implicit” methods
- FAS nonlinear solves supported
- Extensible adaptive controllers, can change order within a family
- Easy to register new methods: `TSARKIMEXRegister()`
- Single step interface so user can have own time loop
- Same interface for Extrapolation IMEX

# Some TS methods

**TSSSPRK104** 10-stage, fourth order, low-storage, optimal explicit SSP Runge-Kutta $c_{\text{eff}} = 0.6$ (Ketcheson 2008)

**TSARKIMEX2E** second order, one explicit and two implicit stages, $L$-stable, optimal (Constantinescu)

**TSARKIMEX3** (and 4 and 5), $L$-stable (Kennedy and Carpenter, 2003)

**TSROSWRA3PW** three stage, third order, for index-1 PDAE, $A$-stable, $R(\infty) = 0.73$, second order strongly $A$-stable embedded method (Rang and Angermann, 2005)

**TSROSWRA34PW2** four stage, third order, $L$-stable, for index 1 PDAE, second order strongly $A$-stable embedded method (Rang and Angermann, 2005)

**TSROSWLLSSP3P4S2C** four stage, third order, $L$-stable implicit, SSP explicit, $L$-stable embedded method (Constantinescu)

# Globalizing the lid-driven cavity

- Pseudotransient continuation ($\Psi tc$)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`
- Make the method nonlinearly implicit: `-snes_type ls -snes_monitor`
  - Compare required number of linear iterations
- Try error-based adaptivity: `-ts_type rosw -ts_adapt_dt_min 1e-4`
- Try increasing `-lidvelocity`, `-grashof`, and problem size
- Coffey, Kelley, and Keyes, Pseudotransient continuation and differential algebraic equations, SIAM J. Sci. Comp, 2003.

# Globalizing the lid-driven cavity

- Pseudotransient continuation ($\Psi tc$)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`

```
16x16 grid, lid velocity = 100, prandtl # = 1, grashof # = 100000
0 TS dt 0.03125 time 0
1 TS dt 0.034375 time 0.034375
2 TS dt 0.0398544 time 0.0742294
3 TS dt 0.0446815 time 0.118911
4 TS dt 0.0501182 time 0.169029
...
24 TS dt 3.30306 time 11.2182
25 TS dt 8.24513 time 19.4634
26 TS dt 28.1903 time 47.6537
27 TS dt 371.986 time 419.64
28 TS dt 379837 time 380257
29 TS dt 3.01247e+10 time 3.01251e+10
30 TS dt 6.80049e+14 time 6.80079e+14

CONVERGED_TIME at time 6.80079e+14 after 30 steps
```

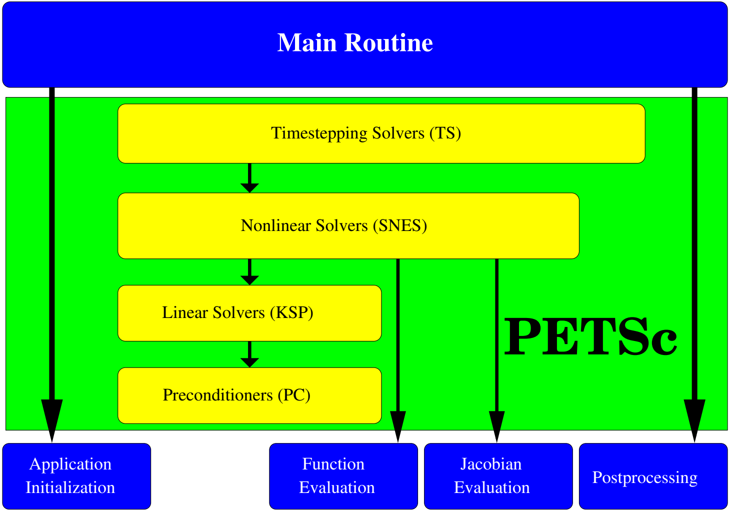
- Make the method nonlinearly implicit: `-snes_type ls -snes_monitor`

# Globalizing the lid-driven cavity

- Pseudotransient continuation ($\Psi tc$)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`
- Make the method nonlinearly implicit: `-snes_type ls -snes_monitor`
  - Compare required number of linear iterations
- Try error-based adaptivity: `-ts_type rosw -ts_adapt_dt_min 1e-4`
- Try increasing `-lidvelocity`, `-grashof`, and problem size
- Coffey, Kelley, and Keyes, Pseudotransient continuation and differential algebraic equations, SIAM J. Sci. Comp, 2003.

# TS Examples

- 1D nonlinear hyperbolic conservation laws
  - `src/ts/examples/tutorials/ex9.c`
  - `./ex9 -da_grid_x 100 -initial 1 -physics shallow -limit minmod -ts_ssp_type rks2 -ts_ssp_nstages 8 -ts_monitor_draw_solution`
- Stiff linear advection-reaction test problem
  - `src/ts/examples/tutorials/ex22.c`
  - `./ex22 -da_grid_x 200 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type ra34pw2 -ts_adapt_monitor`
- 1D Brusselator (reaction-diffusion)
  - `src/ts/examples/tutorials/ex25.c`
  - `./ex25 -da_grid_x 40 -ts_monitor_draw_solution -ts_type rosw -ts_rosw_type 2p -ts_adapt_monitor`




- IGA used to evaluate nonlinear residuals
- PETSc `DA` used to manage parallelism.
- Adaptive time integration using method of lines.
  - Generalized $\alpha$ method from PETSc `TS`.