KGEN:

Fortran Kernel Generator

Youngsung Kim, John Dennis, Raghu R. Kumar, and Amogh Simha

National Center for Atmospheric Research (NCAR)

Contents

KGEN: Key Features

- KGEN extracts a Fortran subprogram as a stand-alone software out of a large software application such as CESM*

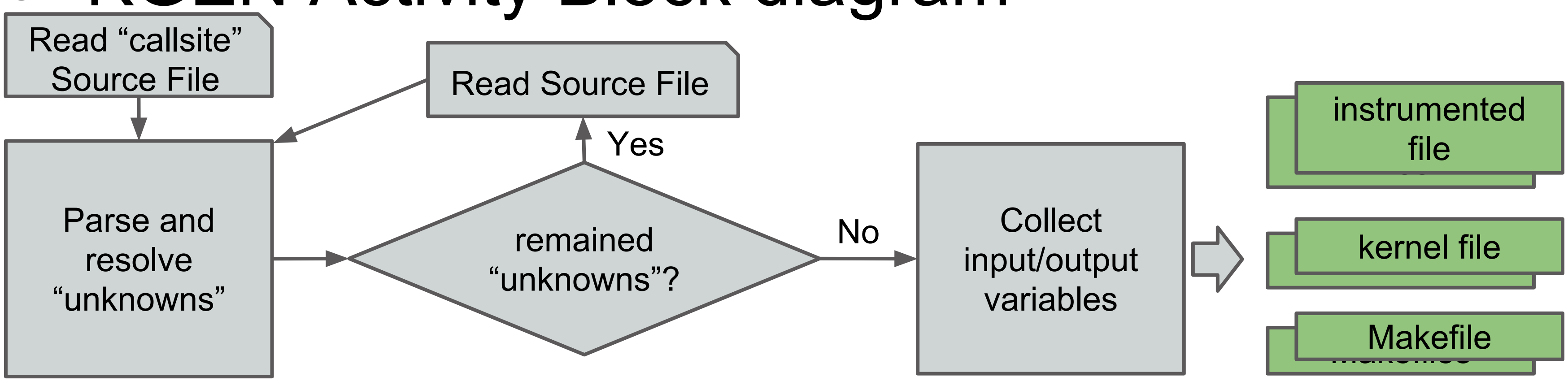
- In addition, it generates instrumented files that save input & output data for the generated kernel

- Correctness check and timing measurement are included in the generated kernel

CESM*: Community Earth System Model

KGEN: Implementation Overview

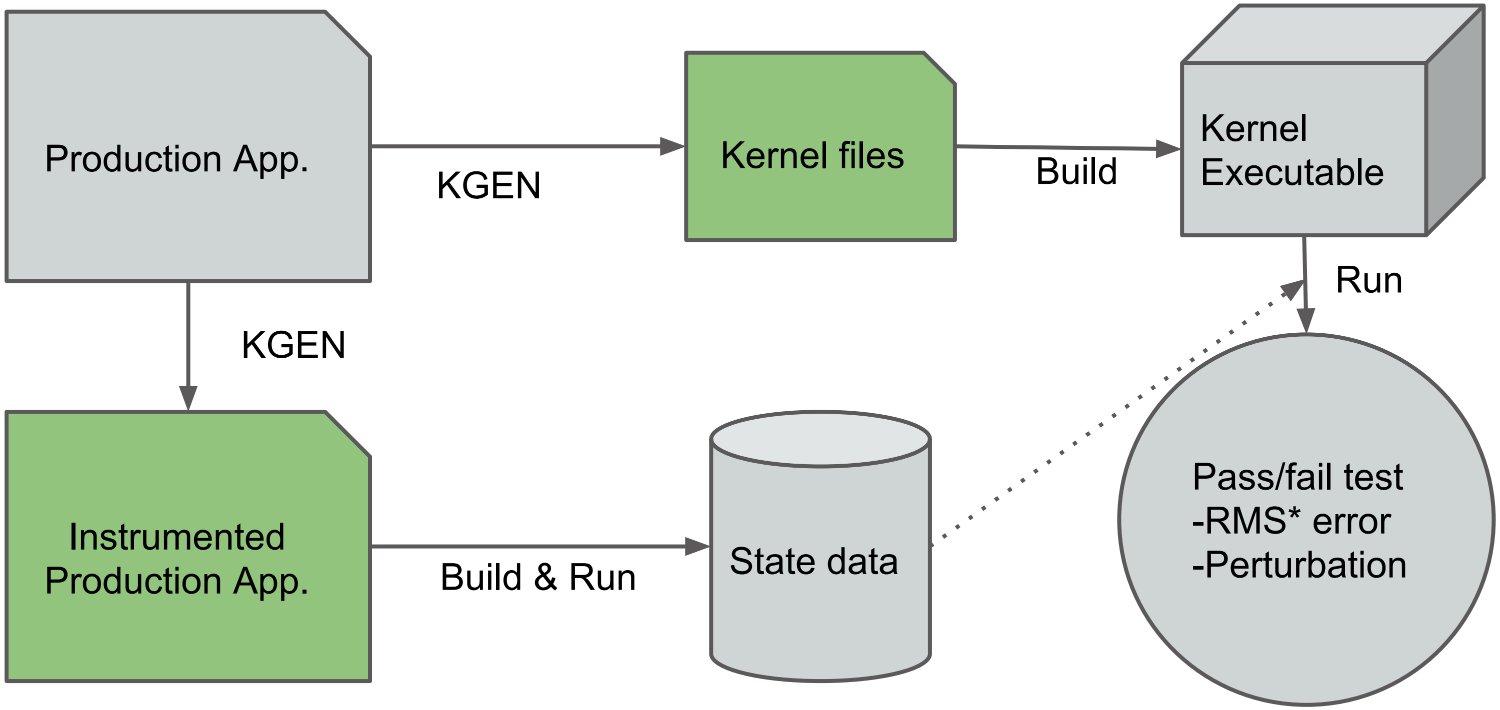
- KGEN is:
  - written in Python 2.6.6 as an extension of F2PY* Fortran parsers. No need for other external modules
  - capable of resolving “unknowns” by searching Abstract Syntax Tree generated by F2PY* parsers
- KGEN Activity Block diagram



F2PY*: https://code.google.com/p/f2py/

KGEN: Workflow

- Two parts: kernel generation and state data generation



RMS*: Root mean square

A kernel generation example

./program.F90

```
PROGRAM demo
   USE update_mod, &
       only : update
   INTEGER t
   DO t=1,10
      CALL update
   END DO
END PROGRAM
```

./update_mod.F90

```
MODULE update_mod
   USE calc_mod, only : calc
   PUBLIC update
CONTAINS
   SUBROUTINE update()
      INTEGER :: i, j
      INTEGER :: output(4,4)
      DO i=1,4
         DO j=1,4

            CALL calc(i, j, output)
         END DO
      END DO
   END SUBROUTINE
END MODULE
```

./calc_mod.F90

```
MODULE calc_mod
   PUBLIC calc
CONTAINS
   SUBROUTINE calc(i, j, output)
      INTEGER, INTENT(IN) :: i, j
      INTEGER, INTENT(OUT), &
       dimension(:,:) :: output
      output(i,j) = i + j
   END SUBROUTINE
END MODULE
```

A kernel generation example - cont.

./program.F90

```
PROGRAM demo
   USE update_mod, &
       only : update
   INTEGER t
   DO t=1,10
      CALL update
   END DO
END PROGRAM
```

./update_mod.F90

```
MODULE update_mod
   USE calc_mod, only : calc
   PUBLIC update
CONTAINS
   SUBROUTINE update()
      INTEGER :: i, j
      INTEGER :: output(4,4)
      DO i=1,4
         DO j=1,4
            !$kgen callsite calc
            CALL calc(i, j, output)
         END DO
      END DO
   END SUBROUTINE
END MODULE
```

Call-site

./calc_mod.F90

```
MODULE calc_mod
   PUBLIC calc
CONTAINS
   SUBROUTINE calc(i, j, output)
      INTEGER, INTENT(IN) :: i, j
      INTEGER, INTENT(OUT), &
       dimension(:,:) :: output
      output(i,j) = i + j
   END SUBROUTINE
END MODULE
```

Kernel

A kernel generation example - cont.

./program.F90

```
PROGRAM demo
   USE update_mod, &
       only : update
   INTEGER t
   DO t=1,10
      CALL update
   END DO
END PROGRAM
```

./update_mod.F90

```
MODULE update_mod
   USE calc_mod, only : calc
   PUBLIC update
CONTAINS
   SUBROUTINE update()
      INTEGER :: i, j
      INTEGER :: output(4,4)
      DO i=1,4
         DO j=1,4
            !$kgen callsite calc
            CALL calc(i, j, output)
         END DO
      END DO
   END SUBROUTINE
END MODULE
```

Call-site

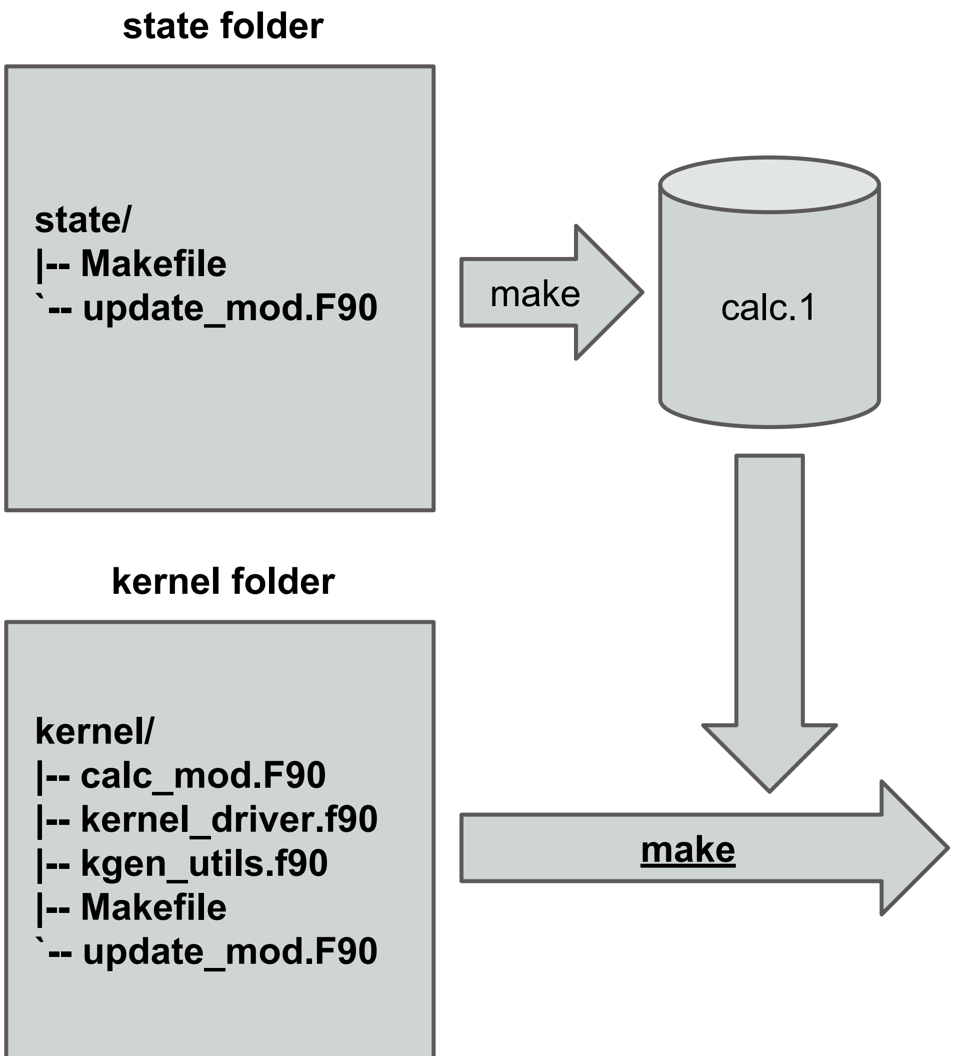
./calc_mod.F90

```
MODULE calc_mod
   PUBLIC calc
CONTAINS
   SUBROUTINE calc(i, j, output)
      INTEGER, INTENT(IN) :: i, j
      INTEGER, INTENT(OUT), &
       dimension(:,:) :: output
      output(i,j) = i + j
   END SUBROUTINE
END MODULE
```

Kernel

```
>> export KGEN=/glade/u/tdd/asap/contrib/kgen/std/src/kgen.py
>> python ${KGEN} ./update_mod.F90
```

A kernel generation example - cont.




```
>> make
ifort -O3 -c -o kgen_utils.o kgen_utils.f90
ifort -O3 -c -o calc_mod.o calc_mod.F90
ifort -O3 -c -o update_mod.o update_mod.F90
ifort -O3 -c -o kernel_driver.o kernel_driver.f90
ifort -O3   -o kernel.exe kernel_driver.o
       update_mod.o kgen_utils.o calc_mod.o

./kernel.exe

** Verification against 'calc.1' **

 calc: Tolerance for normalized RMS:
9.999999824516700E-015
 calc: Number of variables checked:          1
 calc: Number of Identical results:          1
 calc: Number of warnings detected:          0
 calc: Number of fatal errors detected:      0
 calc: verification PASSED

 Elapsed time (sec):   1.0000000E-07
```

KGEN for Large-scale Applications
- Preprocessing

- Pre-processing using “fpp” or “cpp”
  - macros are specified using “-D” or “-i <path>” flag
  - include paths are specified using “-I” or “-i <path>”

>> **python ${KGEN}** -D NC=4,NP=4,PLEV=30 -i include.ini \
${SRC_DIR}/micro_mg_cam.F90

KGEN for Large-scale Applications - Preprocessing

- Pre-processing using “fpp” or “cpp”
  - macros are specified using “-D” or “-i <path>” flag
  - include paths are specified using “-I” or “-i <path>”

Macro definitions

ini file for macros and include paths

>> **python ${KGEN}** -D NC=4,NP=4,PLEV=30 -i include.ini \
${SRC_DIR}/micro_mg_cam.F90

include.ini

```
[include]
/ncar/opt/intel/12.1.0.233/impi/4.0.3.008/intel64/include =

[~/cam5/components/cam/src/physics/cam/micro_mg_cam.F90]
PSUBCOLS = 1
HAVE_F2003_PTR_BND_REMAP = 1
…
[~/cam5/components/cam/src/physics/cam/micro_mg_cam.F90]
...
```

KGEN for Large-scale Applications - Excluding unnecessary searching

- Excluding unnecessary “searching”
  - Intrinsic subroutines are not searched as default
  - User-specified variables or subprograms can be ignored

>> **python ${KGEN}** -e exclude.ini **${SRC_DIR}/micro_mg_cam.F90**

KGEN for Large-scale Applications - Excluding unnecessary searching

- Excluding unnecessary "searching"
  - Intrinsic subroutines are not searched as default
  - User-specified variables or subprograms can be ignored

ini file for exclusion

>> **python ${KGEN}** -e exclude.ini **${SRC_DIR}/micro_mg_cam.F90**

exclude.ini

```
[common]
endrun = comment
t_initf = comment
t_setLogUnit = comment
t_getLogUnit =comment
```

KGEN for Large-scale Applications - Data generation from MPI app.

- State data generation from MPI application
  - User can specify MPI ranks and Nth invocation to save input data to and output data from the kernel

```
>> python ${KGEN} \
    --ordinal-numbers 1,10,20 \
    --mpi ranks=0,100,300 \
   ${SRC_DIR}/micro_mg_cam.F90
```

KGEN for Large-scale Applications - Data generation from MPI app.

- State data generation from MPI application
  - User can specify MPI ranks and Nth invocation to save input data to and output data from the kernel

**>> python ${KGEN} **

Nth invocation to kernel for data generation

--ordinal-numbers 1,10,20 \

--mpi ranks=0,100,300 \

MPI ranks for data generation

${SRC_DIR}/micro_mg_cam.F90

MG2 Kernel(CESM)
Demo.

MG2 (Morrison-Gettleman) Kernel

- MG2 is “a new two-moment stratiform cloud microphysics scheme in a general circulation model.”[1]
- 10% of the cost of CAM[2] on Yellowstone
- Being optimized by multiple experts from various compiler vendors and NCAR using the MG2 kernel generated from KGEN.

1: Hugh Morrison and Andrew Gettelman, 2008: A New Two-Moment Bulk Stratiform Cloud Microphysics Scheme in the Community Atmosphere Model, Version 3 (CAM3). Part I: Description and Numerical Tests. *J. Climate*, **21**, 3642–3659.
doi: http://dx.doi.org/10.1175/2008JCLI2105.1

2: Community Atmospheric Model

An example: Creating a kernel of MG2 microphysics from CESM

SUBROUTINE micro_mg_tend (…..... 114 dummy arguments ……..)

- Around 2300 lines of Fortran code in this subroutine only.
- Other source files provide sub-programs and specifications such as type declarations and parameters required to compile this subroutine
- Need data to drive this kernel and to verify the correctness of its result

END SUBROUTINE

KGEN command line for MG2 kernel

```
#!/bin/bash
# kgen_run.sh

CASE_DIR := ${CAM5_HOME}/cime/scripts/FC5-cam5-mg2-SNB
SOURCE_MODS := ${CASE_DIR}/SourceMods/src.cam
SRC := ${SRC_DIR}/micro_mg_cam.F90

python ${KGEN} \
  -i include.ini \
  --outdir ${OUTPUT_DIR} \
  --ordinal-numbers 10,50,100 \
  --mpi ranks=0:100:300,comm=mpicom,use="spmd_utils:mpicom" \
  --kernel-compile FC=ifort,FC_FLAGS='-xHost -O2' \
  ${SRC}:micro_mg_cam.micro_mg_cam_tend.micro_mg_tend2_0
```

User-provided macros definitions and included paths

```
; include.ini
[include]
/ncar/opt/intel/12.1.0.233/impi/4.0.3.008/intel64/include =

[~/cam5_3_74/components/cam/src/physics/cam/micro_mg_cam.F90]
PSUBCOLS = 1
HAVE_F2003_PTR_BND_REMAP = 1
HAVE_SLASHPROC = 1
HAVE_NANOTIME = 1
…
[~/cam5_3_74/components/cam/src/physics/cam/micro_mg_cam.F90]
...
```

NOTE: KGEN provides a python script that generates include paths and macro definitions from CESM log files.

Screen output

```
>> kgen_run.sh

Pre-processing is done
Reading ~/cam5/components/cam/src/physics/cam/micro_mg_cam.F90
Call-site location is found
Reading ~/cam5/components/cam/src/physics/cam/micro_mg2_0.F90
Reading ~/cam5/components/cam/src/physics/cam/micro_mg_utils.F90
Reading ~/cam5/components/cam/src/physics/cam/wv_sat_methods.F90
Reading ~/cam5/cime/share/csm_share/shr/shr_spfn_mod.F90
Reading ~/cam5/cime/share/csm_share/shr/shr_kind_mod.F90
Kernel information is collected
Instrumented files are generated
Kernel files are generated
Makefiles are generated
Post-processing is done
Completed.
```

Generated files in output folders

```
kernel
|-- kernel_driver.f90
|-- kgen_utils.f90
|-- Makefile
|-- micro_mg2_0.F90
|-- micro_mg_cam.F90
|-- micro_mg_utils.F90
|-- shr_kind_mod.F90
|-- shr_spfn_mod.F90
`-- wv_sat_methods.F90
```

```
state
|-- Makefile
|-- micro_mg2_0.F90
|-- micro_mg_cam.F90
|-- micro_mg_utils.F90
`-- wv_sat_methods.F90
```

Potential use-cases

- Optimization and Porting
  - Original use case
  - Enable fast cycle of workflow
- Debugging
  - Potentially no need for queue
  - Focusing on the target code
  - Quick cycle-time
- Unit-test
  - Verification of new-updates in production code with data generated from the production code
- Benchmark test
  - allows organization to generate their own collection of test cases
- And more
  - New compiler verification
  - Automated optimization
  - Training

Development status and plans

- Development status
  - Funded by Intel Parallel Computing Center focused on Weather and Climate Simulation (IPCC-WACS)
  - Has been applied to mostly CAM of CESM with Intel compiler
  - Available to "early-adopters" TODAY
    - svn co https://proxy.subversion.ucar.edu/pubasap/kgen/trunk
    - Available in "/glade/u/tdd/asap/contrib/kgen/std" on Yellowstone
    - Welcome your opinion and comments on using KGEN.

      If you have one, please send it to "kgen@ucar.edu"
- Plans
  - Improving supports for Fortran specification
  - Expanding tests to components of CESM and other applications
  - Expanding supports for various compilers
  - Supporting for "early-adopters" of KGEN

Thank you!

Funded by Intel Parallel Computing Center focused on Weather and Climate Simulation (IPCC-WACS)

Youngsung Kim, John Dennis, Raghu R. Kumar, and Amogh Simha
(kgen@ucar.edu)