PCAST
Portland Group Compiler Aided Software Testing
PCAST aims to answer two common questions:

Is my application getting the same results:

with the new version of the compile
with different compilation flags
on different computing platforms

as it previously reported?

If my results differ between the GPU and the CPU, where are those differences coming from?
Community history

Jim Rosinski’s Pergrow test:

• Randomly perturb the initial conditions of a simulation with the understanding that the answers produced should still converge to some reasonable result.
• Test would run a 48 hour simulation
• Results would be looked at by a climate scientist to determine if they were reasonable
• Test was able to identify compiler issues but was not usable for a compiler engineer as it required application specific judgement of results and it gave no indication where divergence was occurring. None the less, a good first step.
Name: CESM – Allison Baker

- Principle Component Analysis
  - Looks a key components of climate
  - Matches different outputs statistically to see if results are bounded
  - Tested more than 50 variables under 100’s of conditions

- This approach isn’t specifically targeted for compiler differences, but can be used to show that compiler changes produce statistically different outputs, indicating compiler related issues.

- Again, the testing mechanism gave no real indication to the compiler engineer where an issue was occurring.
Determining GPU/CPU differences in F77

SUBROUTINE TRDSLV0(nz,ims,ime,ips,ipe,A,B,C,Y,X)

  IMPLICIT NONE
  INTEGER, INTENT(IN  ) ::  nz ,ims,ime,ips,ipe
  REAL,   INTENT(IN   ) :: A(nz+1,ims:ime)
  REAL,   INTENT(IN   ) :: B(nz+1,ims:ime)
  REAL,   INTENT(IN   ) :: C(nz+1,ims:ime)
  REAL,   INTENT(IN   ) :: Y(nz+1,ims:ime)
  REAL,   INTENT( OUT) :: X(nz+1,ims:ime)

  • Write a driver which reads in data to initial INTENT(IN) arrays
  • Write out INTENT(OUT) arrays
  • compare outputs between CPU and GPU
  • Technique can isolate issues to a single kernel.  Standard bisection techniques may be able to further narrow.

  • Discussed technique with Oliver Fuhrer at iCAS2k11 – he was essentially doing the same thing
  • Sadly, not all codes are written in Fortran77
Fortran90 interfaces, especially modules, complicate things

Youngsung Kim and John Dennis developed KGEN

KGEN extracts kernels from codes and captures the state of the data at input to the kernel.

Provides a stand alone testing environment for a smaller section of code.

Can be used to test GPU/CPU differences, compiler differences, flag differences, etc.

Delivers everything that we’re after!
Enter PCAST

• The compiler can generate code for the GPU
• The compiler can generate code for the CPU
• The compiler can bundle both GPU and CPU code into the same executable
• The GPU and the CPU have separate memory spaces
• Why not
  • run a kernel on the CPU and keep results in CPU memory
  • run the same kernel on the GPU
  • compare the results between the CPU and the GPU
  • do this all “automatically”
• Great idea – let’s hire an intern to figure out how to do it.
Compiler flags for PCAST

• Compile with –ta=tesla:autocompare
• Avoid initial confusion by turning off fused multiply add on both CPU and GPU
  • -Mnofma (for CPU)
  • -ta=tesla:nofma (for GPU)
  • These flags can always be removed after initial analysis to understand the impact of FMA differences on the two chips
• Cannot use managed memory – CPU and GPU must have separate memory spaces
• CPU cannot change data while GPU is running (no asynchronous compute in code)

• Set environmental variable:
  • export PGICOMPARE=rel=5,verboseautocompare

• Run the code
• Look at output – run.out.sbe08.initial
Use case: RRTMGP

comparing sfc_emis_spec in function rrtmgp_rfmip_lw, /proj/weather/Sites/CSCS/rte-rrtmgp-master/examples/rfmip-clear-sky/rrtmgp_rfmip_lw.F90:266

comparing array in function zero_array_3d, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../rte/mo_util_array.F90:209

comparing tropo in function interpolation, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../rrtmgp/kernels-openacc/mo_gas_optics_kernels.F90:93

PCAST Unsigned int tropo in function interpolation, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../rrtmgp/kernels-openacc/mo_gas_optics_kernels.F90:93

idx: 200 FAIL act: 1 exp: 4294967295
idx: 201 FAIL act: 1 exp: 4294967295
idx: 202 FAIL act: 1 exp: 4294967295
idx: 203 FAIL act: 1 exp: 4294967295
idx: 204 FAIL act: 1 exp: 4294967295
idx: 205 FAIL act: 1 exp: 4294967295
idx: 206 FAIL act: 1 exp: 4294967295
idx: 207 FAIL act: 1 exp: 4294967295
idx: 208 FAIL act: 1 exp: 4294967295
idx: 209 FAIL act: 1 exp: 4294967295
Use case: RRTMGP – where are compares done?

comparing sfc_emis_spec in function rrtmgp_rfmip_lw, /proj/weather/Sites/CSCS/rte-rrtmgp-master/examples/rfmip-clear-sky/rrtmgp_rfmip_lw.F90:266

• Comparisons are done when the host copy of an array is updated.
• Host copies are updated at the exit of a data region
• Host copies can be forced to update with a !$acc update host directive
• Alternatively, !$acc compare(a,b,c(2:N)) can be used to force compares
• Host updates can be inserted before a kernel to check input data as a debugging technique
Use case: RRTMGP

PCAST Unsigned int `tropo` in function interpolation, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../../rrtmgp/kernels-openacc/mo_gas_optics_kernels.F90:93

idx: 200 FAIL act: 1 exp: 4294967295
idx: 201 FAIL act: 1 exp: 4294967295
idx: 202 FAIL act: 1 exp: 4294967295

The 200\textsuperscript{th} element is 1 on the GPU and 4294967295 on the CPU.
tropo is declared as a logical.

\begin{verbatim}
! outputs
    integer, dimension(ncol,nlay), intent(out) :: jtemp, jpress
    logical(wl), dimension(ncol,nlay), intent(out) :: tropo
\end{verbatim}

If values other than “0” or “1” are being used to set the logical, or if the logical is passed as a c_ptr, use the –Munixlogical flag. This extends a logical to “0” and “not 0”. (Best practice is to always use the –Munixlogical flag).

Look at run.out.sbe08.Munixlogical
Use case: RRTMGP – initialization false(?) flags

comparing flux_dn in function apply_bc_0, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../rte/kernels-openacc/mo_rte_solver_kernels.F90:1167

PCAST Double flux_dn in function apply_bc_0, /proj/weather/Sites/CSCS/rte-rrtmgp-master/build/../rte/kernels-openacc/mo_rte_solver_kernels.F90:1167

REL tol: 1.00000000000000004e-10  ABS tol: 0.0000000000000000e+00  ULP tol: 1.00000000000000008e-15

idx: 8 FAIL REL  act: 0.0000000000000000e+00  exp: 1.14288518042947529e+00  dif: 1.0000000000000000e+00

idx: 9 FAIL REL  act: 0.0000000000000000e+00  exp: 1.15681086272589884e+00  dif: 1.0000000000000000e+00

When data is allocated on the GPU, it tends to be initialized to zero, whereas those arrays on the CPU tend to contain the values in memory where it was allocated

Halo type updates tend to cause the most confusion

The easiest way to remove this issue from the comparison is to (temporarily) set arrays to 0 when they are created.
Use case: RRTMGP – initialization false(?) flags

!DWN

\textcolor{red}{\texttt{flux_dn=0.0}}

! Upper boundary condition
if(top_at_1) then
  !$\texttt{acc parallel loop collapse(2)}$
  do igpt = 1, ngpt
    do icol = 1, ncol
      \begin{align*}
      \texttt{flux_dn}(icol, 1, \texttt{igpt}) &= \texttt{inc_flux}(icol, \texttt{igpt}) \\
      \end{align*}
    end do
  end do
else
  !$\texttt{acc parallel loop collapse(2)}$
  do igpt = 1, ngpt
    do icol = 1, ncol
      \begin{align*}
      \texttt{flux_dn}(icol, nlay+1, \texttt{igpt}) &= \texttt{inc_flux}(icol, \texttt{igpt}) \\
      \end{align*}
    end do
  end do
end if

Look at run.out.sbe08.Munixlogical.initialized
Use case: RRTMGP – propagate initialization to kernels

!DWN
sfc_src=0.0
lay_src=0.0
lev_src_inc=0.0
lev_src_dec=0.0
pfrac=0.0
planck_function=0.0

!$acc enter data
    copyin(tlay,lev,tsfc,fmajor,jeta,tropo,jtemp,jpress,gpoint_bands,pfracin,totplnk,gpoint_flavor,one)
    !$acc enter data create(sfc_src,lay_src,lev_src_inc,lev_src_dec)
    !$acc enter data create(pfrac,planck_function)
    !$acc enter data copyin(sfc_src,lay_src,lev_src_inc,lev_src_dec)
    !$acc enter data copyin(pfrac,planck_function)

Look at run.out.sbe08.final
Use case: RRTMGP

- Munixlogical

Initialize arrays

Change data directive to copyin initialized array rather than create

Remove data directive causing partial present error and replaced with declare create clause

PCAST is now happy.

Compare-to-references.py says Intel CPU, Power CPU, Intel+Volta, and Power+Volta all generate the same results

Had there been a real issue with the code, after these steps, it will be easy to spot and work on.
## PGICOMPARE environment variable options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs=n</td>
<td>Compare absolute difference; tolerate differences up to $10^{-(n)}$, only applicable to floating point types. Default value is 0.</td>
</tr>
<tr>
<td>create</td>
<td>Specifies that this is the run that will produce the reference file.</td>
</tr>
<tr>
<td>compare</td>
<td>Specifies that the current run will be compared with a reference file.</td>
</tr>
<tr>
<td>datafile=&quot;name&quot;</td>
<td>Name of the file that data will be saved to, or compared against. If empty will use the default, 'pgi_compare.dat'.</td>
</tr>
<tr>
<td>disable</td>
<td>Calls to pgi_compare, acc_compare, acc_compare_all, and directives (pgi compare, acc compare, and acc compare) all immediately return from the runtime with no effect. Note that this doesn't disable redundant execution; that will require a recompile.</td>
</tr>
<tr>
<td>ieee</td>
<td>Compare IEEE NaN checks (only implemented for floats and doubles).</td>
</tr>
<tr>
<td>outputfile=&quot;name&quot;</td>
<td>Save comparison output to a specific file. Default behavior is to output to stderr.</td>
</tr>
<tr>
<td>patch</td>
<td>Patch errors (outside tolerance) with correct values.</td>
</tr>
<tr>
<td>patchall</td>
<td>Patch all differences (inside and outside tolerance) with correct values.</td>
</tr>
</tbody>
</table>
# PGI_COMPARE environment variable options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel=n</td>
<td>Compare relative difference; tolerated differences up to $10^{(-n)}$, only applicable to floating point types. Default value is 0.</td>
</tr>
<tr>
<td>report=n</td>
<td>Report up to n (default of 50) passes/fails</td>
</tr>
<tr>
<td>reportall</td>
<td>Report all passes and fails (overrides limit set in report=n)</td>
</tr>
<tr>
<td>reportpass</td>
<td>Report passes; respects limit set with report=n</td>
</tr>
<tr>
<td>silent</td>
<td>Suppress output - overrides all other output options, including summary and verbose</td>
</tr>
<tr>
<td>stop</td>
<td>Stop at first differences</td>
</tr>
<tr>
<td>summary</td>
<td>Print summary of comparisons at end of run</td>
</tr>
<tr>
<td>ulp=n</td>
<td>Compare Unit of Least Precision difference (only for floats and doubles)</td>
</tr>
<tr>
<td>verbose</td>
<td>Outputs more details of comparison (including patches)</td>
</tr>
<tr>
<td>verboseautocompare</td>
<td>Outputs verbose reporting of what and where the host is comparing (autocompare only)</td>
</tr>
</tbody>
</table>
Using PCAST on pure CPU code

- PCAST can be used to compare the correctness of:
  - a new version of the compiler to an older version
  - one set of compiler flags to another set (e.g. –O0 to –O2)
  - one CPU architecture to another
  - one revision of code and another
Comparing by means of PCAST functions

```fortran
subroutine pgi_compare(a, datatype, len, varname, filename, funcname, lineno)
    type(*), dimension(..) :: a
    character(*) :: datatype, varname, filename, funcname
    integer(8),value :: len
    integer(4),value :: lineno

The call takes seven arguments, all of which are described below:
• The address of the data to be saved or compared.
• A string containing the data type.
• The number of elements to compare.
• A string treated as the variable name.
• A string treated as the source file name.
• A string treated as the function name.
• An integer treated as a line number.

Usage:
call pgi_compare(a, 'real', n, 'a', 'pgi_compare1.f90', 'program', 9)
```
Too complicated? Use directives instead

The directive syntax is much simpler than the API syntax. Most of what the compare call needs to output data to the user can be gleaned by the compiler at compile-time (The type, variable name, file name, function name, and line number).

```plaintext
#pragma pgi compare(a[0:n])
!$pgi compare(a(1:N))
```

The directive is only enabled when the `-Mcast` flag is set, so the source need not be changed when testing is complete.

The run-time call `pgi_compare` highlights differences between successive program runs. It has two modes of operation, depending on the presence of a data file named `pgi_compare.dat` by default. If the file does not exist, `pgi_compare` assumes this is the first "golden" run. It will create the file and fill it with the computed data at each call to `pgi_compare`. If the file exists, `pgi_compare` assumes it is a test run. It will read the file and compare the computed data with the saved data from the file.

The default behavior is to consider the first 50 differences to be a reportable error, no matter how small.

By default, the `pgi_compare.dat` file is in the same directory as the executable. The behavior of `pgi_compare`, and other comparison parameters, can be changed at runtime with the `PGI_COMPARE` environment variable.
PCAST !$pgi compare directives/pragmas

• Need to be inserted in the code by the user
  • Would it be of value if the compiler inserted ghost directives for all INTENT(OUT) variables in a code as part of the compilation process?
• Behavior is controlled by the –Mpcast flag
• Output can be varied with the PGI_COMPARE environment variable
• These directives are useful for tracking down the location of differences between versions of code, versions of compilers, different compiler options

• OpenACC codes can be augmented with !$acc compare directives
  • !$acc compare can be used on any variable or array in device memory
  • !$acc compare all can be used to compare all variables and arrays in device memory to the same values computed on the CPU
# PGI COMPILERS FOR EVERYONE

The PGI 19.4 Community Edition

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAMMING MODELS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLATFORMS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>X86, OpenPOWER, NVIDIA GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UPDATES</td>
<td>1-2 times a year</td>
<td>6-9 times a year</td>
<td>6-9 times a year</td>
</tr>
<tr>
<td>SUPPORT</td>
<td>User Forums</td>
<td>PGI Support</td>
<td>PGI Premier Services</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Annual</td>
<td>Perpetual</td>
<td>Volume/Site</td>
</tr>
</tbody>
</table>

[pgicompliers.com/community](http://pgicompliers.com/community)