Performance Portable Applications – Is portability even a possibility?

John M. Levesque
CTO Office – HPC,MCS, AI
Performance Evangelist for Coral 2 CoEs
Technical Advisor for Cray Programming Environment
Hewlett Packard Enterprise
Outline of Talk

• Definition of Portability

• Evidence of the lack of Portability today

• What about Performance Portability

• Advise for Compiler Vendors

• A solution?
Definition of Portability

• The same application, regardless of the language, should be able to compile, load, execute and give close to the same answers on Intel, ARM, AMD, NVIDIA or any new systems.

• Clang and LLVM have aided the portability of C++ applications
  • if and only if the compiler adheres to the latest versions of C++ and Clang
  • It is still possible to defeat portability by adding extensions to the language

• We are not even close to portability in Fortran
  • Some Fortran compilers support Fortran 2003, some Fortran 2008, fewer support 2018
  • Many Fortran compilers have non-standard extensions to Fortran
  • Very few Fortran compilers enforce strict compliance — e.g. mismatched arguments, undefined variables
    • Few programmers use interface blocks
  • When Flang/LLVM gets to Fortran 2018 will this fix the portability problem?
Evidence of the lack of Portability

The following Fortran looks dangerous:

\[ a = \text{merge}(t, v_{c}\%v_{c}(l), \text{present}(t)) \]

If present(t) is false this code is non-conforming

If t is not present, the merge() may attempt to load t from address 0. It would be safer to code this thus:

\[
\text{if (present(t)) then} \\
\quad a = t \\
\text{else} \\
\quad a = v_{c}\%v_{c}(l) \\
\text{endif}
\]

The Intel Compiler ran this without warning
Evidence of the lack of Portability

Brian H. Johnson added a comment - 09/Oct/17 1:35 AM

After I finally figure out how to add command line options to the build, I built with -e0 added to the -O2. The test passes with that option. -e0 initializes stack variables to 0. I believe this is a user code problem where it assumes a zero value. Perhaps intel has some similar option, or perhaps their stack use and layout is different enough to hide the problem. In any event, I don't see a compiler problem. I would not recommend using the -e0 option as a solution because of performance issues. Setting some default initializations of the struct components might be the way to solve it. I would guess that the component we were looking at (cell_token) is the one that needs to be initialized.

John Levesque added a comment - 09/Oct/17 7:51 AM

Thanks I will pass it on to the Xrage people.

Brian H. Johnson added a comment - 09/Oct/17 9:07 AM

It appears I was right about the fix. I removed -e0, and changed Types/mesh_types.f90, line 164 to add a default initialization of 0 and the test passed.

```fortran
integer(INT64) :: cell_token(2,3) = 0
```
Evidence of the lack of Portability

- ☑️ Brian H. Johnson added a comment - 10/Jun/20 1:11 PM

   My run is successful. I have the changed source file here

   /cray/css/sumusa/users/bhj/eap_debug/eap/Cassio/eap.xrage/source.xrage/State/Ins

   It does not have any optimize directives. It has the suggested fix for the two illegal optional arg references. I don’t believe there are any more compiler issues with this xrange version.

- ☑️ John Levesque added a comment - 10/Jun/20 3:19 PM

   So this is a coding error on their part. Where there any actual compiler errors?

- ☑️ Brian H. Johnson added a comment - 10/Jun/20 3:26 PM

   No recent fixes. You do need my compiler until a 10.0.2 package is built. The last two runtime issues were

   1. stacksize limit
   2. illegal reference of non-present optional arg.

   Also, my source has the original change for the first error messages you got (the target attribute problem). You had said the developers admitted to that problem.
Some Compilers let obvious errors past

• Reference to an option argument that wasn’t present – A good compiler throws out an error
• Mismatch of arguments – passing an integer into a routine and treating it as a real

The Intel Compiler accepts these without
The Intel Compiler didn’t point these out

```plaintext
ftn-7212 crayftn: WARNING XXXXXXXXXXXX, File = XXXXXXXXXXXX.f90, Line = 2745
Variable "x_con" is used before it is defined.
ftn-7212 crayftn: WARNING XXXXXXXXXXXX, File = XXXXXXXXXXXX.f90, Line = 2745
Variable "y_con" is used before it is defined.
ftn-7212 crayftn: WARNING XXXXXXXXXXXX, File = XXXXXXXXXXXX.f90, Line = 2745
Variable "d_con" is used before it is defined.
```

LANL has used the Cray Compiler to identify all of these undefined variables and in the process found potential errors.
What about Performance Portability?

• You will always have some compilers that are better at generating efficient object code than other compilers; however, these differences will be minor (<20%)

• Where you really see differences in performance portability is when one uses the C++ abstraction layers to run on non-GPU systems using shared memory OpenMP. Typically, users can do a much better job at implementing OpenMP directly.
  • Additionally, never use the abstraction layer to generate OpenACC and/or OpenMP Offload – it is best to go directly to Cuda/HIP which is where the performance comes from.
Performance Portability between GPUs and CPUs

• Since most applications are memory bandwidth limited, applications developers need to efficiently utilize close registers/cache
  • CPUs have very large caches and like large meaty loops that get a lot of cache reuse
  • GPUs have very small register sets/caches and prefer to have smaller computational loops to assure that they do not encounter register spills
• A solution for this discrepancy in close memory is to strip mine loops with large iteration counts, so that, the inner loops avoid register spills on GPUs and effectively utilize cache on CPUs
NCHUNK IS AN INPUT

#ifdef OPENMP_OFFLOAD
!$omp target teams distribute parallel do collapse(2)  
  0 0 0 0 0 0
#else
!$omp parallel do collapse(2)  
  0 0 0 0 0 0
#endif

DO k=1,az
  DO j=1,ay
    DO ii=1,ax,nchunk
      i1 = ii
      i2 = min(ax, i1+nchunk-1)
      ic = i2-i1+1
      0 0 0 0
      ic = 0
      do i = i1, i2
        ic = ic + 1
        A1(ic) = zero
        A2(ic) = zero
        A3(ic) = zero
        A4(ic) = zero
        B1(ic) = one/r(i,j,k)
        B2(ic) = E(i,j,k)
    end do

Tons of these inner loops which are spread across the wavefronts on GPU or vectorized on the CPU.
A Strip-Mined Example

The idea is to control which system you are targeting by varying the variable NCHUNK – this could be an input variable. In testing the GPU I found that 32 was the best for NCHUNK – size of the warp. The host might run better if NCHUNK were reduced to 16. The reason for the performance increase one the GPU is the reuse of cache from loop to I loop. In the original The Table Lookups wipe out the cache. In the restructured the Table lookups utilize the cache very effectively.
Advise for Compiler Vendors

• Stick to the standard
  • At least have a switch that forces strict compliance to standards

• If you are going to support a language, keep up with the standard

• Check for potential errors
  • Uninitialized variables
  • Unmatched arguments (LLVM does not like mis-matched arguments)
What About Extensions for GPUs

• Fortran – WHAT A MESS Why do we have two solutions? It’s like the Democrats and the Republicans not agreeing on a common solution
  • Currently there is only ONE vendor who supports the latest OpenACC standard. One really cannot say OpenACC is portable
  • While NVIDIA (4.5), IBM(4.5) and HPE(5.0) have OpenMP Offload they each support different levels of the standard – One really cannot say that OpenMP Offload is portable

• C++ is in a much better situation with portability abstraction layers. Additionally, many of the abstractions are being proposed to be part of future standards
  • RAJA
  • KOKKOS

NVIDIA and Intel seem to be leaning to Parallel C++
A solution

- Build a compilation system that generates the best optimized code for Fortran, C and C++
  - Accepts all extensions introduced by Intel, gnu, and PGI
  - Accepts all compiler command options users by Intel, gnu, and PGI
  - Generates executables for ARM (NVIDIA), Intel, and AMD CPUs
  - Accepts latest standards for Fortran, C, and C++ (Clang)
  - Accepts latest standards for OpenACC and OpenMP
  - Generates executables for NVIDIA, AMD, and Intel GPUs
A Pipe Dream?

• Maybe; however, HPE is a company who is a system vendor, not a chip vendor, so we are the only compiler vendor that has a direct incentive to provide compilers/models/tools that work across all platforms

• Additionally, it would be nice to work with
  • SPACK
  • Containers
  • Web Servers

• And have state-of-the-art
  • Profiling tools
  • Debugging tools
  • Parallelization tools
  • Visualization tools