Using Compilers

Never forget

- **Always use the `ftn`, `cc`, and `CC` wrappers**
  - The wrappers uses your module environment to get all libraries and include directories for you. You don't have to know their real location.
- **You select the your environment by selecting one of the programming environments `PrgEnv-X` with `X=[cray|pgi|gnu]`**
  "`module swap PrgEnv-cray PrgEnv-gnu`"
- **Use `aprun` to start your application on the compute nodes**

Compiler man pages

- **The cc(1), CC(1), and ftn(1) man pages contain information about the compiler driver commands**
- **Cray compiler: man craycc(1), crayCC(1), and crayftn(1)**
- **GNU compiler: gcc(1), g++(1), and gfortran(1)**
- **PGI compiler: `pgf90(1),pgcc(1)`**

- **To verify that you are using the correct version of a compiler, use:**
  - -V option on a cc, CC, or ftn command with CCE
  - --version option on a cc, CC, or ftn command with GNU

Loopmark: Compiler feedback

- Compilers can generate annotated listing of your source code indicating important optimizations
- CCE
  - `ftn -rm`
  - `cc -hlist=a`
- PGI
  - `ftn/cc -Mlist`
  - See `pgcc manpage for more details`
- GNU
  - `-ftree-vectorizer-verbose=9`

Loopmark: Compiler feedback

- For example, with the Cray compiler

```
%%%    L o o p m a r k   L e g e n d    %%%
Primary Loop Type      Modifiers
------- ---- ----      ---------
                       a - vector atomic memory operation
A  - Pattern matched   b - blocked
C  - Collapsed         f - fused
D  - Deleted           i - interchanged
E  - Cloned            m - streamed but not partitioned
I  - Inlined           p - conditional, partial and/or computed
M  - Multithreaded     r - unrolled
P  - Parallel/Tasked   s - shortloop
V  - Vectorized        t - array syntax temp used
W  - Unwound           w - unwound
```

Loopmark: Compiler feedback

```
29.  b-------<  do i3=2,n3-1
30.  b b-----<     do i2=2,n2-1
31.  b b Vr--<        do i1=1,n1
32.  b b Vr            u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
33.  b b Vr    *         + u(i1,i2,i3-1) + u(i1,i2,i3+1)
34.  b b Vr            u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
35.  b b Vr    *         + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
36.  b b Vr-->        enddo
37.  b b Vr--<        do i1=2,n1-1
38.  b b Vr            r(i1,i2,i3) = v(i1,i2,i3)
39.  b b Vr    *         - a(0) * u(i1,i2,i3)
40.  b b Vr    *         - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
41.  b b Vr    *         - a(3) * ( u2(i1-1) + u2(i1+1) )
42.  b b Vr-->        enddo
43.  b b----->     enddo
44.  b------->  enddo
```

Loopmark: Compiler Feedback

```
 ftn-6289 ftn: VECTOR File = resid.f, Line = 29
  A loop starting at line 29 was not vectorized because a
recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
  A loop starting at line 29 was blocked with block size 4.
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
  A loop starting at line 30 was not vectorized because a
recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
  A loop starting at line 30 was blocked with block size 4.
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
  A loop starting at line 31 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
  A loop starting at line 31 was vectorized.
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
  A loop starting at line 37 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
  A loop starting at line 37 was vectorized.
```

Recommended compiler optimization levels

- **Cray compiler**
  - The default optimization level (i.e. no flags) is equivalent of **`-O3`** of most other compilers
    - CCE optimizes rather aggressively by default, but this is also most thoroughly tested configuration
  - Try with **`-O3 -hfp3`**
    - We also test this thoroughly
    - -hfp3 gives you a lot more floating point optimization, esp. 32-bit
    - In case of precision errors, try a lower –hfp number (–hfp1 first, only –hfp0 if absolutely necessary)
- **PGI compiler**
  - The default optimization level (equal to -O2) is safe and gives usually good performance
  - Try with **–fast**
  - If that works still, you may try with **-fast, -Mipa=fast,inline**
- **GNU compiler**
  - Almost all HPC applications compile correctly with using **`-O3`**, so do that instead of the cautious default
  - **`-ffast-math`** may give some extra performance

Loop transformations

- **Cray compiler**
  - Most useful techniques in their aggressive state already by default
  - One may try to improve loop restructuration for better vectorization with **`-h vector3`**
- **PGI compiler**
  - Loop unrolling **`-Munroll`**
- **GNU compiler**
  - Loop blocking (aka tiling) **`-floop-block`**
  - Loop unrolling **`-funroll-loops`** or **`-funroll-all-loops`**

Inlining & inter-procedural optimization

- **Cray compiler**
  - Inlining within a file is enabled by default
  - Command line options **`-OipaN`** (ftn) and **`-hipaN`** (cc/CC) where N=0..4, provides a set of choices for inlining behavior
    - 0 disables inlining, 3 is the default, 4 is even more elaborate
  - The **`-Oipafrom=`** (ftn) or **`-hipafrom=`** (cc/CC) option instructs the compiler to look for inlining candidates from other source files, or a directory of source files
- **Intel compiler**
  - Inlining within a file is enabled by default
  - Multi-file inlining –Mipa=inline
- **GNU compiler**
  - Quite elaborate inlining enabled by **`-O3`**

CCE Directives

- **Cray compiler supports a full and growing set of directives and pragmas, e.g.**
  - !dir$ concurrent
  - !dir$ ivdep
  - !dir$ interchange
  - !dir$ unroll
  - !dir$ loop_info [max_trips] [cache_na]
  - !dir$ blockable
  - man directives
  - man loop_info

```
!dir$ blockable(j,k)
!dir$ blockingsize(16)
    do k = 6, nz-5
       do j = 6, ny-5
          do i = 6, nx-5
          ! stencil
          end do
       end do
    end do
```

Summary

- **Three compiler environments available on HECToR: Cray, PGI, GNU**
  - All of them accessed through the wrappers ftn, cc and CC – just do module swap to change a compiler!
- **There is no universally fastest compiler – but performance depends on the application, even input**
  - We try however to excel with the Cray Compiler Environment
  - If you see a case where some other compiler yields better performance, let us know!
- **Compiler flags do matter – be ready to spend some effort for finding the best ones for your application**