PGI Fortran, C & C++ Compilers
Optimizing, SIMD Vectorizing, OpenMP

Accelerated Computing Features
OpenACC Directives
CUDA Fortran

Multi-Platform Solution
x86-64 and OpenPOWER CPUs, Tesla and Radeon GPUs
Supported on Linux, macOS, Windows

MPI/OpenMP/OpenACC Tools
PGDBG® debugger
PGPROF® profiler
Interoperable with DDT, Totalview

www.pgroup.com
Riding Waves of Disruption

PGI Founded
Jan 3, 1989

1989

TFLOPs / Beowulf

1997

x86-64 vs IA64

2003

PGI + LLVM

2009

Accelerated Computing

2014
Our Science requires that we continue to advance OLCF’s computational capability over the next decade on the roadmap to Exascale.

Since clock-rate scaling ended in 2003, HPC performance has been achieved through increased parallelism. Jaguar scaled to 300,000 cores.

Titan and beyond deliver hierarchical parallelism with very powerful nodes. MPI plus thread level parallelism through OpenACC or OpenMP plus vectors.
Porting and optimizing production HPC applications from one platform to another can be one of the most significant costs in the adoption of breakthrough hardware technologies. The PGI compiler has been our primary compiler on Jaguar and Titan since 2005. Having the PGI compiler suite available in the POWER environment will provide continuity and facilitate code portability of existing CPU-only and GPU-enabled Titan applications to our next major system.

— Buddy Bland, Titan Project Director, Oak Ridge National Lab
PGI Compilers 2014 ...

- **C++11**
  - OpenMP, OpenACC

- **Fortran 2003**
  - OpenMP, OpenACC, CUDA Fortran

- **x64** CPUs
  - x64
  - CUDA C

- **x64 + Tesla**
  - CUDA C
  - PGI IR +

- **x64 + Radeon**
  - PGI IR +
  - OpenCL

- **PGI IR**
  - Global Optimization
  - Dependence Analysis
  - Function Inlining
  - Loop Exchange/Loop Fusion

- **PGI IR + CUDA C**
  - Loop Peeling/Altcode
  - IPA/IPO
  - Loop Tiling
  - SIMD Vectorization
  - Vector Datatypes
  - Alignment Optimization

- **PGI IR + OpenCL**
  - SW Prefetching
  - OpenMP
  - CCFF
  - OpenACC
  - Profile Feedback
  - Memory Hierarchy Optimization

- **PGI IR + OpenCL**
  - Heterogeneous Targets
  - Auto-parallelization
  - PGI Unified Binary
  - Loop Unrolling
  - SIMD Intrinsics

- **Heterogeneous Targets**
  - SIMD Intrinsics
  - Loop Unrolling

- **Global Optimization**
  - Dependence Analysis
  - Function Inlining
  - Loop Exchange/Loop Fusion

- **Dependence Analysis**
  - Function Inlining
  - SIMD Vectorization
  - Vector Datatypes
  - Alignment Optimization

- **Function Inlining**
  - SIMD Vectorization
  - Vector Datatypes
  - Alignment Optimization

- **Loop Exchange/Loop Fusion**
  - SIMD Vectorization
  - Vector Datatypes
  - Alignment Optimization

- **Loop Peeling/Altcode**
  - IPA/IPO
  - Loop Tiling
  - SIMD Vectorization
  - Vector Datatypes
  - Alignment Optimization

- **SW Prefetching**
  - OpenMP
  - CCFF
  - OpenACC
  - Profile Feedback
  - Memory Hierarchy Optimization

- **OpenMP**
  - CCFF
  - OpenACC
  - Profile Feedback
  - Memory Hierarchy Optimization

- **Auto-parallelization**
  - PGI Unified Binary
  - Loop Unrolling
  - SIMD Intrinsics

- **Heterogeneous Targets**
  - Auto-parallelization
  - PGI Unified Binary
  - Loop Unrolling
  - SIMD Intrinsics

- **PGI Unified Binary**
  - Loop Unrolling
  - SIMD Intrinsics

- **Loop Unrolling**
  - SIMD Intrinsics

- **SIMD Intrinsics**
  - Loop Unrolling

- **PGI Unified Binary**
  - Loop Unrolling
  - SIMD Intrinsics

- **Auto-parallelization**
  - PGI Unified Binary
  - Loop Unrolling
  - SIMD Intrinsics
PGI Compilers 2016 ...

- C11
  - OpenMP, OpenACC
- Fortran 2003
  - OpenMP, OpenACC, CUDA Fortran
- C++14
  - OpenMP, OpenACC, NVCC host Compiler

Global Optimization
- Dependence Analysis
- IPA/IPO
- Function Inlining
- Loop Exchange/Loop Fusion

Loop Peeling/Altcode
- Loop Tiling
- SIMD Vectorization
- Vector Datatypes
- Alignment Optimization

SW Prefetching
- OpenMP
- CCFF

Heterogeneous Targets
- Auto-parallelization
- PGI Unified Binary

SIMD Intrinsic
- Loop Unrolling

Profile Feedback
- Memory Hierarchy Optimization

PGI IR
- x64 CPUs
- x64 + Tesla
- x64 + Radeon

LLVM IR
- OpenPOWER + Tesla
PGI for OpenPOWER+Tesla

Fortran 2003, C11, C++14 compilers, PGPROF profiler

CUDA Fortran, OpenACC, OpenMP, NVCC host compiler

Integrated with LLVM for OpenPOWER code generation

First production release now available
PGI for OpenPOWER+Tesla

Fortran 2003, C11, C++14 compilers, PGPROF profiler

CUDA Fortran, OpenACC, OpenMP, NVCC host compiler

Integrated with LLVM for OpenPOWER code generation

First production release now available
Porting an 800K line HPC Application from x86 to OpenPOWER

Recompile ...

- Makefile
- Source Code
- x86 wrf.exe
- OpenPOWER wrf.exe

Run ...

WRF 3.8.1 OpenMP Performance
PGI 16.10 vs GNU 6.1

<table>
<thead>
<tr>
<th>CPU</th>
<th>PGI 1.2X Faster</th>
<th>PGI 1.8X Faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswell 32 Cores</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POWER8 20 Cores</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x86 CPU: Intel Xeon E5-2698 v3, 2 sockets, 32 cores
OpenPOWER CPU: IBM 8247-42L POWER8E, 4 sockets, 20 cores
PGI options: -fast -Mstack_arrays -mp
GNU options: -O3 -funroll-loops -fpeel-loops -fopenmp
OpenPOWER+Tesla HPC Node

POWER8 CPU

Tesla P100

Shared Cache

NVLink

High Capacity Memory

High Bandwidth Memory
Porting the Gyrokinetic Toroidal Code (GTC) from Xeon+Tesla to OpenPOWER+Tesla using OpenACC

Multiple MPI Ranks + OpenACC

- Haswell 4xK80: 4.7x speed-up
- Haswell 2xP100: 8.8x speed-up
- POWER8 4xP100: 14x speed-up

X86 CPU: Intel Xeon E5-2698 v3,
POWER CPU: IBM POWER8NVL
OpenACC Directives

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, Manycore
... 

#pragma acc data copy(b[0:n][0:m]) \ 
  create(a[0:n][0:m])
{
  for (iter = 1; iter <= p; ++iter){
    #pragma acc kernels
    {
      for (i = 1; i < n-1; ++i){
        for (j = 1; j < m-1; ++j){
          a[i][j]=w0*b[i][j]+ 
          w1*(b[i-1][j]+b[i+1][j]+ 
          b[i][j-1]+b[i][j+1])+ 
          w2*(b[i-1][j-1]+b[i-1][j+1]+ 
          b[i+1][j-1]+b[i+1][j+1]);
        }
      }
      for( i = 1; i < n-1; ++i )
        for( j = 1; j < m-1; ++j )
          b[i][j] = a[i][j];
    }
  }
}
OpenACC for Multicore CPUs & GPUs

% pgfortran a.f90 -ta=multicore -c -Minfo
  sub:
  10, Loop is parallelizable
  Generating Multicore code
  10, !$acc loop gang
  11, Loop is parallelizable

%!acc kernels loop
  do j = 1, m
  do i = 1, n
    a(j,i) = b(j,i)*alpha + c(i,j)*beta
  enddo
  enddo

% pgfortran a.f90 -ta=tesla -c -Minfo
  sub:
  10, Loop is parallelizable
  11, Loop is parallelizable
  Accelerator kernel generated
  Generating Tesla code
  10, !$acc loop gang, vector(4)
  11, !$acc loop gang, vector(32)
PGI OpenACC - SPEC ACCEL 1.0 Benchmarks

Geometric mean across all 15 benchmarks

Performance measured November, 2016 and are considered estimates per SPEC run and reporting rules. SPEC® and SPEC ACCEL® are registered trademarks of the Standard Performance Evaluation Corporation (www.spec.org).
attributes(global) subroutine mm_kernel
  ( A, B, C, N, M, L )
real :: A(N,M), B(M,L), C(N,L), Cij
integer, value :: N, M, L
integer :: i, j, kb, k, tx, ty
real, shared :: Asub(16,16), Bsub(16,16)
tx = threadIdx%x
ty = threadIdx%y
i = blockIdx%x * 16 + tx
j = blockIdx%y * 16 + ty
Cij = 0.0
do kb = 1, M, 16
  Asub(tx,ty) = A(i,kb+tx-1)
  Bsub(tx,ty) = B(kb+ty-1,j)
call syncthreads()
do k = 1,16
  Cij = Cij + Asub(tx,k) * Bsub(k,ty)
enddo
call syncthreads()
enddo
C(i,j) = Cij
end subroutine mmul_kernel

real, device, allocatable, dimension(:,:) ::
  Adev,Bdev,Cdev
allocate (Adev(N,M), Bdev(M,L), Cdev(N,L))
Adev = A(1:N,1:M)
Bdev = B(1:M,1:L)
call mm_kernel <<<dim3(N/16,M/16),dim3(16,16)>>>
  ( Adev, Bdev, Cdev, N, M, L )
C(1:N,1:L) = Cdev
deallocate ( Adev, Bdev, Cdev )

CPU Code

Tesla Code

CUDA Fortran for Tesla GPUs
module madd_device_module
use cudafor
contains

subroutine madd_dev(a,b,c,sum,n1,n2)
real,dimension(:,:), device :: a,b,c
integer :: n1,n2
real :: sum

a(i,j) = b(i,j) + c(i,j)
sum = sum + a(i,j)
enddo
end subroutine
end module
PGI + LLVM Integration
PGI Compilers 2016 ...

- C11
  - OpenMP, OpenACC
- Fortran 2003
  - OpenMP, OpenACC, CUDA Fortran
- C++14
  - OpenMP, OpenACC, NVCC host Compiler

Global Optimization
- Dependence Analysis
- IPA/IPO
- Loop Peeling/Altcode
- Loop Tiling
- Function Inlining
- SIMD Vectorization
- Vector Datatypes
- Alignment Optimization
- Loop Exchange/Loop Fusion

SW Prefetching
- OpenMP
- CCFF

Heterogeneous Targets
- Auto-parallelization
- PGI Unified Binary
- Memory Hierarchy Optimization

SIMD Intrinsics
- Loop Unrolling

PGI IR
- x64 CPUs
- x64 + Tesla
- x64 + Radeon
- OpenPOWER + Tesla

LLVM IR
LLVM is not a code generator...

it is “... a collection of modular and reusable compiler and toolchain technologies.”
Integrating LLVM into the PGI Compilers

- PGI ILI -> LLVM IR bridge, CPU-side and GPU-side
- C/C++/Fortran language support, scalar code generation
- Target independent vectorizer
- OpenMP re-implementation, SMP auto-parallelization
- Enabling OpenACC and CUDA Fortran
- Integration, Testing, Documentation
- Dovetailing PGI optimizer and LLVM opt
subroutine add_exp(n,a,b,c)
  integer n
  real*8, dimension(n) :: a,b,c
  integer i
  do i = 1,n
    a(i) = b(i) + exp(c(i))
  enddo
end subroutine

Leveraging LLVM Vector Data Types
**Target-independent vectorizer**

```
subroutine add_exp(n,a,b,c)
  integer n
  real*8, dimension(n) :: a,b,c
  integer i
  do i = 1,n
    a(i) = b(i) + exp(c(i))
  enddo
end subroutine
```

```
leal  -3(%rbx), %r12d

.LBB0_3:
  vmovupd (%r14,%rbp), %ymm0
  callq  __gvd_exp4
  vaddpd (%r15,%rbp), %ymm0, %ymm0
  vmovupd %ymm0, (%r13,%rbp)
  addq  $32, %r12d
  addl  $-4, %r12d
  testl %r12d, %r12d
  jg  .LBB0_3

.LBB0_4:
  lxvd2x 0, 30, 24
  stxvd2x 0, 1, 23
  ori 2, 2, 0
  lxvd2x 0, 1, 23
  ld 4, 40(1)
  ld 3, 32(1)
  xxswapd 34, 0
  b1 __gvd_exp2
  nop
  lxvd2x 0, 29, 24
  addi 22, 22, -2
  cmpwi 22, 0
  xxswapd 0, 0
  xvadddp 0, 34, 0
  xxswapd 0, 0
  stxvd2x 0, 28, 24
  addi 24, 24, 16
  bgt 0, .LBB0_4
```

**x86-64 AVX-256**

**OpenPOWER VSX**
Outlining parallel regions

```assembly
# Execute SIMD vector loop
# in parallel
.
.call _mp_pexit
```

### subroutine add_exp(n,a,b,c)
```
integer n
real*8, dimension(n) :: a,b,c
integer i
!
$omp parallel do
do i = 1,n
  a(i) = b(i) + exp(c(i))
.enddo

end subroutine
```
PGI + LLVM to do list

- Fortran DWARF generation
- OpenPOWER performance analysis
- OpenMP performance tuning, OpenMP 4.5
- PGI vectorizer performance tuning
- Dovetailing PGI optimizer and LLVM opt
- POWER9 128-bit IEEE floating-point support
Flang
An open source Fortran front-end for LLVM
a.k.a. the Flang project

- Multi-year project: NNSA Labs, NVIDIA/PGI
- Based on the existing front-end from PGI's widely-used Fortran compiler
- Re-engineering for integration with LLVM
- Develop CLANG-quality Fortran msg facility
Many Stakeholders, Many Goals

LANL  
New developer productive in source base in 4 – 8 weeks

Sandia  
Single-thread/SIMD and OpenMP 3.1 performance

LLNL  
OpenMP 4.x features, GPU and OpenPOWER support

NVIDIA  
Accelerate Fortran features support, PGI interoperability

Everyone  
Adoption by both the HPC and LLVM communities

ANL, IBM, ARM Ltd, ORNL, Codethink, …
Creating the initial Flang source base

- Front-end 85%
- Runtime Libraries 15%

pgfortran: 25,311
Flang: 1,174

95% fewer #ifdefs

pgfortran: 2,311
Flang: 886

62% fewer files

pgfortran: 945,313
Flang: 391,661

59% fewer LOC

*Clang has 212 #ifdefs in lib, include, tools
Flang Development Status

- Source code clean-up, refactoring & documentation ongoing
- Vendor neutrality nearly complete
- Frequent source and Flang binary updates to partners
- Passes most PGI Fortran Linux/x86 QA tests
- SIMD vectorization via the LLVM vectorizer, tuning ongoing
- Most of OpenMP 4.5 is implemented (CPU-side only)
Flang Source Code

Home page

Github

Doxygen
## Flang Single-core Performance

SPEC CPU 2006 Fortran codes, all times in seconds, 1 Haswell core

<table>
<thead>
<tr>
<th></th>
<th>PGI FORTRAN 16.10</th>
<th>GFORTRAN 6.1</th>
<th>FLANG DEV LLVM 3.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>410.bwaves</td>
<td>182s</td>
<td>220s</td>
<td>251s</td>
</tr>
<tr>
<td>416.gamess</td>
<td>507s</td>
<td>Fails</td>
<td>475s</td>
</tr>
<tr>
<td>434.zeusmp</td>
<td>183s</td>
<td>221s</td>
<td>240s</td>
</tr>
<tr>
<td>436.cactusADM</td>
<td>165s</td>
<td>194s</td>
<td>208s</td>
</tr>
<tr>
<td>437.leslie3d</td>
<td>179s</td>
<td>209s</td>
<td>435s</td>
</tr>
<tr>
<td>454.calculix</td>
<td>171s</td>
<td>297s</td>
<td>608s</td>
</tr>
<tr>
<td>459.GemsFDTD</td>
<td>261s</td>
<td>286s</td>
<td>391s</td>
</tr>
<tr>
<td>465.tonto</td>
<td>295s</td>
<td>373s</td>
<td>Fails</td>
</tr>
<tr>
<td>481.wrf</td>
<td>157s</td>
<td>271s</td>
<td>247s</td>
</tr>
</tbody>
</table>

**PGI Fortran:** `-fast -Mfprelaxed -Mstack_arrays`  
**gfortran:** `-O3 -funroll-loops -fpeel-loops -ffast-math`  
**Flang:** `-O3 -march=core-avx2 -ffp-contract=fast -Knoieee`  

Performance measured November, 2016 and are considered estimates per SPEC run and reporting rules. SPEC® and SPEC CPU® are registered trademarks of the Standard Performance Evaluation Corporation (www.spec.org).
# Flang OpenMP Performance

**SPEC OMP 2012 Fortran codes, all times in seconds, 32 Haswell cores (64 threads)**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>PGI FORTRAN 16.10</th>
<th>GFORTRAN 6.1</th>
<th>FLANG DEV LLVM 3.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>350.md</td>
<td>517s</td>
<td>3460s</td>
<td>459s</td>
</tr>
<tr>
<td>351.bwaves</td>
<td>469s</td>
<td>519s</td>
<td>805s</td>
</tr>
<tr>
<td>357.bt331</td>
<td>449s</td>
<td>492s</td>
<td>474s</td>
</tr>
<tr>
<td>360.ilbdc</td>
<td>541s</td>
<td>6846s</td>
<td>539s</td>
</tr>
<tr>
<td>362.fma3d</td>
<td>575s</td>
<td>504s</td>
<td>656s</td>
</tr>
<tr>
<td>363.swim</td>
<td>633s</td>
<td>634s</td>
<td>632s</td>
</tr>
<tr>
<td>370.mgrid</td>
<td>693s</td>
<td>697s</td>
<td>690s</td>
</tr>
<tr>
<td>371.applu</td>
<td>451s</td>
<td>414s</td>
<td>514s</td>
</tr>
</tbody>
</table>

**PGI Fortran:** `-fast -mp -Mfprelaxed -Mstack_arrays`  
**gfortran:** `-O3 -funroll-loops -fpeel-loops -ffast-math -fopenmp`  
**Flang:** `-O3 -mp -march=core-avx2 -ffp-contract=fast -Knoieee`  
**All:** `OMP_NUM_THREADS=64 OMP_PROC_BIND=true`  

Performance measured November, 2016 and are considered estimates per SPEC run and reporting rules. SPEC® and SPEC OMP® are registered trademarks of the Standard Performance Evaluation Corporation (www.spec.org).
Flang Year 2 Development Plans

- **Source code**
  - Continue source clean-up, refactoring, documentation
  - Create repository and release as open source
  - Deploy an open source testing infrastructure

- **Features**
  - Enhance compile-time Fortran error/warning messages
  - Incremental F08 and OpenMP 4.5 features
  - LLVM enhancements to enable Fortran DWARF generation

- **Performance**
  - Incremental, likely to be reactive after initial pass is done
Concluding Thoughts

- LLVM is integral to HPC compilers at NVIDIA and PGI
- Fortran → First-class citizen in the LLVM community
- LLVM as a platform for out-of-tree developers