PGI® 2010 Compilers & Tools

Dave Norton
Dave.norton@pgroup.com
www.pgroup.com

Craig Toepfer
Craig.toepfer@pgroup.com
www.pgroup.com

NERSC/OLCF/NICS Cray XT5 Workshop
Lawrence Berkeley National Lab
February 2010
HPC Hardware Trends

Today: Clusters of Multicore x86
Tomorrow? Clusters of Multicore x86 + Accelerators
## PGI Workstation / Server / CDK

Linux, Windows, MacOS, 32-bit, 64-bit, Intel 64, AMD64

UNIX-heritage Command-level Compilers + Graphical Tools

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Language</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGF95™</td>
<td>Fortran 95 w/some F2003</td>
<td>pgf95</td>
</tr>
<tr>
<td>PGCC®</td>
<td>ANSI C 99, K&amp;R C and GNU gcc Extensions</td>
<td>pgcc</td>
</tr>
<tr>
<td>PGC++®</td>
<td>ANSI/ISO C++</td>
<td>pgCC</td>
</tr>
<tr>
<td>PGDBG®</td>
<td>MPI/OpenMP debugger</td>
<td>pgdbg</td>
</tr>
<tr>
<td>PGPROF®</td>
<td>MPI/OpenMP profiler</td>
<td>pgprof</td>
</tr>
</tbody>
</table>

Self-contained OpenMP/MPI Development Solution
do j = 1, ny_global
    ULAT_G(i,j) = (-90.0_rad + j*dlat)/rad
  enddo

! calculate grid spacings and other quantities
! compute here to avoid bad ghost cell values due to dropped land
blocks

else ! not lstlon_only

  !$OMP PARALLEL DO PRIVATE(this_block, i, j, jg, jml, dm)
  do n=1,nblocks_clinic
    this_block = get_block(blocks_clinic(n), n)
    do j=1,ny_block
      jg = this_block*j_glob(j)
      jml = jg + 1
      if (jml < 1) jml = ny_global
      do i=1, nx_block
        HIN(i,j,n) = dlon/radius/radian ! convert to cm
        HNE(i,j,n) = dlat/radius/radian ! convert to cm
      enddo
  enddo

  !$OMP END PARALLEL DO
PGI Compilers & Tools

- Compilers & Tools dedicated to scientific computing, where speed of generated code is #1 criteria
- Are focused around parallel/optimization technologies
  - State of the art Local and Global Optimizations
  - Superior automatic SIMD vectorization
  - Support of the latest OpenMP 3.0 standard
  - Automatic loop parallelization for multi-core CPUs
  - Whole-program and profile-guided optimizations
  - PGI Unified Binary technology to target different flavors of x64 architecture or heterogeneous architectures with a single binary
  - Graphical tools to Debug/Profile Multithreaded/Multiprocess applications
- Starting with PGI version 9.0
  - PGI Accelerator programming model implementation to address the GPGPU programming challenge
PGI® 2010 New Features

- **PGI Accelerator™ Programming Model**
  - High-level, Portable, Directive-based Fortran & C extensions (no C++, yet)
  - Supported on NVIDIA CUDA GPUs

- **PGI CUDA Fortran**
  - Extended PGI Fortran, co-defined by PGI and NVIDIA
  - Lower-level explicit NVIDIA CUDA GPU programming

- **PVF Windows/MSMPI Cluster/Parallel Debugging**
  - Debug Fortran & C MSMPI cluster applications
  - PGI Accelerator and CUDA Fortran support

- **Compiler Enhancements**
  - F2003 – several new language features
  - Latest EDG 4.1 C++ front-end – more g++/VC++ compatible
  - AVX code generation, code generator tuning

- **PGPROF Enhancements**
  - Uniform performance profiling across Linux, MacOS and Windows
  - x64+GPU performance profiling
  - Updated Graphical User Interface (GUI)
PGI 2010 Compilers
F2003/C++ Language Support

- PGI Fortran 2003 incremental features
  - *Initial release of PGI 2010*: pointer reshaping, procedure pointers and statement, abstract interfaces, ieee_exceptions module, ieee_arithmetic module
  - *Coming later in PGI 2010*: Object-oriented features

- PGC++/ PGCC enhancements
  - EDG release 4.1 front-end with enhanced GNU and Microsoft compatibility, extern inline support, improved BOOST support, thread-safe exception handling
Intel AVX Support

- **Intel Advanced Vector Extensions**
  - New instructions
  - Wider vector registers, up to 2X floating point performance

- PGI F03/C/C++ compilers will be ready when AVX-enabled systems become available

- Can run with Intel simulator today
  - For those who like to experiment or test for correctness
Invoking AVX on PGI 10.0

• pgfortran –tp sandybridge-64
• GNU Binutils 2.19.51 or newer
• Intel Software Development Emulator
## AVX Example: Vector Add

### FORTRAN

```fortran
subroutine vadd( a, b, c, n )
  real    a(n), b(n), c(n)
  integer i, n
  do i = 1, n
    c(i) = a(i) + b(i)
  enddo
end
```

### SSE

```assembly
x.LB1_438:
  movups (%r10,%rcx), %xmm0
  movups (%r9,%rcx), %xmm1
  addps %xmm0, %xmm1
  movups %xmm1, (%r8,%rcx)
  movups 16(%r10,%rcx), %xmm0
  movups 16(%r9,%rcx), %xmm1
  addps %xmm0, %xmm1
  movups %xmm1, 16(%r8,%rcx)
  addq $32, %rcx
  subl $8, %eax
  testl %eax, %eax
  jg .LB1_438
```

### AVX

```assembly
.LB1_477:
vmovups (%r9,%rcx), %ymm0
Vaddps (%r10,%rcx), %ymm0, %ymm1
vmovups %ymm1, (%r8,%rcx)
vmovups 32(%r9,%rcx), %ymm0
vaddps 32(%r10,%rcx), %ymm0, %ymm1
vmovups %ymm1, 32(%r8,%rcx)
addq $64, %rcx
subl $16, %eax
testl %eax, %eax
jg .LB1_477
```

The Portland Group
Cross-Platform and Mobile HPC Development

- **PGI Workstation® on Linux, MacOS & Windows**
  - Same C, C++, and Fortran compilers on all platforms
  - PGI Accelerator support and CUDA Fortran
  - MPI, OpenMP, PGDBG®, PGPROF®

- **Cross-platform licensing**
  - One license can cover all platforms
  - Floating license only

- **'Borrow' licensing**
  - No separate license needed to work off-line on your notebook
  - Check out a floating license for your notebook for home or travel
Multicore X64 Performance

SPEC® and SPECfp® are registered trademarks of the Standard Performance Evaluation Corporation (SPEC) (www.spec.org)

Competitive benchmark results stated above reflect results performed by The Portland Group during the week of November 8th, 2009.
The Intel Nehalem system used is a Dell R610 using 2 Intel Xeon X5550 with 24GB DDR3-1333. The AMD Istanbul system is a kit built 2 Opteron 2431 system with 32GB DDR2-800. Since this system is not generally available, the AMD results should be considered estimates.
SPEC and the –fast flag

- Earlier version of the SPEC benchmark only allowed 4 compilers flags for the “base” run. To accommodate this PGI introduced the –fast flag and used that to enable numerically safe, best practices flags.

- Intel and Pathscale also have convenience flags – but they differ from PGI in the optimizations they invoke and the side effects.

- PGI’s –fast is conservative and intended for everyday use. Intel and Pathscale’s convenience flags are intended to maximize SPEC performance without causing errors in SPEC codes. For regular user codes – YMMV
-fast comparison - PGI

- **PGI**: -fast typically includes these options:
  - O2 Specifies a code optimization level of 2.
  - Munroll Unrolls loops, executing multiple instances of the loop during each iteration.
  - Mnoframe Indicates to not generate code to set up a stack frame.
  - Mlre Indicates loop-carried redundancy elimination.
  - Mpre Indicates partial redundancy elimination.

These additional options are also typically available when using -fast for 64-bit targets:

- Mvect=sse Generates SSE instructions.
- Mscalarssse Generates scalar SSE code with xmm registers; implies –Mflushz.
- Mcache_align Aligns long objects on cache-line boundaries.
- Mflushz Sets SSE to flush-to-zero mode.
-fast comparison - Intel

- **Intel**: `-fast` This option maximizes speed across the entire program.
- **Linux**: `-ipo`, `-O3`, `-no-prec-div`, `-static`, and `--xHost`

- **IPO** - Interprocedural optimization between files
- **NO-PREC-DIV** - With some optimizations, such as `-xSSE2` (Linux) the compiler may change floating-point division computations into multiplication by the reciprocal of the denominator. For example, $A/B$ is computed as $A \times (1/B)$ to improve the speed of the computation. However, sometimes the value produced by this transformation is not as accurate as full IEEE division. When it is important to have fully precise IEEE division, use this option to disable the floating-point division-to-multiplication optimization. The result is more accurate, with some loss of performance. If you specify `-no-prec-div` it enables optimizations that give slightly less precise results than full IEEE division.

- **STATIC** - Prevents linking with shared libraries – it causes the executable to link all libraries statically

- **PGI Equivalents:**
  - IPO => `-Mipa=fast`
  - NO-PREC-DIV => `-Mfprelaxed`
  - STATIC => `-Bstatic`
-fast comparison - Pathscale

- Pathscale: -OPT:Ofast and -Ofast
  - The option -OPT:Ofast is equivalent to
    -OPT:roundoff=2:Olimit=0:div_split=ON:alias=typed.
  - -Ofast is equivalent to

“With -O3 -OPT:Ofast and -Ofast, you should look to see if the results are accurate.”

NO-MATH-ERRNO – turns off IEEE floating point error exception handling

ROUNDOFF=2 – allows for fairly extensive code transformations that may result in floating point round off or overflow differences in computations

DIV_SPLIT=ON – Allows conversion of x/y into x*(recip(y)) => less accurate

ALIAS=TYPED – Assumes that pointers don’t point to the same memory.

IPA – Interprocedural analysis

-O3 – additional optimizations that may or may not result in improved performance and may or may introduce numerical errors.

PGI Equivalents:

NO-MATH-ERRNO => -Knoieee
ALIAS=TYPED => -Msafeptr

-O3 – includes zero cost exception handling => -zc_eh
Optimized SPEC

• Code: Leslie3D
• Base flag: -fast –Mipa=fast,inline
• Base performance: 350s
• Optimized flag: -fast –Mipa=fast,inline -Mvect=fuse
• Optimized performance: 327s

• Reason the code runs faster: Fuses loops together for increased vectorization.
Optimized SPEC

- Code: Hummer
- Base flag: -fast -Mipa=fast,inline
- Base performance: 402s
- Optimized flag: -fast -Mipa=fast,inline -Msafeptr
- Optimized performance: 383s
- Additional flags: -Mvect=partial
- Additional flags performance: 297s

- Reason the code runs faster: Once the compiler knows that pointers don’t overlap, it can vectorize the code for better performance
- Reason additional flag runs faster: Loop has conditionals in it which Compiler doesn’t generally vectorize these types of loops as metrics show it often isn’t helpful. However – in the case of this code it is beneficial.
Optimized SPEC

- Code: MCF
- Base flag: -fast –Mipa=fast,inline
- Base performance: 573s
- Optimized flag: -fast –Mipa=fast,inline –Msmartalloc=huge
- Optimized performance: 282s

Reason the code runs faster: Code allocates a large block of memory at initialization and then manages that memory itself
Optimized SPEC

- Code: Gromacs
- Base flag: -fast –Mipa=fast,inline
- Base performance: 620s
- Optimized flag: -fast –Mipa=fast,inline -Mfprelaxed
- Optimized performance: 400s

Reason the code runs faster: Code does less precise arithmetic then then IEEE standard. This reduction results in improved performance while not causing enough loss of precision to effect the programs answers substantially.
Important PGI Compiler Options

- **-fast** Includes “-fast -Mvect=sse -Mcache_align -Mnoframe -Mlre”

- **-Mipa=fast** Enable inter-procedural analysis (IPA) and optimization

- **-Mipa=fast,inline** Enable IPA-based optimization and function inlining

- **-Mfprelaxed** Reduced precision arithmetic operations

- **-Minline** Inline functions and subroutines

- **-Mconcur** Try to auto-parallelize loops for SMP/Dual-core systems

- **-mp[=align]** Process OpenMP/SGI directives and pragmas

- **-mcmodel=medium** Enable data > 2GB on AMD64/EM64T running 64-bit Linux

- **-Minfo** Compile-time optimization/parallelization messages

- **-Mneginfo** Compile-time messages indicating what prevented an optimization

- **-help** Compiler options and usage

- **-fast -Mipa=fast -Minfo** usually best for “compile-and-go”
Byteswapio Optimization

ORIGINAL

```
Open(200,status='new',file='binary.data',
    &     form='UNFORMATTED',
    &     recl=numdouble*8, access='direct')
write(200,rec=1) doublearray
Close(200)
```

MODIFIED

```
Open(200,status='new',file='binary.data',
    &     form='UNFORMATTED',
    &     recl=numdouble*8, access='direct')
    istat = setvbuf(200,0,numdouble*8,user_buf)
write(200,rec=1) doublearray
Close(200)
```

• Use setvbuf to override default runtime buffer settings.
• On the XT5 setvbuf preferred over setvbuf3f.
• See the fortran reference manual for more info.
Availability and Additional Information

- **PGI 2010 Compilers & Tools** – available now! See [www.pgroup.com](http://www.pgroup.com) for details

- **PGI Accelerator programming model** – supported for x64+NVidia targets in the PGI 2010 F95/03 and C99 compilers, available now; see [http://www.pgroup.com/accelerate](http://www.pgroup.com/accelerate) for a detailed specification, FAQ and related articles and white papers

- **CUDA Fortran** – supported on NVidia GPUs in PGI 2010 F95/03 compiler; see [http://www.pgroup.com/resources/cudafortran.htm](http://www.pgroup.com/resources/cudafortran.htm) for a detailed specification
Task Example

• Uses OpenMP 3.0 tasks
• Actual use by PGI compiler when specifying the –Mcuda=emu compiler option (CUDA Fortran emulation mode)
• Analogous to the thread execution control unit on NVIDIA GPUs
What is the PGI Accelerator Model?

- High-level, Portable, Directive-based Fortran & C extensions (no C++, yet)
- Supported on NVIDIA CUDA enabled GPUs
- Supported on Linux, MacOS and Windows
What is CUDA Fortran?

- CUDA Fortran is an analog to NVIDIA's CUDA C language
- Co-defined by PGI and NVIDIA, implemented in the PGI 2010 Fortran 95/03 compiler
- Supported on Linux, MacOS and Windows
- CUDA Fortran gives HPC developers direct control over all aspects of GPU programming
PGI Accelerator vs CUDA Fortran?

- The **PGI Accelerator** programming model is a high-level *implicit* model for x64+GPU systems, similar to OpenMP for multi-core
  - Supported in both the PGI F95/03 and C99 compilers
  - Offload compute-intensive code to a GPU accelerator using directives
  - Programs remain 100% standard-compliant and portable
  - Makes GPGPU programming and optimization incremental and accessible to application domain experts

- **CUDA Fortran** is a lower-level *explicit* model for direct control of:
  - Splitting source into host code and GPU kernel subroutines/functions
  - Allocation of page-locked host memory, GPU device main memory, GPU constant memory and GPU shared memory
  - All data movement between host memory and GPU memory hierarchy
  - Definition of thread/block grids and launching of compute kernels
  - Synchronization of threads within a CUDA thread group
  - Asynchronous launch of GPU kernels, synchronization with host CPU
  - All CUDA Runtime API features and functions