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

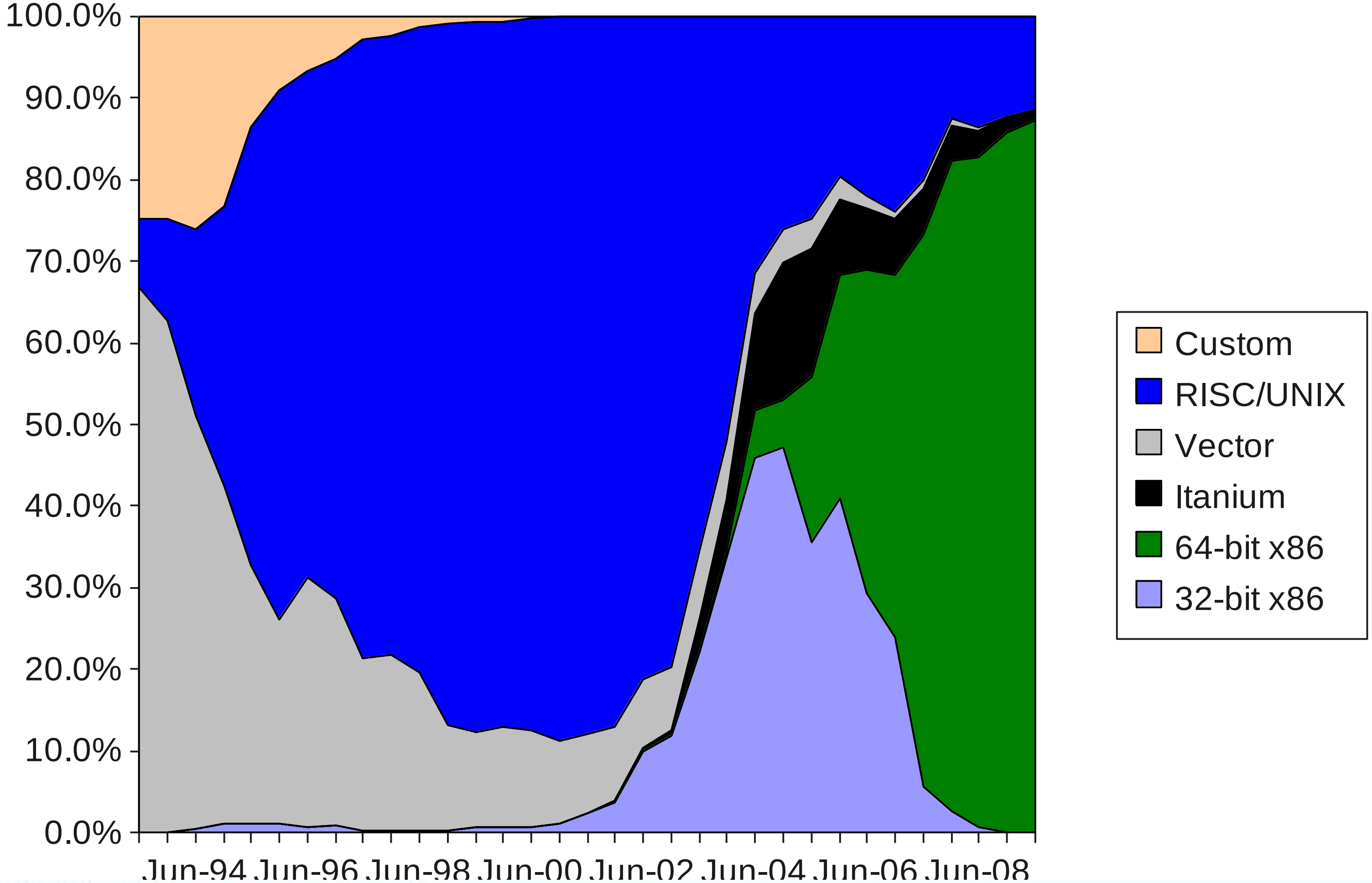
Lawrence Berkeley National Lab

February 2010

HPC Hardware Trends

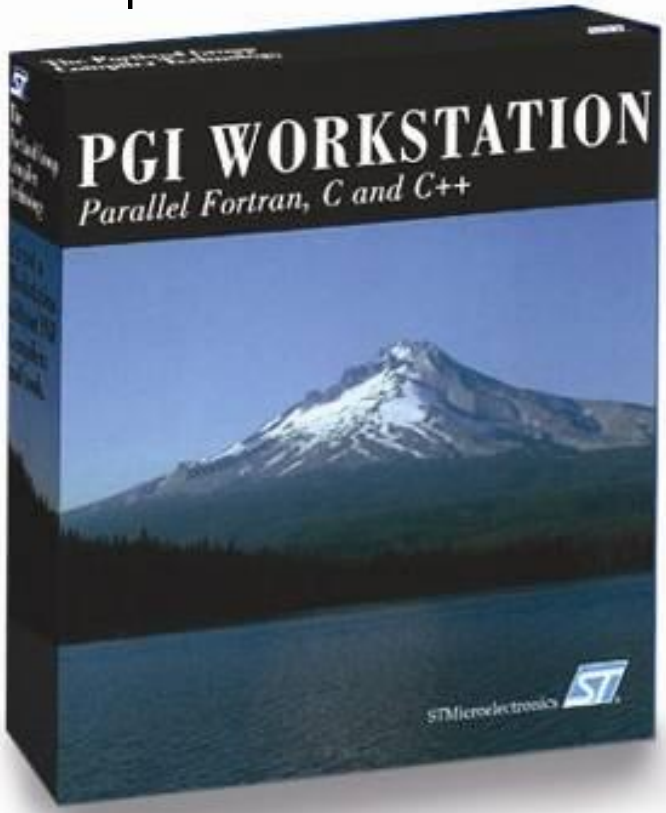
Today: Clusters of Multicore x86
Tomorrow? Clusters of Multicore x86 + Accelerators

Top 500



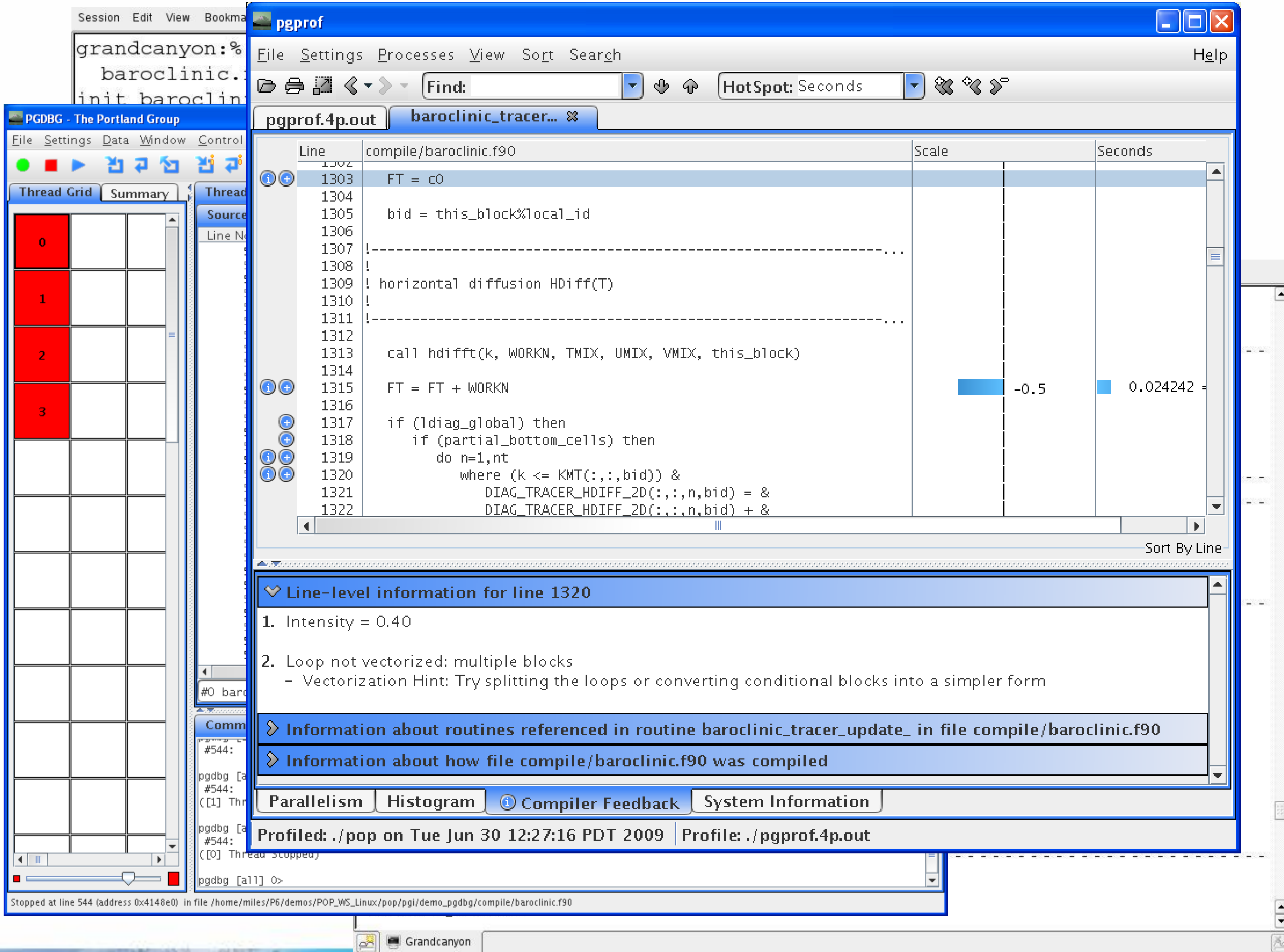
PGI Workstation / Server / CDK

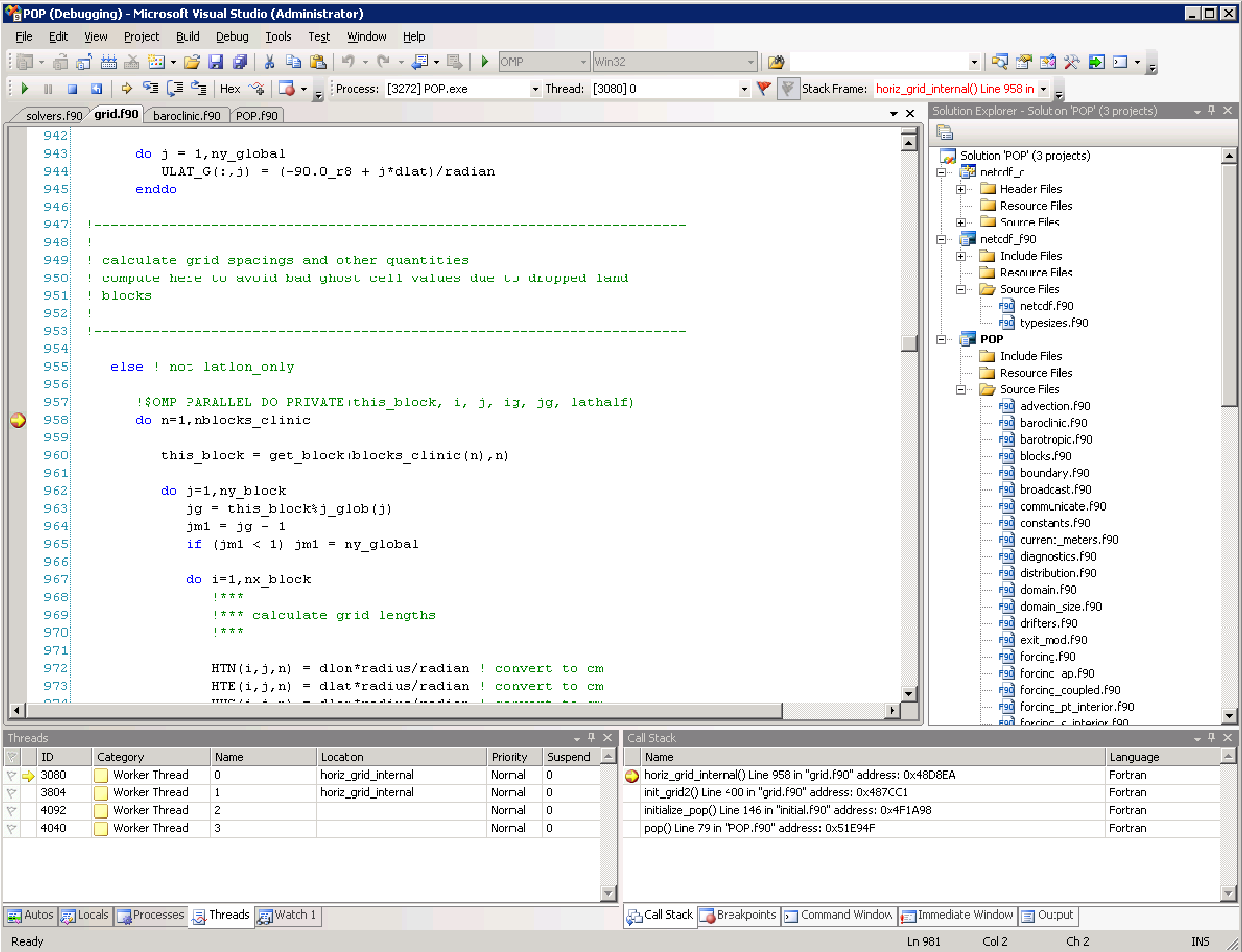
Linux, Windows, MacOS, 32-bit, 64-bit, Intel 64, AMD64
UNIX-heritage Command-level Compilers + Graphical Tools



Compiler	Language	Command
PGF95™	Fortran 95 w/some F2003	pgf95
PGCC®	ANSI C 99, K&R *C* and *GNU gcc Extensions*	pgcc
PGC++®	ANSI/ISO *C++*	pgCC
PGDBG®	MPI/OpenMP debugger	pgdbg
PGPROF®	MPI/OpenMP profiler	pgprof

Self-contained OpenMP/MPI Development Solution


pgprof
File Settings Processes View Sort Search Help
Find:
HotSpot: Seconds
pgprof.4p.out
baroclinic_tracer...
Line
compile/baroclinic.f90
Scale
Seconds
1303 FT = c0
1304
1305 bid = this_block%local_id
1306
1307 !---...
1308 !
1309 ! horizontal diffusion HDiff(T)
1310 !
1311 !---...
1312
1313 call hdifft(k, WORKN, TMIX, UMIX, VMIX, this_block)
1314
1315 FT = FT + WORKN
-0.5
0.024242
1316
1317 if (ldiag_global) then
1318 if (partial_bottom_cells) then
1319 do n=1,nt
1320 where (k <= KMT(:,:,bid)) &
1321 DIAG_TRACER_HDIFF_2D(:,:,n,bid) = &
1322 DIAG_TRACER_HDIFF_2D(:,:,n,bid) + &
Sort By Line
Line-level information for line 1320
1. Intensity = 0.40
2. Loop not vectorized: multiple blocks
– Vectorization Hint: Try splitting the loops or converting conditional blocks into a simpler form
Information about routines referenced in routine baroclinic_tracer_update_ in file compile/baroclinic.f90
Information about how file compile/baroclinic.f90 was compiled
Parallelism
Histogram
Compiler Feedback
System Information
Profiled: ./pop on Tue Jun 30 12:27:16 PDT 2009
Profile: ./pgprof.4p.out
PGDBG - The Portland Group
File Settings Data Window Control
Thread Grid
Summary
0
1
2
3
Stopped at line 544 (address 0x4148e0) in file /home/miles/P6/demos/POP_WS_Linux/pop/pgi/demo_pgdbg/compile/baroclinic.f90
Session Edit View
grandcanyon:%
baroclinic.
init_baroclin
Grandcanyon


POP (Debugging) - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Tools Test Window Help
OMP
Win32
Hex
Process: [3272] POP.exe
Thread: [3080] 0
Stack Frame: horiz_grid_internal() Line 958 in
solvers.f90
grid.f90
baroclinic.f90
POP.f90
do j = 1,ny_global
ULAT_G(:,j) = (-90.0_r8 + j*dlat)/radian
enddo
!---
!
! calculate grid spacings and other quantities
! compute here to avoid bad ghost cell values due to dropped land
! blocks
!
!---
else ! not latlon_only
!$OMP PARALLEL DO PRIVATE(this_block, i, j, ig, jg, lathalf)
do n=1,nblocks_clinic
this_block = get_block(blocks_clinic(n),n)
do j=1,ny_block
jg = this_block%j_glob(j)
jm1 = jg - 1
if (jm1 < 1) jm1 = ny_global
do i=1,nx_block
!***
!*** calculate grid lengths
!***
HTN(i,j,n) = dlon*radius/radian ! convert to cm
HTE(i,j,n) = dlat*radius/radian ! convert to cm
Solution Explorer - Solution 'POP' (3 projects)
Solution 'POP' (3 projects)
netcdf_c
Header Files
Resource Files
Source Files
netcdf_f90
Include Files
Resource Files
Source Files
netcdf.f90
typesizes.f90
POP
Include Files
Resource Files
Source Files
advection.f90
baroclinic.f90
barotropic.f90
blocks.f90
boundary.f90
broadcast.f90
communicate.f90
constants.f90
current_meters.f90
diagnostics.f90
distribution.f90
domain.f90
domain_size.f90
drifters.f90
exit_mod.f90
forcing.f90
forcing_ap.f90
forcing_coupled.f90
forcing_pt_interior.f90
Threads
ID Category Name Location Priority Suspend
3080 Worker Thread 0 horiz_grid_internal Normal 0
3804 Worker Thread 1 horiz_grid_internal Normal 0
4092 Worker Thread 2 Normal 0
4040 Worker Thread 3 Normal 0
Call Stack
Name Language
horiz_grid_internal() Line 958 in "grid.f90" address: 0x48D8EA Fortran
init_grid2() Line 400 in "grid.f90" address: 0x487CC1 Fortran
initialize_pop() Line 146 in "initial.f90" address: 0x4F1A98 Fortran
pop() Line 79 in "POP.f90" address: 0x51E94F Fortran
Autos
Locals
Processes
Threads
Watch 1
Call Stack
Breakpoints
Command Window
Immediate Window
Output
Ready
Ln 981
Col 2
Ch 2
INS

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
  - http://software.intel.com/en-us/articles/intel-software-development-emulator/

AVX Example: Vector Add

FORTRAN	SSE	AVX

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
	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
		.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

Cross-Platform and Mobile HPC Development

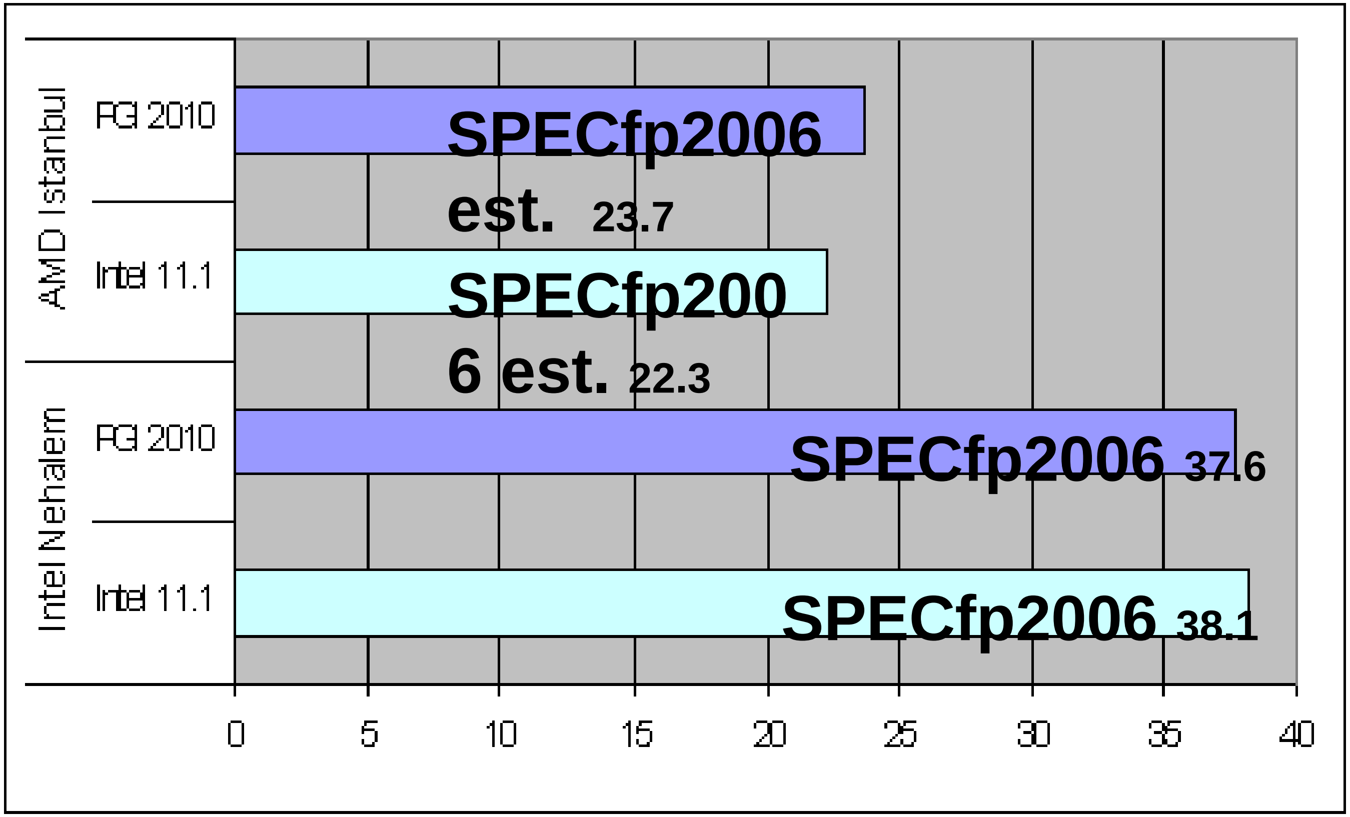
- PGI Workstation® on Linux, MacOS & Windows
  - Same C, C++, and Fortran compilers on all platforms
  - PGI Accelerator support and CUDA Fortran
  - MPI, OpenMP, PGDBG®, PGPROF®

- 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.
- –Mscalarsse 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 * (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

  -O3 -ipa -OPT:Ofast -fno-math-errno.

"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 does'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 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 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 for a detailed specification

Task Example

- Uses OpenMP 3.0 tasks
- Actual use by PGI compiler when specifing 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