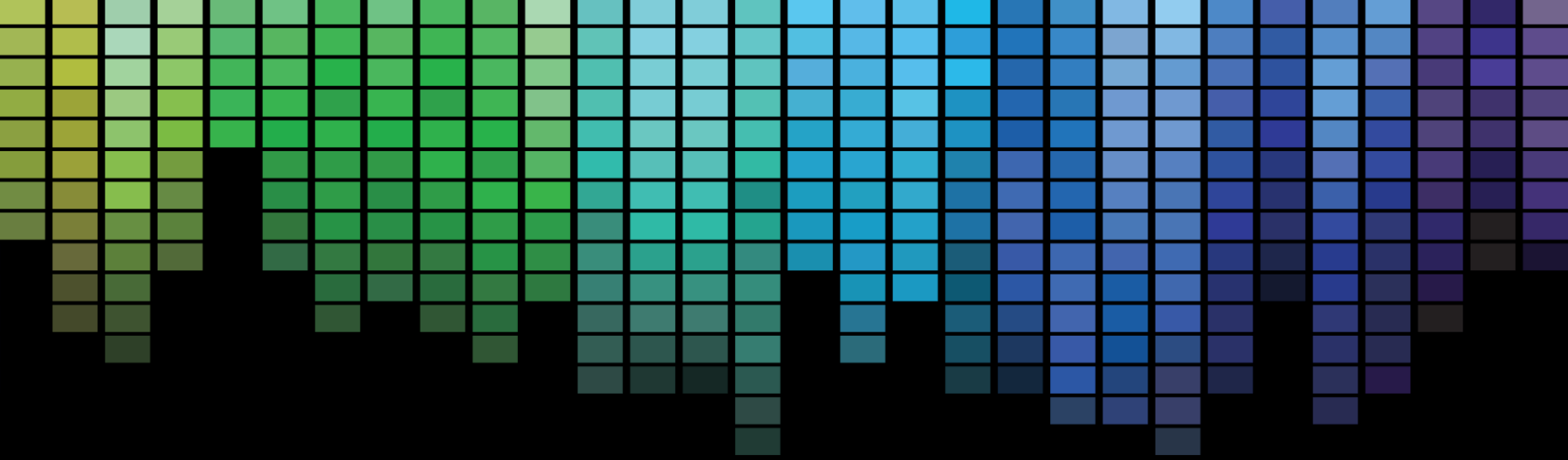


Porting to OpenPOWER+Tesla with PGI

Brent Leback, PGI Service Manager, brent.leback@pgroup.com

PGI®

PGI for OpenPOWER+Tesla

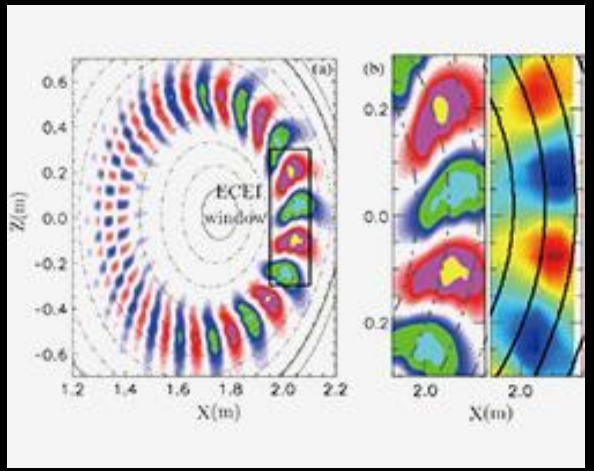
- Feature parity with PGI Compilers on Linux/x86+Tesla
- CUDA Fortran, OpenACC, OpenMP, NVCC host compiler
- Integrated with LLVM OpenPOWER code generator
- Now in production

PGI for OpenPOWER+Tesla

- Feature parity with PGI Compilers on Linux/x86+Tesla
- CUDA Fortran, OpenACC, OpenMP, NVCC host compiler
- Integrated with LLVM OpenPOWER code generator
- Now in production

PGI

Gyrokinetic Toroidal Code (GTC)



Particle Turbulence Simulations for Sustainable Fusion Reactions in ITER

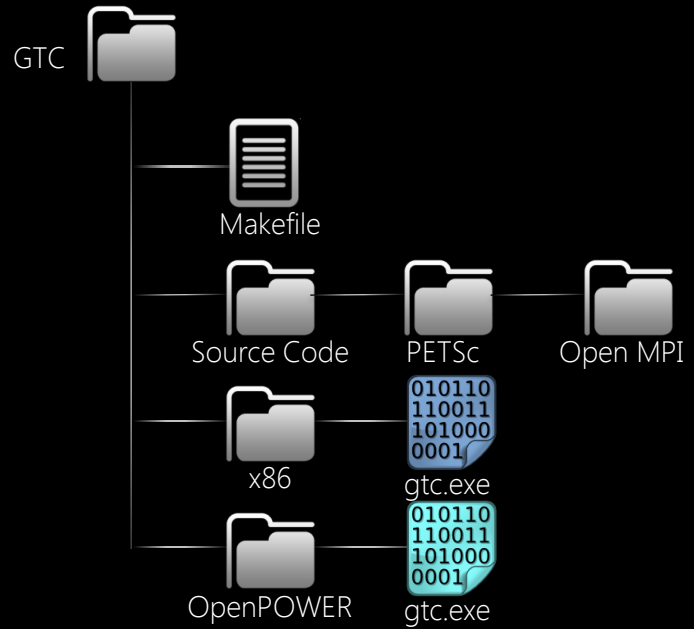
Science Domain: Plasma Physics

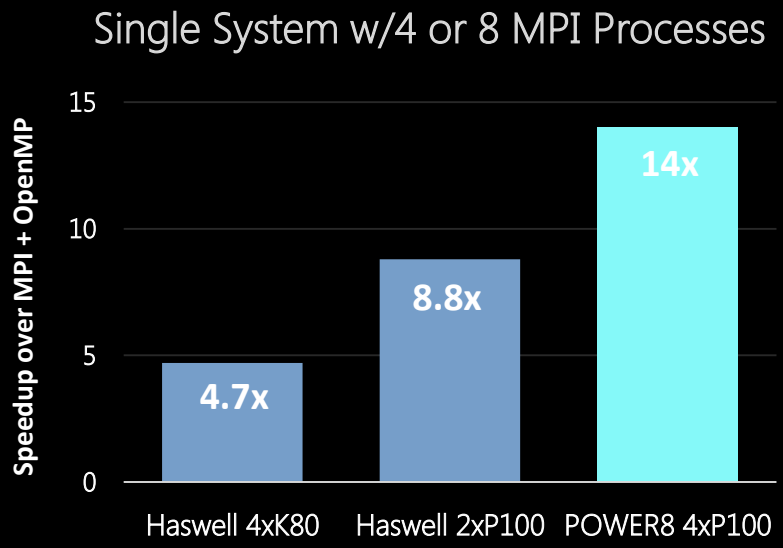
PI: Zhihong Lin, University of California–Irvine

PGI®

Re-compile and Run

Porting GTC from Xeon to OpenPOWER

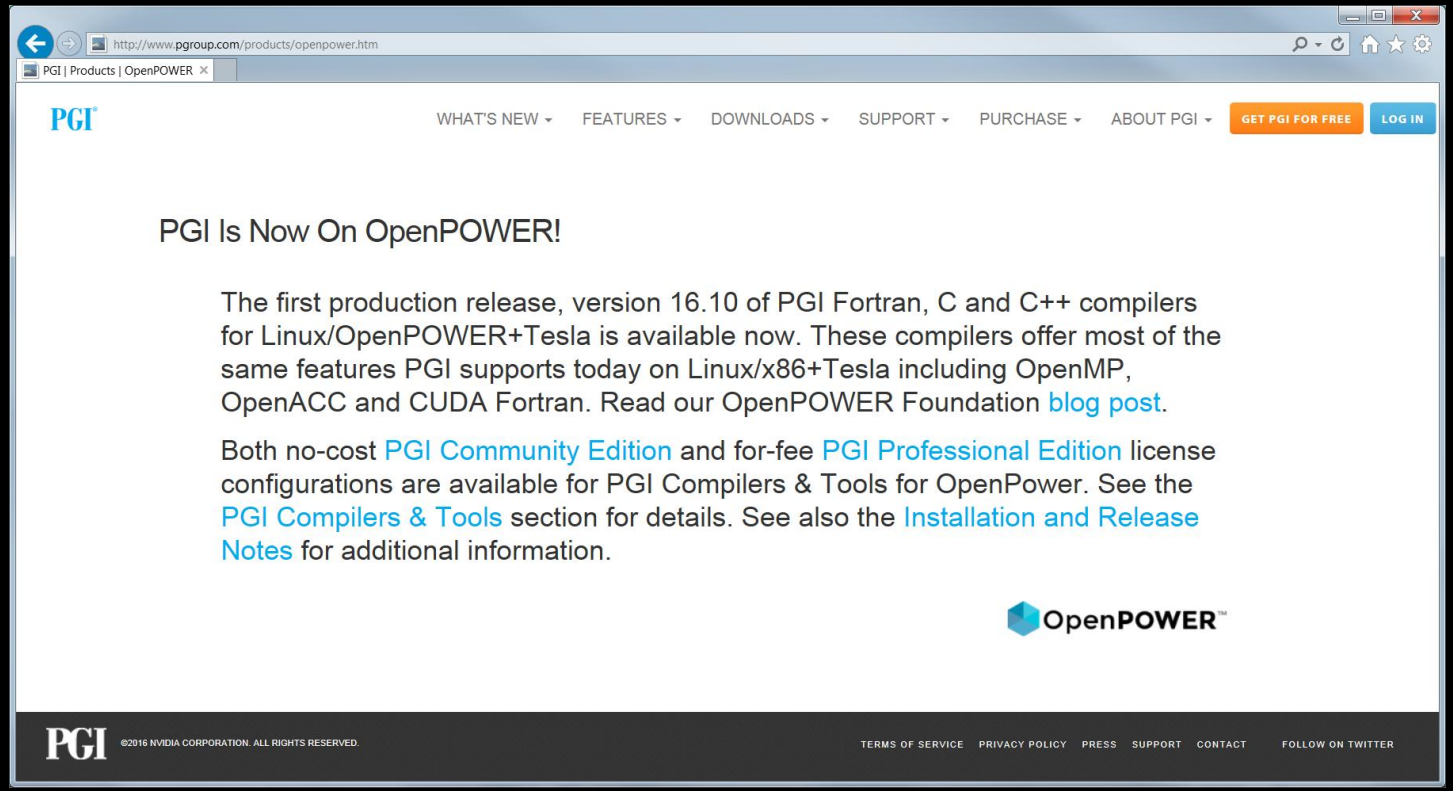




X86 CPU: Intel Xeon E5-2698 v3,
POWER CPU: IBM POWER8NVL

PGI®

PGI For OpenPower is Available Now



PGI

www.pgroup.com/openpower

OpenACC for Everyone

New PGI Community Edition Now Available

	FREE PGI Community Edition	PGI Professional Edition	PGI Enterprise Edition
PROGRAMMING MODELS OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools	✓	✓	✓
PLATFORMS x86, OpenPOWER, NVIDIA GPU	✓	✓	✓
UPDATES	1-2 times a year	6-9 times a year	6-9 times a year
SUPPORT	User Forums	PGI Support	PGI Premier Services
LICENSE	Annual	Perpetual	Volume/Site

PGI

3rd Party Packages for PGI on OpenPOWER

- Open MPI 1.10.2
- ScaLAPACK 2.0.2
- OpenBLAS 0.2.18

Bundled

- MPICH 3.2
- HDF5 1.8.15-patch1
- NetCDF 4.3.3.1
- NetCDF-C++ 4.2.1
- NetCDF-Fortran 4.4.2
- CURL 7.46.0
- ZLIB 1.2.8
- SZIP 2.1

Binaries available

- FFTW 2.1.5 / 3.3.4

Build instructions available

PGI®

Getting Started Using PGI

```
export PGI=/opt/pgi
type arch > /dev/null 2>&1
if test $? -ne 0 ; then
    alias arch="uname -m"
fi
case "`arch`" in
    ppc64le )  arch=linuxpower ;;
    x86_64 )   arch=linux86-64 ;;
     * )           arch= ;;
esac
export MYPGIVERSION=/$arch/16.10
export PATH=${PGI}${MYPGIVERSION}/bin:${PATH}
```

PGI®

PGI Porting Guidelines

C/C++ ABI differences

- Signed vs unsigned default char
- Long double

Platform differences

- Large memory model
  - Remove –mcmodel=medium from makefiles
- C varargs differences

X86-specific features

- Inlined assembly
- SSE/AVX intrinsics

Numerical differences

- Intrinsics may differ across x86, OpenPOWER, and GPUs
- FMA vs no FMA
  - Control with -Mnofma

C/C++ Default Char Differences

```
brentl@gsn1:~> cat defaultchar.c
#include "stdio.h"
#include "ctype.h"
int main()
{
  char c = 'a';
  while (c >= 0) {
    if (isprint(c))
      printf("%c",c);
    c++;
  }
  printf("\n");
}
brentl@gsn1:~> pgcc defaultchar.c
brentl@gsn1:~> ./a.out
abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQ
RSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123456789:;<=>?@AB
. . .
```

```
brentl@hsw1:~> pgcc defaultchar.c
brentl@hsw1:~> ./a.out
abcdefghijklmnopqrstuvwxyz{|}~
brentl@hsw1:~>
```

PGI®

C/C++ Default Char Differences

TWO SOLUTIONS:

```
brentl@gsn1:~> cat defaultchar.c
#include "stdio.h"
#include "ctype.h"
int main()
{
  signed char c = 'a';
  while (c >= 0) {
    if (isprint(c))
      printf("%c",c);
    c++;
  }
  printf("\n");
}
brentl@gsn1:~> pgcc -Mschar defaultchar.c
```

PGI®

C/C++ Long Double Differences

```
brentl@gsn1:~> cat longdouble.c
#include "stdio.h"
#include "math.h"
int main()
{
  printf("%2d %36.32e\n",sizeof(float),expf(1.0F));
  printf("%2d %36.32e\n",sizeof(double),exp(1.0));
  printf("%2d %36.32Le\n",sizeof(long double),expl(1.0L));
}
```

```
brentl@gsn1:~> pgcc longdouble.c
brentl@gsn1:~> ./a.out
 4 2.71828174591064453125000000000000e+00
 8 2.71828182845904509079559829842765e+00
16 2.71828182845904523536028747135266e+00
brentl@gsn1:~>
```

```
brentl@hsw1:~> pgcc longdouble.c
brentl@hsw1:~> ./a.out
 4 2.71828174591064453125000000000000e+00
 8 2.71828182845904509079559829842765e+00
16 2.71828182845904523542816810799394e+00
```

PGI®

C/C++ Long Double Differences

Solution:

- Say “Thank you” for the increased precision in long double.
- Don’t rely on more or less precision in long double.

PGI®

Variadic Function Differences

```
brentl@gsn1:~> cat var.c
/*            */
#include "math.h"
int main()
{
  printf("%2d %36.32e\n",sizeof(float),expf(1.0F));
  printf("%2d %36.32e\n",sizeof(double),exp(1.0));
  printf("%2d %36.32Le\n",sizeof(long double),expl(1.0L));
}

brentl@gsn1:~> pgcc var.c
brentl@gsn1:~> ./a.out
 4 5.26354424712089031287393635327398e-315
 8 2.37151510003798341204753020576746e-322
16 6.71929278344095300080133558300781e-322
brentl@gsn1:~>
```

```
brentl@hsw1:~> pgcc var.c
brentl@hsw1:~> ./a.out
 4 2.71828174591064453125000000000000e+00
 8 2.71828182845904509079559829842765e+00
16 2.71828182845904523542816810799394e+00
```

PGI®

Variadic Function Differences

Problem:

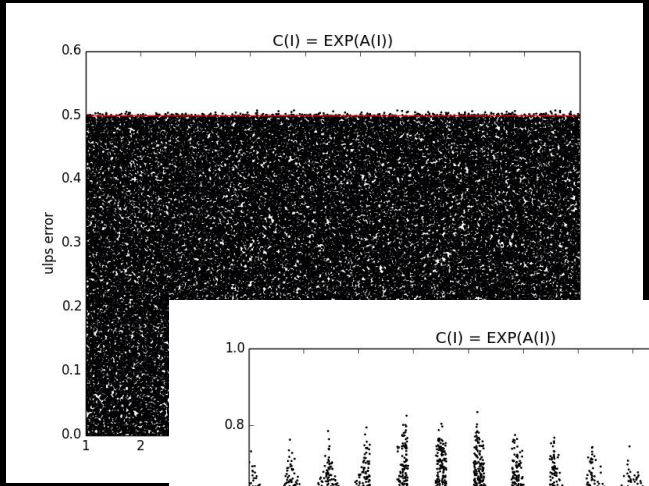
- Floating point values are passed differently to a function known to be “varargs”.

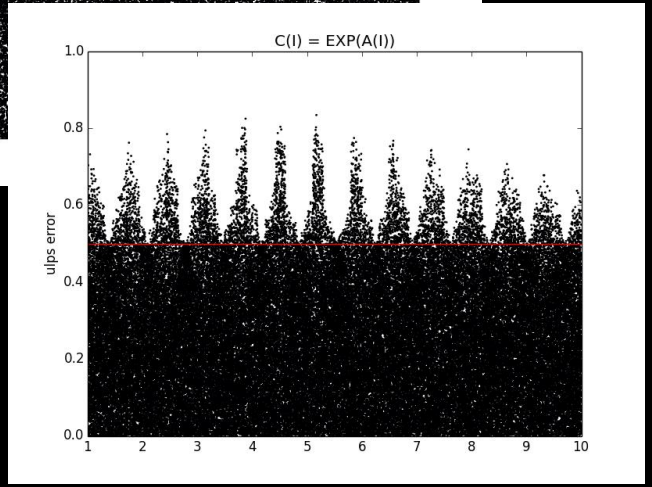
Solution:

- gcc or pgc++ will issue a warning or error for missing function declarations.
- Always include stdio.h or other headers.
- Outstanding pgcc RFE.

Single Precision exp()

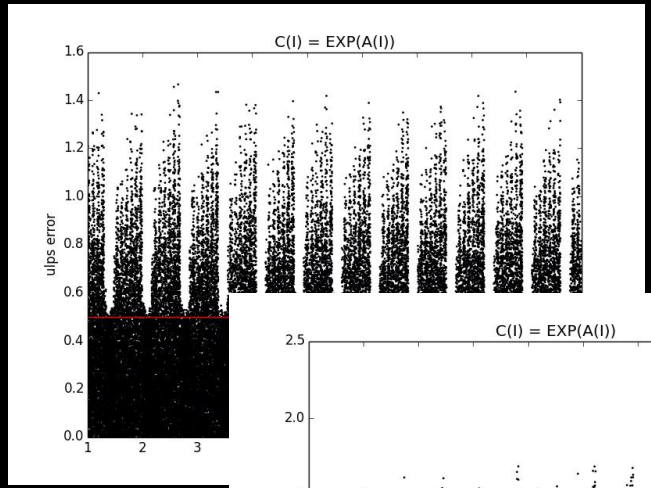
Bulldozer

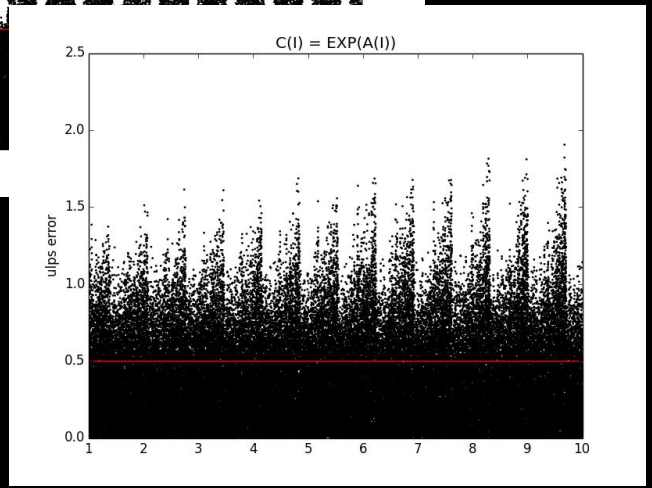




Haswell

POWER8



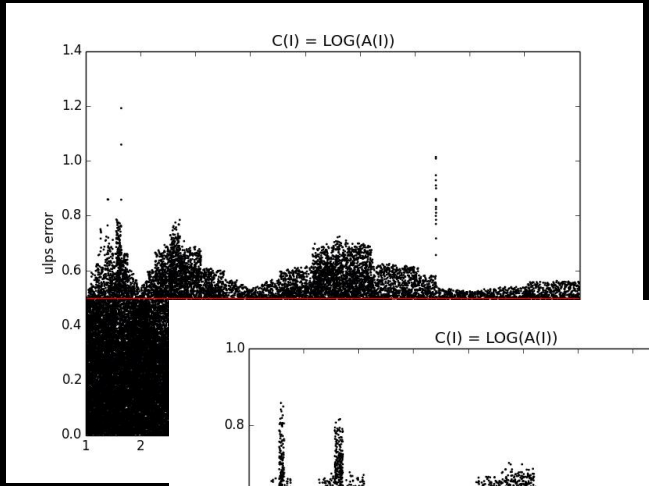


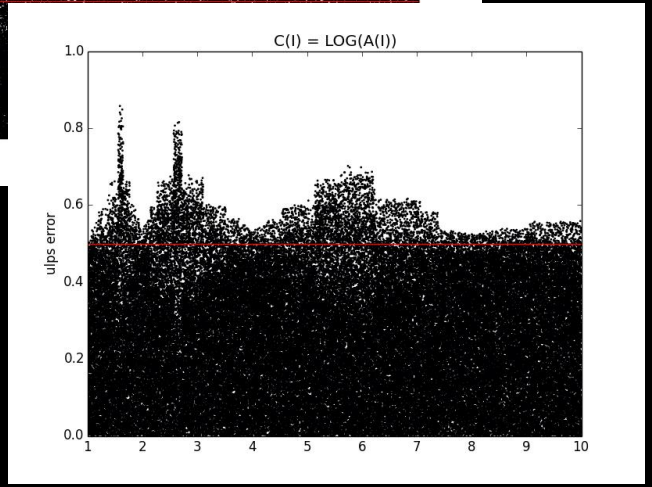
Tesla

PGI®

Single Precision log()

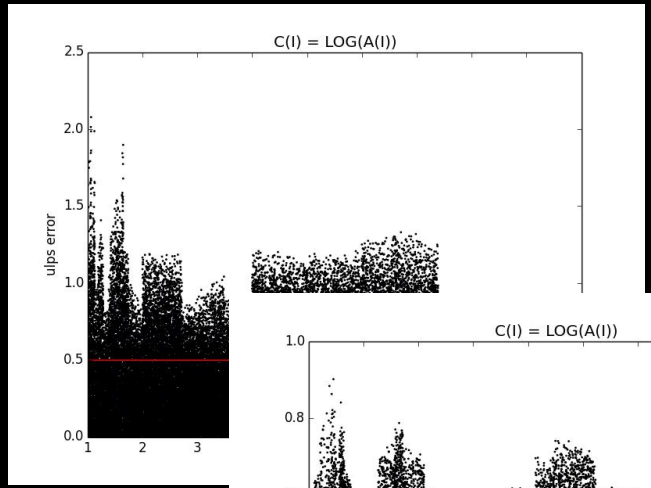
Bulldozer

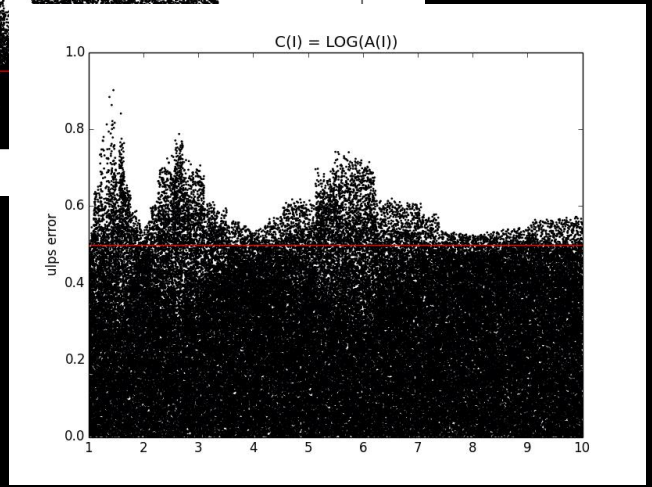




Haswell

POWER8



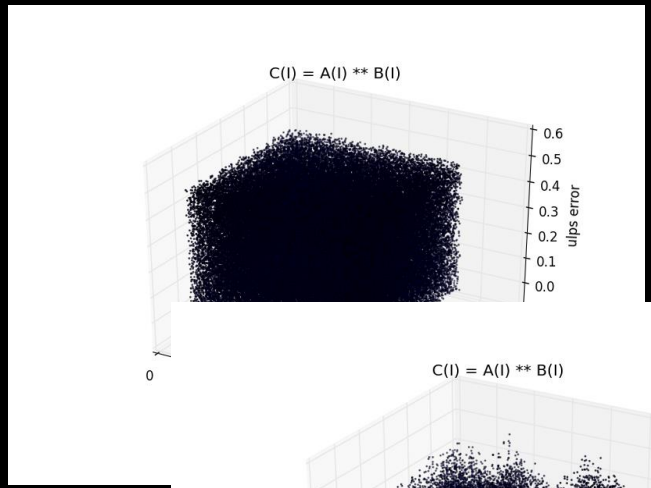


Tesla

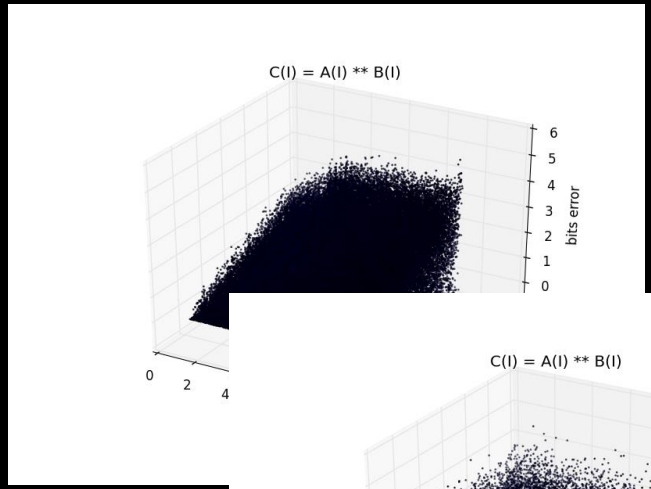
PGI®

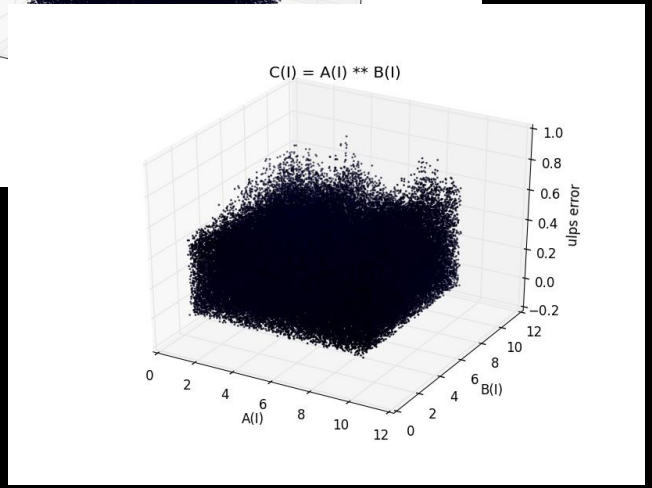
Single Precision pow()

Bulldozer

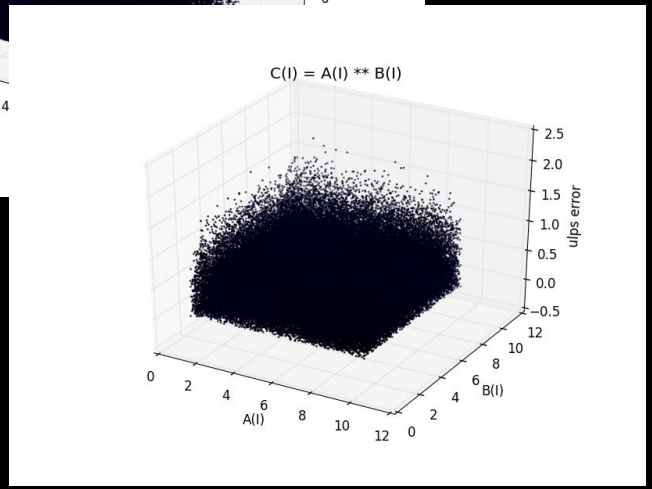


POWER8





Haswell

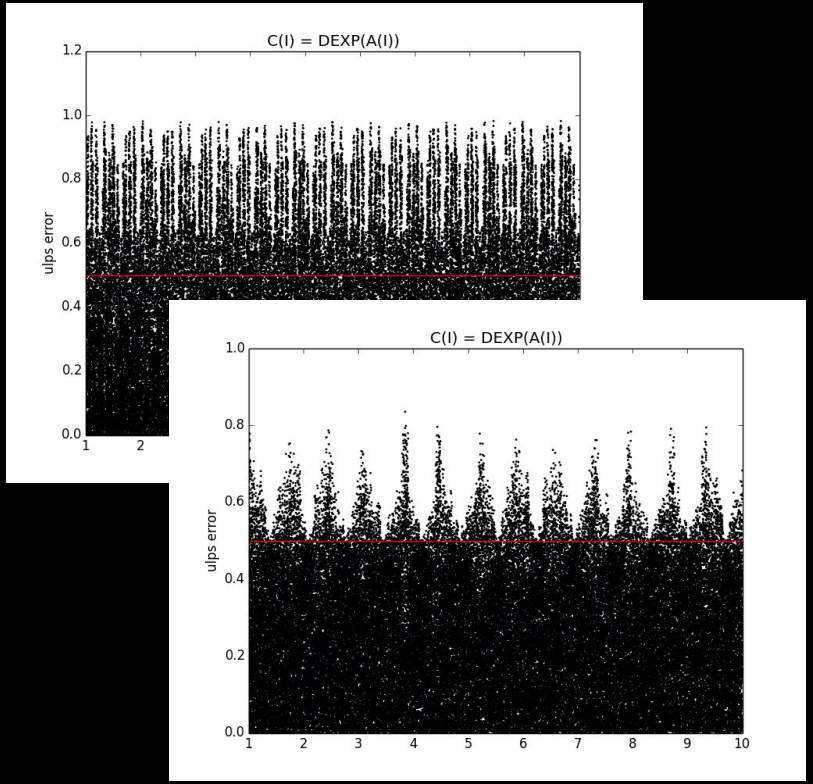


Tesla

PGI®

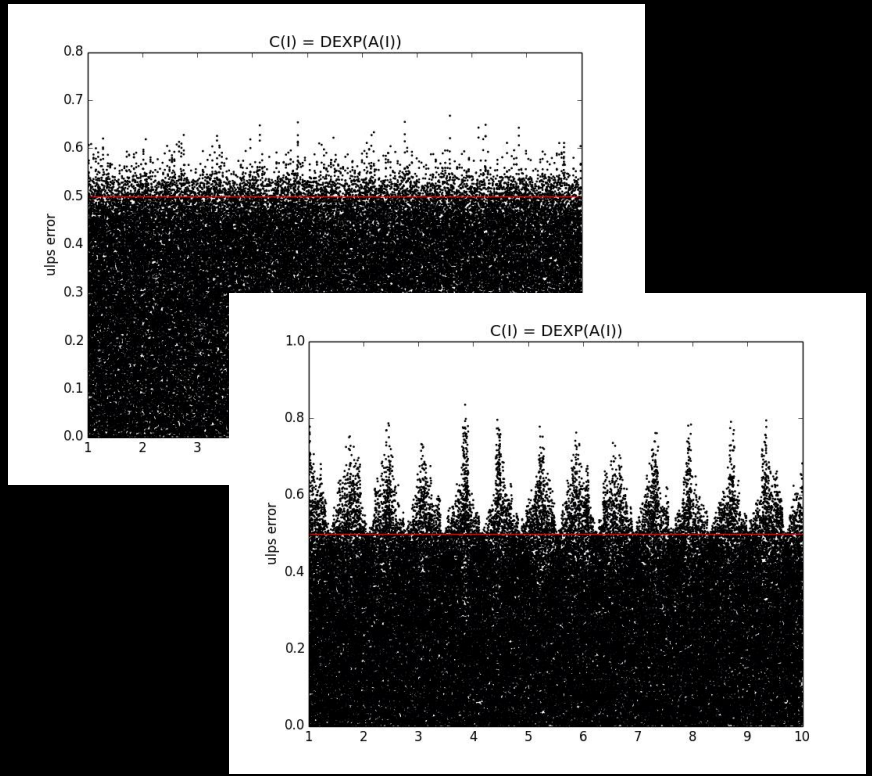
Double Precision exp()

Bulldozer



Haswell 17.1

POWER8

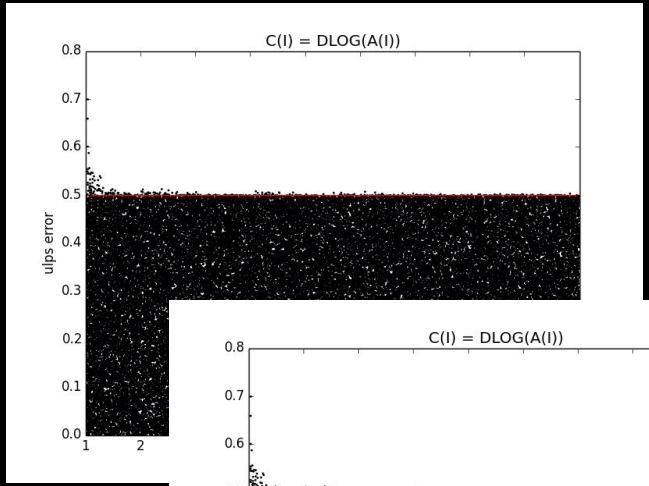


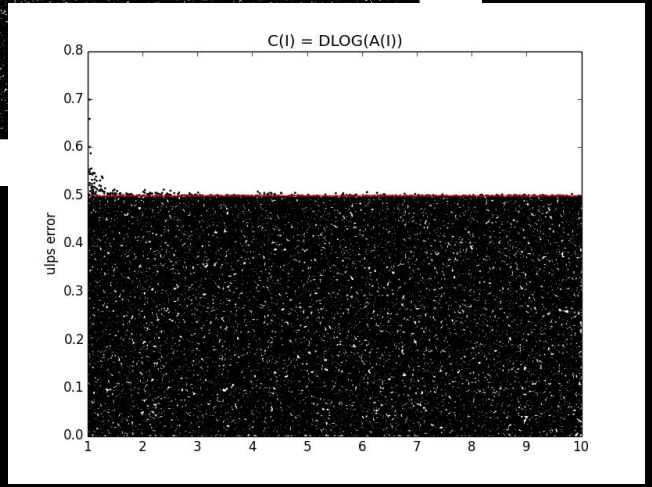
Tesla

PGI®

Double Precision log()

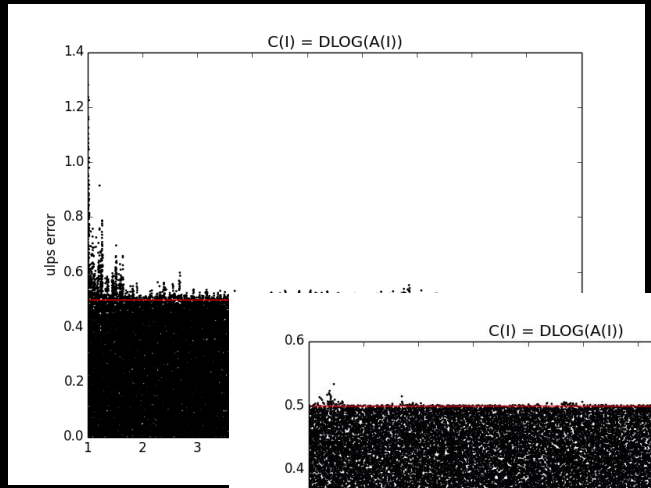
Bulldozer

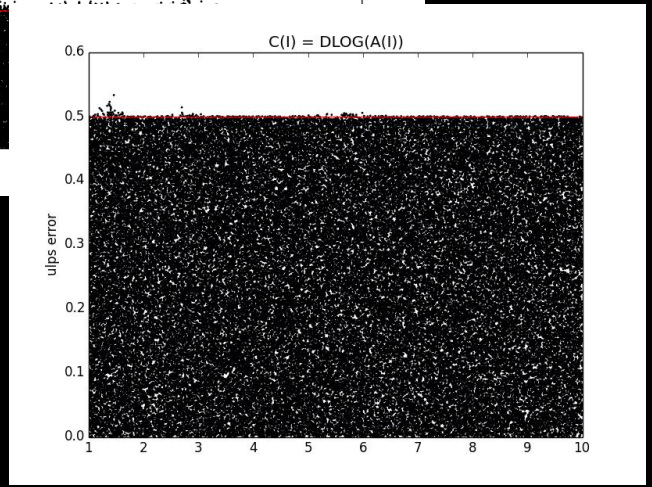




Haswell

POWER8



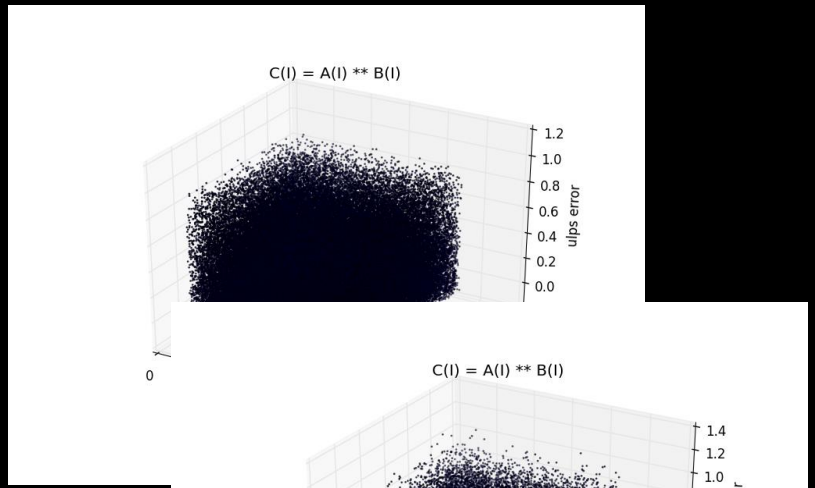


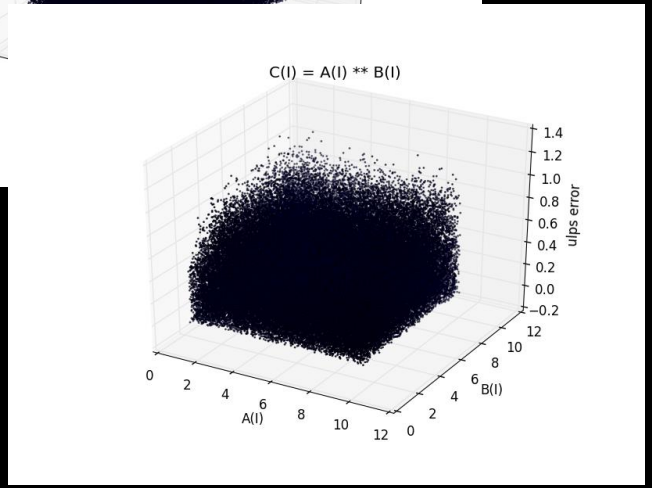
Tesla

PGI®

Double Precision pow()

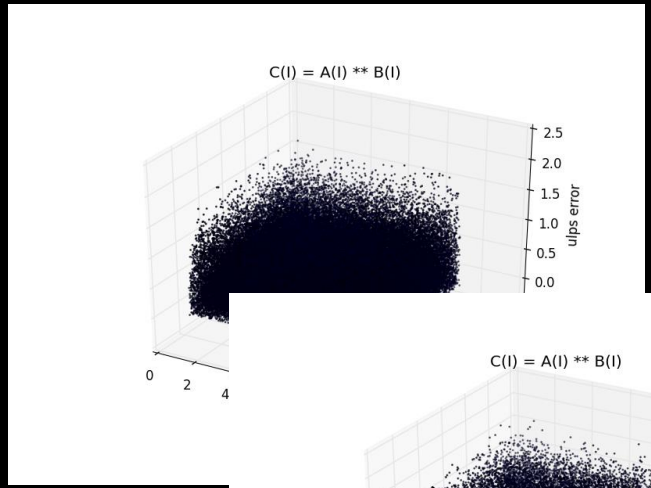
Bulldozer

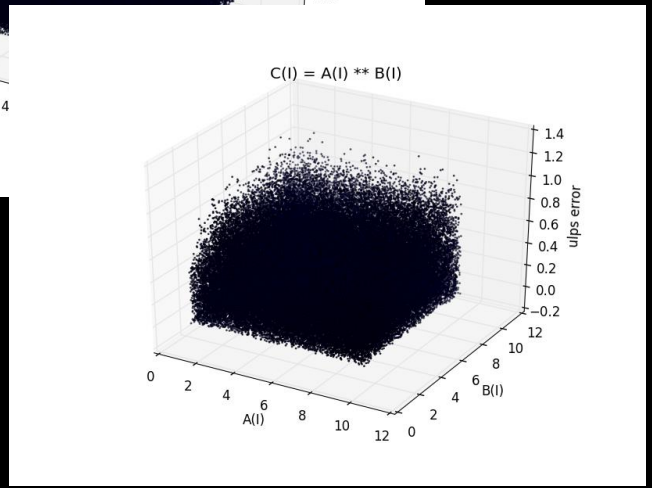




Haswell 17.1

POWER8





Tesla

PGI®

PGI Intrinsic Function Summary

Maximum and Average Ulp Error for the Input Range 1 - 10

	Bulldozer			Power8			Haswell			Tesla		
	Max Err	Ave Err	cycles/val	Max Err	Ave Err	cycles/val	Max Err	Ave Err	cycles/val	Max Err	Ave Err	cycles/val
sp exp	0.507	0.250	41	1.46	0.334	17	0.834	0.258	7	1.90	0.383	xxx
sp log	1.19	0.259	24	2.08	0.360	18	0.857	0.257	9	0.901	0.259	xxx
sp pow	0.508	0.250	143	32.6	3.03	31	0.849	0.256	19	2.30	0.449	xxx
dp exp	0.981	0.332	71	0.668	0.251	31	0.981	0.332	67	0.835	0.257	xxx
dp log	0.699	0.250	61	1.28	0.251	32	0.699	0.250	40	0.533	0.250	xxx
dp pow	1.01	0.333	157	2.07	0.473	59	1.01	0.333	145	1.22	0.339	xxx

Solution:

- Determine if you can live with these errors
- Replace single precision with double precision
- Use the PGI –Kieee compiler option

PGI®

SSE/AVX Intrinsics, Extended asm

- SSE and AVX intrinsics are usually used for performance reasons

```
__m256d __md1, __md2;
 double a[2], c[2];
__md1 = _mm256_set_pd(a[0], a[1], c[0], c[1]);
__md2 = _mm256_sqrt_pd(__md1);
```

- Extended or Inlined asm is often used to access hardware registers or functionality not available from high-level languages

```
unsigned long lo, hi;
asm( "rdtsc" : "=a" (lo), "=d" (hi) );
return( lo | (hi << 32) );
```

SSE/AVX Intrinsics, Extended asm

Solutions:

- The code in question may need to be rewritten and generalized.
- There may be equivalent functionality available on OpenPOWER

```
long t;
__asm__ __volatile__ ("mftb %0" : "=r" (t));
```

PGI®

PGI OpenMP Environment Differences

OpenMP API return values for Compiler/Runtime Options

	Haswell X86			OpenPOWER		
C/R Options OpenMP Call	**pgf90 o.f90**	**pgf90 -mp o.f90**	**OMP_NUM_THREADS =4**	**pgf90 o.f90**	**pgf90 -mp o.f90**	**OMP_NUM_THREADS =4**
omp_get_num_threads()	1	1	1	1	1	1
omp_get_max_threads()	1	1	4	1	160	4
omp_get_num_procs()	64	64	64	1	160	160
omp_get_thread_limit()	64	64	64	1	21478483647	2147483647
omp_in_parallel(), omp_get_thread_num(), omp_get_num_threads()	N, xxx, xxx	N, xxx, xxx	Y, 0-3, 4	N, xxx, xxx	Y, 0-159, 160	Y, 0-3, 4

PGI®

PGI OpenMP Environment Variables

Haswell X86

- `MP_BIND=yes`
- `MP_BLIST=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31`

OpenPOWER

- `OMP_PROC_BIND=true`
- `OMP_PLACES={0},{8},{16},{24},{32},{40},{48},{56},{64},{72},{80},{88},{96},{104},{112},{120},{128},{136},{144},{152}`

PGI OpenMP Environment Differences

Solutions/Recommendations:

- Always set OMP_NUM_THREADS
- Don't base data structures/control on omp_get_thread_limit, or at least limit it using the OMP_THREAD_LIMIT environment variable
- Setting OMP_PROC_BIND seems to improve performance
- Experiment with the hyper-threading level

PGI®

PGI Unformatted I/O Compatibility

- PGI Fortran unformatted sequential i/o files are consistent between X86 and OpenPower (and with Intel ifort)
- PGI Fortran unformatted sequential i/o files are consistent with xlf as long as the record byte counts are 32-bit integers
- PGI does not support record byte counts that are 64-bit integers. We split them up (same as ifort). Xlf will use them when the XLFRTEOPTS environment variable has uwidth=64

Options That are Dropped/Ignored/Different

- -fastsse, -Mvect=[no]sse, -Mvect=[no]altcode, . . .
- -fPIC, -KPIC, -kpic
- -Ktrap
- -tp=[bulldozer|sandybridge|…]
- -M[no]prefetch[=distance|n|…]
- -M[no]movnt
- -Minstrument
- -M[no]stack_arrays
- -mcmodel=small|medium|large
- Options related to debugging, dwarf, trace-back, stack

PGI®

PGI on OpenPOWER Current Limitations

Version 16.10

- No support for parsing of OpenMP 4.0/4.5 pragmas/directives.
- -Mipa is not enabled (no PGI inter-procedural analysis or optimization); the command-line option is accepted and silently ignored.
- -Mpfi/-Mpfo are not enabled (no profile-feedback optimization); the command-line options are accepted and silently ignored.
- No debugging support for Fortran.

PGI®

PGI on OpenPOWER Work in Progress

- Performance improvements
- Support for OpenMP 4.0/4.5 pragmas/directives
- Enabling -Mipa and all sub-options
- Debugging support for Fortran

PGI®