OLCF Hackathon: Appendix

GCC5 Nvidia GPU offloading: Initial experiences and benchmarks

By Philip LoCascio

16th December 2015

Outline

- GCC5
  - Compilation, installation and testing
- C program, with function
  - Calculate pi (2,000,000,000 summations)
  - Openacc and multiple precisions
  - OpenMP
- Fortran90
  - Openacc and multiple precisions just for comparison.
    - No noticable difference
- Results
  - Comparison of vector lengths for differing GPU architectures
  - Comparison of OpenMP performance
- Summary

GCC5

- Compilation
  - Automated from existing online resources (5.2.1)
    - auto-gcc5-offload-openacc-build-install.sh
      - Check github for tar bundle.
      - Patch needed for non SM_30 architecture
- |

```
--- a/src/gcc/libgomp/plugin/plugin-nvptx.c
+++ b/src/gcc/libgomp/plugin/plugin-nvptx.c
@@ -833,7 +833,7 @@
   optvals[5] = (void *) 1;

   opts[6] = CU_JIT_TARGET;
-  optvals[6] = (void *) CU_TARGET_COMPUTE_50;
+  optvals[6] = (void *) CU_TARGET_COMPUTE_30;
```

  - Possibly need further configure option
    - src/nvptx-tools/nvptx-as.c is wrapper around “ptxas”, so can take “-arch=” options.
    - Request GCC to add flag to configure as target, or -march= for compile?

GCC5 (cont)

- All in user specified directory
  - A wrapper script is generated after build:
    - rungcc5.sh <command> <args>
  -

```
[GCC5 offload wrapper] rungcc5: <gcc command> <args....>
Following gcc commands in the path:
c++ cpp g++ gcc gcov gfortran ;
Some examples of compilation:
a) using offload via openacc    -> rungcc5 gcc test-pi.c -fopenacc -foffload=nvptx-none -foffload=-O3 -O3 -o gpu.x
b) not using offload            -> rungcc5 gcc -O3 test-pi.c -o cpu.x
```

A simple C program:

calculate pi, using double precision

```
#include <stdio.h>

#define N 2000000000
int main(void)
{

    double pi = 0.0;
    for (long long ii = 0.0; ii < N; ii = ii + 1) {
        double t = (ii + 0.5) / N;
        double s = 4.0 / (1.0 + t * t);
        pi = pi + s;
    }
    printf("pi=%11.10f\n", pi / N);
    return 0;
}
```

Philip LoCascio

Openacc C program:

calculate pi, double precision, using openacc, within function

- Function isolated to help profiling
- 2 directives
  - parallel vector_length(len)
    - indicate how many parallel operations
  - loop reduction(var)
    - Indicate variable (+:pi) is the dependency for a reduction operation.
  - end parallel is superfluous in C

```
#include <stdio.h>

#define N 2000000000

#define vl 1024
double calcpi(long long n);

int main(void)
{

    double pi = 0.0f;
    pi = calcpi(N);

    printf("pi=%11.10f\n", pi / N);
    return 0;

}

double calcpi(long long n)
{
    double pi = 0.0f;
    double nf = 1.0 / (double) n;
#pragma acc parallel vector_length(vl)
#pragma acc loop reduction(+:pi)
    for (long long ii = 0.0; ii < n; ii = ii + 1) {
        double t = (double) ((ii + 0.5) * nf);
        pi += 4.0 / (1.0 + t * t);
    }
#pragma acc end parallel
    return pi;
}                                   //end calcpi
```

rungcc5 gcc testpi.c fopenacc foffload=nvptxnone O3 foffload=”O3” o testpi.bin

OpenAcc C program:

calculate pi, mixed precision, using openacc, within function

- Switch a few operations to single precision, to investigate effect on performance. The GPU's under test have a great deal of precision/performance difference.
- GTX980 is SP=1/32 DP
  - Target SM_5X
- TITAN BLACK SP=1/3 DP
  - Target SM_30
  - Equivalent to Kepler on OLCF Titan

```
#include <stdio.h>

#define N 2000000000
/* Maximum parallelism allowed by Nvidia*/
#define vl 1024
double calcpi(long long n);

int main(void)
{

  double pi = 0.0;
  pi = calcpi(N);

  printf("pi=%11.10f\n", pi / N);
  return 0;

}

double calcpi(long long n)
{
  double pi = 0.0;
#pragma acc parallel vector_length(vl)
#pragma acc loop reduction(+:pi)
  for (long long ii = 0.0; ii < n; ii = ii + 1) {
    float t = (ii + 0.5f) / n;
    float s = 4.0f / (1.0f + t * t);
    pi = pi + s;
  }
  return pi;
}                                    //end calcpi
```

OpenAcc F90 program:

calculate pi, mixed precision, using openacc, within function

- Fortran 90 is freeform
-
- Notice the increased memory management
  - COPYIN, COPYOUT
- Incorrect results occurred without data management or any warnings!
- A conversation with Michael Wolfe of NVIDIA (referencing his OLCF talk) suggests that this in the OPENACC standard. The PGI compiler, however, generates these COPYIN/COPYOUT statements and puts them in the error output.
- FORTRAN also requires the closing END PARALLEL statement.

```
      PROGRAM TESTPI

      IMPLICIT NONE

      INTERFACE
         FUNCTION CALCPI (N)
         INTEGER, PARAMETER :: DP = KIND(1.0D0)
         INTEGER*8 , INTENT(IN) :: N
         REAL(DP) ::  CALCPI
         END FUNCTION CALCPI
      END INTERFACE

      INTEGER*8 ,PARAMETER :: N=2000000000
      INTEGER, PARAMETER :: DP = KIND(1.0D0)

      REAL(DP) OPI

      OPI=CALCPI(N)

      PRINT *,'PI=',(OPI/N)


      END PROGRAM TESTPI


      FUNCTION CALCPI( N)

      IMPLICIT NONE

      INTEGER, PARAMETER ::  VL=1024
      INTEGER, PARAMETER :: DP = KIND(1.0D0)
      INTEGER, PARAMETER :: SP = KIND(1.0)
      REAL(DP):: CALCPI
      REAL(DP) PI
      INTEGER*8 I
      REAL(SP) T,II
      INTEGER*8 , INTENT(IN) :: N

      PI=0.D0
      II=0
      T=0.0
!$ACC PARALLEL VECTOR_LENGTH(VL) COPYOUT(PI) COPYIN(N)
!$ACC LOOP REDUCTION(+:PI)
      DO  I=0,N
         II=REAL(I)
         T= ((II+0.5)/N)
         PI = PI+4.0/(1.0+T*T)
      ENDDO
!$ACC END PARALLEL
      CALCPI=PI

      END FUNCTION CALCPI
```

OpenAcc F90 program:

calculate pi, double precision, using openacc, within function

- Notice change in precision: REAL(I,DP)
- Scope of the declarations appears important

```
      PROGRAM TESTPI

      IMPLICIT NONE

      INTERFACE
         FUNCTION CALCPI (N)
         INTEGER, PARAMETER :: DP = KIND(1.0D0)
         INTEGER*8 , INTENT(IN) :: N
         REAL(DP) ::  CALCPI
         END FUNCTION CALCPI
      END INTERFACE

      INTEGER*8 ,PARAMETER :: N=2000000000
      INTEGER, PARAMETER :: DP = KIND(1.0D0)

      REAL(DP) OPI,NDP
      NDP=REAL(N,DP)

      OPI=CALCPI(N)

      PRINT *,'PI=',(OPI/NDP),'OPI=',OPI,'NDP=',NDP


      END PROGRAM TESTPI


      FUNCTION CALCPI( N)

      IMPLICIT NONE

      INTEGER, PARAMETER ::  VL=1024
      INTEGER, PARAMETER :: DP = KIND(1.0D0)
      INTEGER, PARAMETER :: SP = KIND(1.0)
      REAL(DP):: CALCPI
      REAL(DP) PI
      INTEGER*8 I
      REAL(DP) T,II,NDP
      INTEGER*8 , INTENT(IN) :: N

!$ACC PARALLEL VECTOR_LENGTH(VL) COPYOUT(PI) COPYIN(N,NDP,T,II)
      NDP=REAL(N,DP)
      T=0.D0
      II=0.D0
      PI=0.D0
!$ACC LOOP REDUCTION(+:PI)
      DO  I=0,N
         II=REAL(I,DP)
         T= ((II+0.D5)/NDP)
         PI = PI+4.D0/(1.D0+(T*T))
      ENDDO
!$ACC END PARALLEL
      CALCPI=PI

      END FUNCTION CALCPI
```

OpenMP C program:

calculate pi, double precision, using OpenMP, within function

- OpenMP code compiled using same options as OpenACC
- Equivalent statements and underneath supported by libgomp, which implements the same code
- However this runs only in Multi-threaded mode on the host, for comparison.

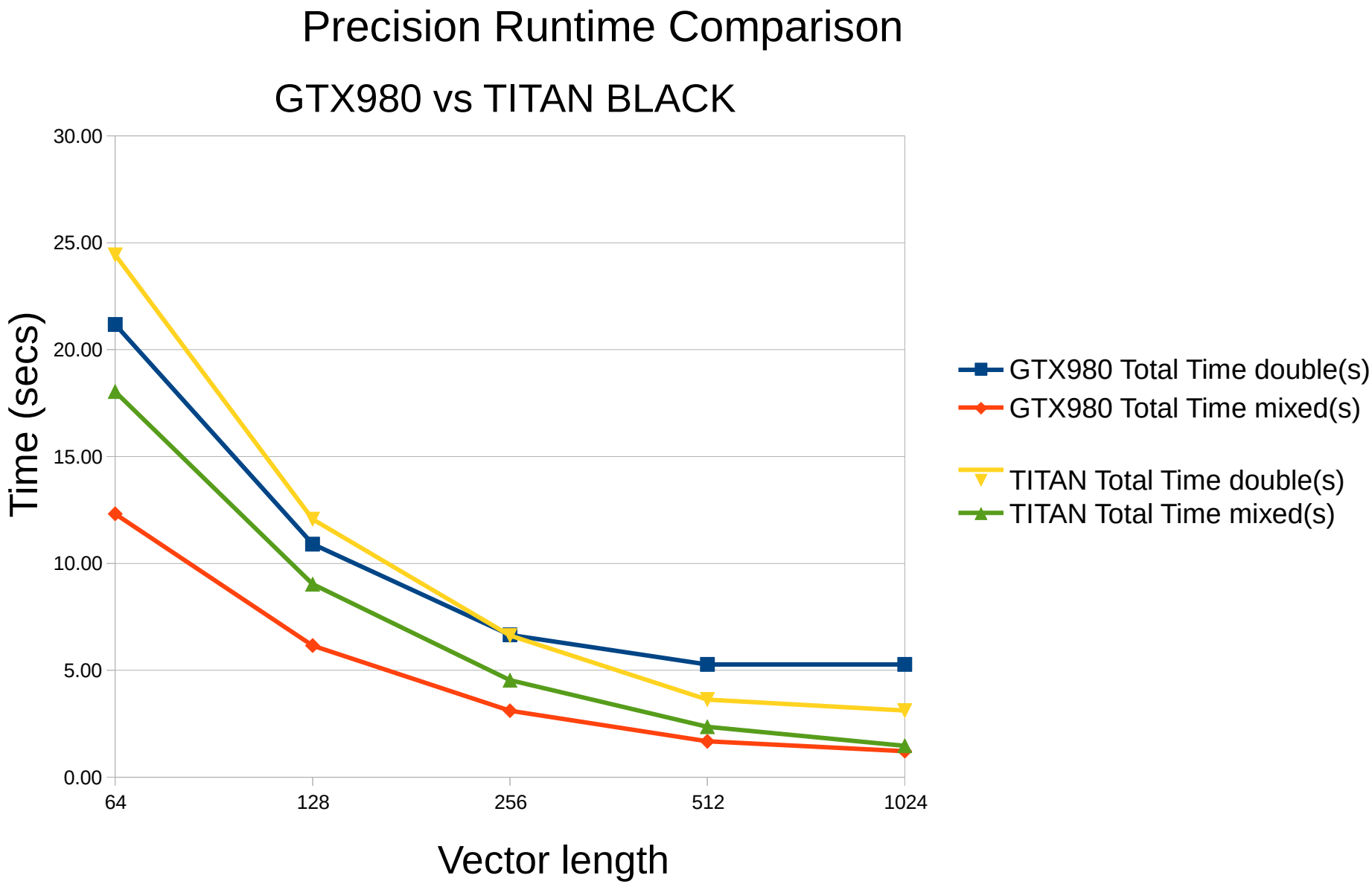
```
#include <stdio.h>
#include <omp.h>


#define N 2000000000

#define vl 1024
double calcpi(long long n);

int main(void) {

  double start, end;
  double pi = 0.0f;
  start=omp_get_wtime();
  pi=calcpi(N);
  end=omp_get_wtime();

  double delta = end-start;
  printf("pi=%11.10f  time=%11.6f secs\n",pi/N,delta);
  return 0;

}

double calcpi(long long n){
  double pi = 0.0f;
  double nf=1.0/(double)n;
  #pragma omp parallel for   reduction(+:pi)
  for ( long long  ii=0.0; ii<n; ii=ii+1) {
    double t= (double)((ii+0.5)*nf);
    pi+=4.0/(1.0+t*t);
  }
        return pi;
}//end calcpi
```

Methods

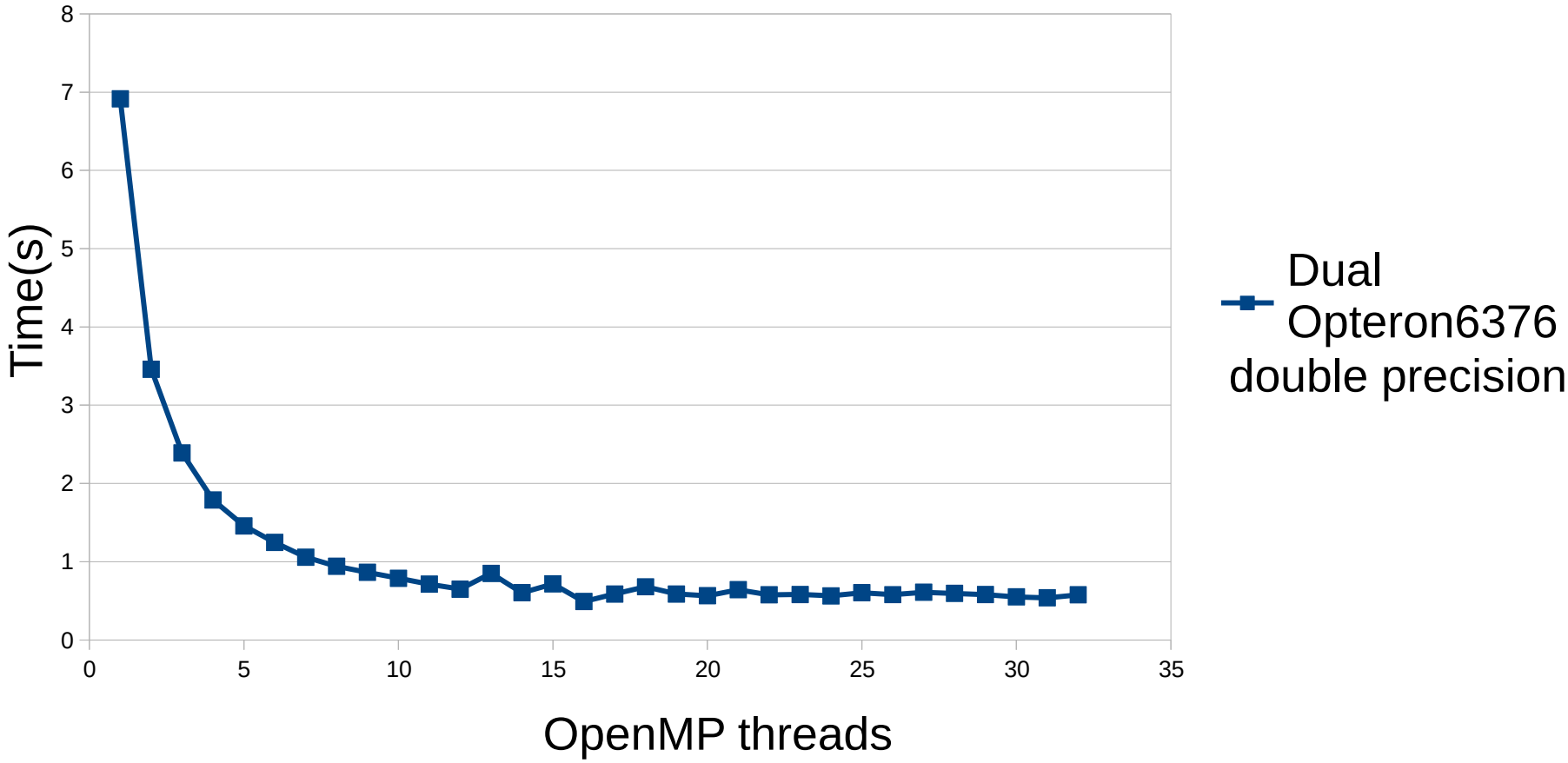
- Two systems used
  - AMD Dual Opteron 6376 (2.3Ghz) with Dual GTX980
    - CUDA 6.5 , linux 4.1.1, gcc5 built with (Debian 4.9.1-19)
    - One dedicated GTX980.
  - AMD FX 8120 Eight-Core with Titan Black
    - CUDA 6.5, linux 3.16-0.4, gcc5 built with (Debian 4.9.1-19)

Times adjusted for
cuCtxCreate = ~300ms

Results



Results
OpenMP scaling



Tables of performance data

- GPU GTX980 vs TITAN BLACK

Threads	GTXTotal Time double(s)	GTX Total Time mixed(s)	TITAN Total Time double(s)	TITAN Total Time mixed(s)
64	21.18	12.32	24.44	18.04
128	10.90	6.16	12.07	9.03
256	6.66	3.11	6.62	4.54
512	5.27	1.68	3.64	2.36
1024	5.27	1.22	3.11	1.47

- OpenMP

OpenMP Threads	Dual Opteron6376
1	6.91
2	3.46
4	2.39
8	1.79
16	1.46
32	1.25

Summary

- GCC 5.2.1 was tested and Openacc offloading confirmed to work correctly
- Auto build script constructed
- 2 types of GPUs tested SM_30 and SM_50
  - Direct effect on perfomance due to precision
  - Efficiency against CPU and OpenMP suggests further analysis needed.

- Request GCC/Nvidia developers add some target arch ability
  - Either in compile or at build.

- Further tests to compile the HPL benchmark by Pathscale was not successful.
  - https://github.com/pathscale/hpl-2.0-openacc
- Failed in link stage (SEGV!!); May need to rebuild MPI using new compiler as final linking used mpif77 to produce binary
  - OpenACC compilation appear successful

References

- https://gcc.gnu.org/wiki/Offloading#How_to_try_offloading_enabled_GCC
- http://scelementary.com/2015/04/25/openacc-in-gcc.html
- http://mirrors.concertpass.com/gcc/snapshots/
- https://github.com/olcf/OLCFHack15