SGEMM (document version 1.1)

Introduction

Matrix multiplication is a standard problem in HPC. This computation is exemplified in the Basic Linear Algebra Subroutine (BLAS) function SGEMM. Many libraries contain highly optimized code to execute this problem. In this exercise we define 3 matrices A, B and C of dimension N x N. All elements of matrix A are equal to 1, and all values in B are set to 2. The resulting matrix C should therefore consist of elements equal to 1*2*N. Note: to stay in line with many standard solutions for matrix problems the programs use one dimensional arrays instead of more natural 2-dimensional variations.

Difficulty: Simple to Medium

ENVIRONMENT SETUP

In this example the behavior of the serial version of a matrix multiplication is reviewed

1) Our Xeon Phi board is available on the machine titan.dii.unisi.it . To log into it you will be assigned authentication credentials: login=userXXX, pw=PPPPPPPP (XXX and PPPPPPPP will be given to you during the lab session). Then issue:

   ssh userXXX@titan.dii.unisi.it

   (when prompted specify the password PPPPPPPP and eventually answer yes to accept the host id).

2) To download and use the lab material issue:

   wget –i lab-mic1.link (you should get the file “many-faces-of-parallelism-lab.zip”

   unzip many-faces-of-parallelism-lab.zip

TEST01: Serial Version

In this example the behavior of the serial version of a matrix multiplication is reviewed

1) Change the working directory: cd ~/LAB/02_sgemm

2) Open up an editor and examine sgemm_simple.cpp

3) Compile sgemm_simple: icpc -o sgemm_simple sgemm_simple.cpp

4) Run sgemm_simple without options: ./sgemm_simple

5) Time a run of sgemm_simple with 1000 steps (3 zeroes): time ./sgemm_simple 1000 Write down real, user and sys time.

TEST02: Intel Cilk Plus

In this example a simple program used to compute a matrix multiplication is parallelized using Intel Cilk Plus.

1) Change the working directory: cd ~/LAB/02_sgemm
2) Copy sgemm_simple.cpp to sgemm_cilk.cpp: cp sgemm_simple.cpp sgemm_cilk.cpp
3) Copy sgemm_simple.h to sgemm_cilk.h: cp sgemm_simple.h sgemm_cilk.h

4) Add header definitions to sgemm_cilk.h: #include <cilk/cilk.h>

5) Edit sgemm_cilk.cpp:

a. Fix the include to parse “sgemm_cilk.h“

b. In the compute() function replace the outermost for with a cilk_for loop: cilk_for(int i = 0; i < N; i++)

6) Note: completed files are available as sgemm_cilk_sol.cpp and sgemm_cilk_sol.h

7) Compile sgemm_cilk: icpc -o sgemm_cilk sgemm_cilk.cpp

8) Run both serial version and parallel version with a count of 10 and compare the output. Ensure the line “Standard Element: 20” appears in both runs: ./sgemm_simple 10 ./sgemm_cilk 10

9) Run sgemm_cilk with different sizes 100 to 3000.

10) Time a run of sgemm_cilk with 1000 steps (3 zeroes): time ./sgemm_cilk 1000 Write down real, user and sys time.

11) Compare both runs – in the Intel Cilk Plus enabled version the program runs in parallel and uses much more user than real time.

TEST03: Intel Cilk Plus on Intel Xeon Phi Coprocessor

In this example we will offload the Intel Cilk Plus parallelized version of SGEMM to the Intel Xeon Phi coprocessor.

1) Change the working directory: cd ~/LAB/02_sgemm

2) Copy sgemm_cilk_sol.cpp to sgemm_cilk_mic.cpp: cp sgemm_cilk_sol.cpp sgemm_cilk_mic.cpp

3) Copy sgemm_cilk_sol.h to sgemm_cilk_mic.h: cp sgemm_cilk_sol.h sgemm_cilk_mic.h

4) Edit sgemm_cilk_mic.h:

a. The offloaded part will include Intel Cilk Plus commands. As those are defined in an external header file, the compiler needs to be made aware of them. Enclose these includes by a #pragma offload: #pragma offload_attribute(push,target(mic)) #include <cilk/cilk.h> #pragma offload_attribute(pop)

5) Edit sgemm_cilk_mic.cpp:

a. Fix the include to parse “sgemm_cilk_mic.h“

b. In function compute() the input parameter Count specifies the length of one dimension of the matrices. The length of the matrices must be the square of that value. Thus, add a declaration and assignment of Size: int Size = N*N;

c. In the function compute() create a {..} block to enclose the complete calculation. This block will be offloaded and starts right before the cilk_for reaching down until before the return(0.) statement. To offload this block add a #pragma offload clause in front of the opening {. This clause must contain all variables that are transferred between host and the Intel Xeon Phi coprocessor. The direction of transfer is in, out or inout. In case of an array you also have to specify the length (note: length is determined at runtime). The complete #pragma offload then takes the shape of: #pragma offload target(mic:0) in(A:length(Size)) in(B:length(Size)) inout(C:length(Size)) in(N)

6) Note: completed files are available as sgemm_cilk_mic_sol.cpp and sgemm_cilk_mic_sol.h

7) Compile using the –offload-build option: icpc -offload-build -o sgemm_cilk_mic sgemm_cilk_mic.cpp

8) Run the program with: time ./sgemm_cilk_mic 1000 Notice that user time is less than real time (only a single CPU is used on the host).

TEST04: Intel MKL

As Intel MKL contains special functions to compute the SGEMM problem we start with a complete source file.

1) Change the working directory: `cd ~/LAB/02_sgemm`

2) View the *sgemm_mkl_sol.cpp* source file and note the MKL sgemm() call

3) Compile the source: `icpc -openmp -mkl -o sgemm_mkl sgemm_mkl_sol.cpp`

4) Ensure the environment variable `OMP_NUM_THREADS` matches the number of cores (to determine this, look at the output of cat /proc/cpuinfo and then set the variable: `export OMP_NUM_THREADS=8`; and `MKL_MIC_ENABLE` is NOT set: `unset MKL_MIC_ENABLE printenv OMP_NUM_THREADS`

If the OMP_NUM_THREAD is not set then set it to match the number of cores. You can find the number of cores on your host by looking the output from `cat /proc/cpuinfo`:

```
export OMP_NUM_THREADS=8;
```

```
4b) source /opt/intel/composerxe/bin/compilervars.sh   intel64
```

5) Execute the program with different sizes 100 to 10000: `./sgemm_mkl 1000`

6) Use time to ensure all CPUs are used (user >> real): `time ./sgemm_mkl 10000`

7) Enforce program is running on a single CPU: `export OMP_NUM_THREADS=1`

8) Time a run of sgemm_mkl with N=5000: `time ./sgemm_mkl 5000` Write down real, user and sys time..

9) Enable Intel Xeon Phi Coprocessor support by setting the environment variable `MKL_MIC_ENABLE` to 1: `export MKL_MIC_ENABLE=1`

10) Time a run of sgemm_mkl with N=5000: `time ./sgemm_mkl 5000` Write down real, user and sys time..

11) Result: Intel MKL can make use of an Intel Xeon Phi coprocessor without changing code! This technique is called Automated Offloading.

TEST05: OpenMP on Intel Xeon Phi Coprocessor

In this example we will offload an already parallelized version of SGEMM to the Intel Xeon Phi coprocessor.

1) Change the working directory: `cd ~/LAB/02_sgemm`

2) Copy *sgemm_openmp_sol.cpp* to *sgemm_openmp_mic.cpp*: `cp sgemm_openmp_sol.cpp sgemm_openmp_mic.cpp`

3) Copy *sgemm_openmp_sol.h* to *sgemm_openmp_mic.h*: `cp sgemm_openmp_sol.h sgemm_openmp_mic.h`

4) Edit *sgemm_openmp_mic.cpp*:

a. Fix the include to parse "`sgemm_openmp_mic.h`"

b. For your convenience a `{}` block enclosing the complete calculation is already present in the source code. Offload the block by adding a `#pragma offload` in front. This `#pragma` must contain all variables that are transferred between host and the Intel Xeon Phi coprocessor. The direction of transfer is `in`, `out` or `inout`. In case of an array you also have to specify the length (note: length is determined at runtime): `#pragma offload target(mic:0) in(A:length(Size)) in(B:length(Size)) inout(C:length(Size)) in(N)`

5) Note: completed files are available as *sgemm_openmp_mic_sol.cpp* and *sgemm_openmp_mic_sol.h*

6) Compile: `icpc -openmp -o sgemm_openmp_mic sgemm_openmp_mic.cpp`

7) Set the max thread value:

```
export MIC_ENV_PREFIX=MIC
export MIC_OMP_NUM_THREADS=120
```

8) Run the program with: `time ./sgemm_openmp_mic 1000` Notice that user time is less than real time (only a single CPU is used on the host).

TEST06: Running a MKL version natively on the Intel Xeon Phi Coprocessor

An Intel MKL version of the code can be found in the file *sgemm_mkl_sol.cpp.*

1) Change the working directory: `cd ~/LAB/02_sgemm`

2) View *sgemm _mkl_sol.cpp* source

3) Compile using the –mmic option: `icpc -mmic -openmp -mkl -o sgemm_mkl_mic_native sgemm_mkl_sol.cpp`

4) Copy the binary to the Intel® Xeon Phi™ coprocessor: `scp sgemm_mkl_mic_native mic0:/tmp`

5) Run the program on the Intel® Xeon Phi™ coprocessor using ssh: `ssh mic0 '/tmp/sgemm_mkl_mic_native 1000'`

6) Notice the error complaining about a missing library.

7) Copy the missing library onto the Intel® Xeon Phi™ coprocessor using scp:

```
scp $MKLROOT/../compiler/lib/mic/libiomp5.so mic0:
scp $MKLROOT/lib/mic/libmkl_intel_lp64.so mic0:
scp $MKLROOT/lib/mic/libmkl_intel_thread.so mic0:
scp $MKLROOT/lib/mic/libmkl_core.so mic0:
```

8) Rerun the program, setting the `LD_LIBRARY_PATH` variable to ~/: `ssh mic0 'export LD_LIBRARY_PATH=~/;~/sgemm_mkl_mic_native \ 1000'`

TEST07: Running a Intel Cilk Plus version natively on the Intel Xeon Phi Coprocessor

An Intel Cilk Plus version of the code can be found in the file *sgemm_cilk_sol.cpp.*

1) Change the working directory: `cd ~/LAB/02_sgemm`

2) View *sgemm _cilk_sol.cpp* source

3) Compile using the –mmic option: `icpc -mmic -o sgemm_cilk_mic_native sgemm_cilk_sol.cpp`

4) Copy the binary to the Intel® Xeon Phi™ coprocessor: `scp sgemm_cilk_mic_native mic0:`

5) Run the program on the Intel® Xeon Phi™ coprocessor using the ssh: `ssh mic0 '~/sgemm_cilk_mic_native 1000'`

6) Notice the error complaining about a missing library.

15) Copy the missing library onto the Intel® Xeon Phi™ coprocessor using scp: `scp $MKLROOT/../compiler/lib/mic/libcilkrts.so.5 mic0:~/`

7) Rerun the program, setting the `LD_LIBRARY_PATH` variable to ~/: `ssh mic0 'export LD_LIBRARY_PATH=~/;~/sgemm_cilk_mic_native \ 1000'`

TEST08: Running an OpenMP* version natively on the Intel Xeon Phi Coprocessor

An OpenMP version of the code can be found in the file *sgemm_openmp_sol.cpp*.

1) Change the working directory: `cd ~/LAB/02_sgemm`

2) View *sgemm_openmp_sol.cpp* source

3) Compile using the –mmic option: `icpc -mmic -openmp -o sgemm_openmp_mic_native sgemm_openmp_sol.cpp`

4) Copy the binary to the Intel® Xeon Phi™ coprocessor: `scp sgemm_openmp_mic_native mic0:~/`

5) Run the program on the Intel® Xeon Phi™ coprocessor using the ssh: `ssh mic0 '~/sgemm_cilk_mic_native 1000'`

6) Notice the error complaining about a missing library.

7) Copy the missing library onto the Intel® Xeon Phi™ coprocessor using scp: `scp $MKLROOT/../compiler/lib/mic/libiomp5.so mic0:~/`

8) Rerun the program, setting the `LD_LIBRARY_PATH` variable to ~/: `ssh mic0 'export LD_LIBRARY_PATH=~/;export OMP_NUM_THREADS=60; \ ~/sgemm_openmp_mic_native 1000'`