MAE 5032 High Performance Computing: Methods and Applications

Lab 5: Job control on TaiYi

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

Objective

- You will perform three tasks
  - submit a simple jobscript on TaiYi
  - experience the build process with optimization flags on TaiYi
  - experience the use of MKL library on TaiYi

Task 1: Jobscript

- Log into TaiYi with your own account

- Run the following commands

```
mmlsquota -g user-name -block-size auto

bqueues -l

bqueues -l debug

bhosts hg_debug

bhosts -l r01n01
```

Task 1: Jobscript

- Log into TaiYi with your own account

- Run the following commands

```
lsload
```

```
lsload r01n01
```

Task 1: Jobscript

- Prepare a jobscript as follows

- What are the expected outcome?

- Submit it by

```
bsub < jobscript
```

```
bsub -J newname < jobscript
```

What are the differences?

```
#!/bin/bash

#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -e log
#BSUB -o log
#BSUB -R "span[ptile=1]"

echo $LSB_JOBID
echo $LSB_JOBNAME
echo $LSB_QUEUE
echo $LS_SUBCWD
echo $LSB_HOSTS
echo $LSB_MCPU_HOSTS
echo $LSB_DJOB_HOSTFILE
echo $LSB_DJOB_NUMPROC
```

Task 2: Compile with optimization

- Obtain the source file simple.c

```
module load intel/2018.4
```

```
icc simple.c -O3 -qopt-report=2 -qopt-report-phase=vec -o simple2
```

```
cat simple.optrpt
```

```c
#include <stdlib.h>
#include <stdio.h>

#define ARRAY_SIZE 1024
#define NUMBER_OF_TRIALS 1000000

/*
 * Statically allocate our arrays.  Compilers can
 * align them correctly.
 */
static double a[ARRAY_SIZE], b[ARRAY_SIZE], c;

int main(int argc, char *argv[])
{
  int i,t;

  double m = 1.0001;

  /* Populate A and B arrays */
  for (i=0; i < ARRAY_SIZE; i++)
  {
    b[i] = i;
    a[i] = i+1;
  }

  /* Perform an operation a number of times */
  for (t=0; t < NUMBER_OF_TRIALS; t++)
  {
    for (i=0; i < ARRAY_SIZE; i++)
    {
      c += m*a[i] + b[i];
    }
  }

  return 0;
}
```

Task 2: Compile with optimization

- Obtain the source file simple.c

```
module load intel/2018.4
```

```
icc simple.c -O3 -no-vec -qopt-report=2
-qopt-report-phase=vec -o simple2_no_vec
```

```
cat simple.optrpt
```

```c
#include <stdlib.h>
#include <stdio.h>

#define ARRAY_SIZE 1024
#define NUMBER_OF_TRIALS 1000000

/*
 * Statically allocate our arrays.  Compilers can
 * align them correctly.
 */
static double a[ARRAY_SIZE], b[ARRAY_SIZE], c;

int main(int argc, char *argv[])
{
  int i,t;

  double m = 1.0001;

  /* Populate A and B arrays */
  for (i=0; i < ARRAY_SIZE; i++)
  {
    b[i] = i;
    a[i] = i+1;
  }

  /* Perform an operation a number of times */
  for (t=0; t < NUMBER_OF_TRIALS; t++)
  {
    for (i=0; i < ARRAY_SIZE; i++)
    {
      c += m*a[i] + b[i];
    }
  }

  return 0;
}
```

Task 2: Compile with optimization

- Obtain the source file simple.c

```
module load intel/2018.4
```

```
icc simple.c -O3 -xSKYLAKE-AVX512 -qopt-report=2 -qopt-report-phase=vec -o simple2_avx512
```

```
cat simple.optrpt
```

```c
#include <stdlib.h>
#include <stdio.h>

#define ARRAY_SIZE 1024
#define NUMBER_OF_TRIALS 1000000

/*
 * Statically allocate our arrays.  Compilers can
 * align them correctly.
 */
static double a[ARRAY_SIZE], b[ARRAY_SIZE], c;

int main(int argc, char *argv[])
{
  int i,t;

  double m = 1.0001;

  /* Populate A and B arrays */
  for (i=0; i < ARRAY_SIZE; i++)
  {
    b[i] = i;
    a[i] = i+1;
  }

  /* Perform an operation a number of times */
  for (t=0; t < NUMBER_OF_TRIALS; t++)
  {
    for (i=0; i < ARRAY_SIZE; i++)
    {
      c += m*a[i] + b[i];
    }
  }

  return 0;
}
```

Task 2: Compile with optimization

- Obtain the source file simple.c

```
module load intel/2018.4
```

```
icc simple.c -O3 -xSKYLAKE-AVX512 -qopt-zmm-usage=high -qopt-report=2 -qopt-report-phase=vec -o simple2_zmm_hi
```

```
cat simple.optrpt
```

```c
#include <stdlib.h>
#include <stdio.h>

#define ARRAY_SIZE 1024
#define NUMBER_OF_TRIALS 1000000

/*
 * Statically allocate our arrays.  Compilers can
 * align them correctly.
 */
static double a[ARRAY_SIZE], b[ARRAY_SIZE], c;

int main(int argc, char *argv[])
{
  int i,t;

  double m = 1.0001;

  /* Populate A and B arrays */
  for (i=0; i < ARRAY_SIZE; i++)
  {
    b[i] = i;
    a[i] = i+1;
  }

  /* Perform an operation a number of times */
  for (t=0; t < NUMBER_OF_TRIALS; t++)
  {
    for (i=0; i < ARRAY_SIZE; i++)
    {
      c += m*a[i] + b[i];
    }
  }

  return 0;
}
```

Task 2: Compile with optimization

```
#!/bin/bash

#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -e log
#BSUB -o log
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'

# Non-vectorized code
/usr/bin/time -f "simple2_no_vec: %e" ./simple2_no_vec

# Standard SSE vectorized code
/usr/bin/time -f "simple2: %e" ./simple2

# AVX-512 vectorized code
/usr/bin/time -f "simple2_avx512: %e" ./simple2_avx512

# AVX-512 vectorized code, with tweaks
/usr/bin/time -f "simple2_zmm_hi: %e" ./simple2_zmm_hi
```

Task 3: Use MKL

Obtain the main.c code from the github repo

```
module load intel/2018.4

echo $MKLROOT

icc main.c -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -liomp5 -
lpthread -lm -ldl
```

Task 3: Use MKL

Prepare a jobscript and submit it to the supercomputer. You may take this as a reference.

```
#!/bin/bash
#BSUB -J mytest
#BSUB -q debug
#BSUB -n 1
#BSUB -W 00:05
#BSUB -e %J.err
#BSUB -o %J.out
#BSUB -R "span[ptile=1]"
#BSUB -m 'r13n45'

module purge
module load intel/2018.4
module load mpi/intel/2018.4

mpirun /work/mae-liuj/mae-5032/06-lsf/dgemm > $LSB_JOBID.log 2>&1
```