应用：HPL实验

> 姓名：梁鑫嵘；学号：200110619

实验内容

1. 进行HPL(The High-Performance Linpack Benchmark)性能测试
2. 完成实验报告

实验内容

硬件配置

1. CPU 硬件配置：`Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz`
2. 峰值性能计算：
   a. $FLOPS_{\text{cpu, double}} = N_{\text{cores}} \times Freq_{\text{per core}} \times N_{FMA} \times 2_{\text{plus and multiplication}} \times \frac{512_{\text{AVX 512}}}{64}$
   b. $FLOPS_{\text{5218, double}} = 16_{\text{cores}} \times 2.30_{\text{per core}} \times 1_{FMA} \times 2_{\text{plus and multiplication}} \times \frac{512_{\text{AVX 512}}}{64} = 588.8\ GFlops$

软件依赖

1. 运行环境：
   a. Vmmare 虚拟机环境，宿主机为 Windows 10 双 Intel Xeon 5218。
   b. OpenEuler 系统
2. 编译器：`gcc version 9.3.1 (GCC)`
3. MPI：`mpirun (Open MPI) 4.1.1`
4. BLAS库：`libopenblas_skylakexp-r0.3.17.dev.so`
5. HPL：http://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz

HPL 编译

1. 编译安装 OpenBLAS

```
git clone https://github.com/xianyi/OpenBLAS && cd OpenBLAS
make -j16
sudo make install
```

2. 编译安装 OpenMPI

```
wget https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.1.tar.gz
tar xzf openmpi-4.1.1.tar.gz
cd openmpi-4.1.1
./configure
make -j16
sudo make install
```

3. 编译 HPL

```
wget http://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz
tar xzf hpl-2.3.tar.gz
cd hpl-2.3
./configure
make -j16
cd testing
```

4. 准备运行环境

```
mkdir /share/
chmod 775 /share/
cd /share/
mkdir /share/hpc/
cd hpc
cp ~/hpl-2.3/testing xhpl .
```

HPL 运行

1. 准备运行脚本

```
import json
import traceback
import copy
import time
import os

template = '''HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
test.out     output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
{Ns}         Ns
1            # of NBs
{NBs}        NBs
1            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
{Ps}         Ps
{Qs}         Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
'''

results = {}


def update_dat_file(run_args: dict, template_: str = template, target: str = 'openblas', local: bool = True) -> bool:
    data = template_.format(**run_args)
    try:
        if local:
            with open('HPL.dat', 'w', encoding='utf8') as f:
                f.write(data)
        else:
            with open(os.path.join(os.path.join('build', target), 'HPL.dat'), 'w', encoding='utf8') as f:
                f.write(data)
    except Exception:
        traceback.print_exc()
        return False
    return True


def run(run_args: dict, target: str = 'openblas') -> float:
    update_dat_file(run_args=run_args, target=target, local=True)
    os.system("rm nohup.out")
    os.system(f"nohup mpirun -n 28 ./build/{target}/xhpl &")
    time.sleep(0.2)
    pid = int(os.popen(f'ps aux | grep "mpirun.*xhpl"').readline().split()[1])
    print('pid', pid)
    lines_count = 10
    unit, flops = None, None
    while True:
        if len(os.popen(f'ps aux | grep {pid}').readline()) == 0:
            break
        try:
            lines = os.popen(f"tail -n {lines_count} nohup.out").readlines()
```

```python
            if len(lines) == lines_count and lines[0].startswith('=' * 10) and lines[1].startswith("T/V") and len(lines[3].split(
                unit = lines[1].split()[-1]
                flops = float(lines[3].split()[-1])
                break
        except Exception:
            traceback.print_exc()
    # os.system(f"kill {pid}")
    os.system(f"killall mpirun")
    print(f"{flops} {unit} {run_args}")
    results[f"P{run_args['Ps']}_Q{run_args['Qs']}_N{run_args['Ns']}"] = flops
    with open("result.json", "w", encoding="utf8") as f:
        json.dump(results, f, indent=2, sort_keys=True)
    if unit == 'Tflops':
        flops *= 1024
    return flops


def generate_args(run_args_range: dict, cores: int = 16) -> dict:
    tails = {}
    now = {}
    for key in run_args_range:
        if not isinstance(run_args_range[key], list):
            continue
        tails[key] = 0
        now[key] = run_args_range[key][0]
    print("keys:", list(run_args_range.keys()))

    while True:
        for key in run_args_range:
            if not isinstance(run_args_range[key], list):
                continue
            now[key] = run_args_range[key][tails[key]]
        if now['Qs'] * now['Ps'] != cores:
            continue
        now_data = copy.deepcopy(now)
        yield now_data
        p = 0
        over: bool = False
        while True:
            key = str(list(run_args_range.keys())[p])
            tails[key] += 1
            if tails[key] >= len(run_args_range[key]):
                print(f"tails[{key}] {tails[key]} => 0, p {p} => {p + 1}")
                tails[key] = 0
            else:
                break
            p += 1
            if p >= len(run_args_range.keys()):
                over = True
                break

        if over:
            break


def run_args_list(run_args_li: list):
    for run_args in run_args_li:
        if isinstance(run_args, list):
            run_args_list(run_args)
        else:
            run(run_args=run_args)


if __name__ == '__main__':
    # 这里具体写运行脚本方法
    run_args_list([{"Ns": Ns, "NBs": 128, "Ps": 2, "Qs": 7} for Ns in range(10000, 30000, 10000)])
    print(results)
    with open("result.json", "w", encoding="utf8") as f:
        json.dump(results, f, indent=2, sort_keys=True)
```

2. `python test.py` 运行测试
3. 得到部分测试数据如下：

```
{
  "12000_256": 151.12,
  "16000_256": 173.68,
  "20000_256": 189.0,
  "24000_256": 198.43,
  "8000_256": 121.74
}
{
  "12000_256": 152.19,
  "12000_400": 135.08,
```

```
  "12000_512": 123.87,
  "16000_256": 174.34,
  "16000_400": 156.55,
  "16000_512": 147.49,
  "8000_256": 119.02,
  "8000_400": 100.86,
  "8000_512": 92.867
}{
  "12000_128": 153.33,
  "12000_256": 152.09,
  "12000_64": 123.33,
  "16000_128": 169.28,
  "16000_256": 174.56,
  "16000_64": 129.12,
  "8000_128": 131.07,
  "8000_256": 118.41,
  "8000_64": 109.83
}{
  "P2_Q8_N16000": 168.59,
  "P2_Q8_N20000": 178.89,
  "P2_Q8_N24000": 184.34,
  "P2_Q8_N28000": 189.64
}{
  "P2_Q8_N30000": 192.07,
  "P2_Q8_N40000": 199.0
}{
  "P1_Q16_N12000": 152.97,
  "P1_Q16_N16000": 166.75,
  "P1_Q16_N8000": 131.43,
  "P2_Q8_N12000": 153.28,
  "P2_Q8_N16000": 169.44,
  "P2_Q8_N8000": 127.26,
  "P4_Q4_N12000": 137.8,
  "P4_Q4_N16000": 156.89,
  "P4_Q4_N8000": 113.06
}{
  "P2_Q7_N10000": 184.85,
  "P2_Q7_N20000": 219.31
}
```


HPL.dat 运行参数解析

```
HPLinpack benchmark input file # 说明文字
Innovative Computing Laboratory, University of Tennessee # 说明文字
test.out     output file name (if any) # 输出文件
6            device out (6=stdout,7=stderr,file) # 输出文件
1            # of problems sizes (N) # 测试的矩阵个数
{Ns}         Ns # 测试的矩阵规模
1            # of NBs # 第几个NB
{NBs}        NBs # 分块大小
1            PMAP process mapping (0=Row-,1=Column-major) # 二维处理器网格设置
1            # of process grids (P x Q)
{Ps}         Ps
{Qs}         Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

经验参数：

1. $Ps \times Qs = N_{\text{cores}}$时，能用上所有的核心
2. $\frac{Qs}{4} \leq Ps \leq \frac{Qs}{2}$时效果较好
3. 一般来说，Ns越大越能增加实际计算比例，效果越好

4. 对 OpenBLAS，$NBs \approx 256$时效果比较好

实验结果

1. 在此实验环境下，（如果理论值没有计算错误的话），CPU实际计算的利用率能达到$\frac{219.31}{588.8} \approx 37.25\%$
2. 得到一些经验参数设置规律（见上）。