安装数学库和使用 blas 库
实验手册

华为技术有限公司

目录

1 安装数学库和使用 blas 库

1.1 实验介绍

1.1.1 关于本实验

本实验旨在体验鲲鹏加速库的使用方法。并以数学库为例，展示了数学库对函数及向量计算性能带来的提升。

1.1.2 实验目的

- 掌握鲲鹏数学库的环境安装编译流程。
- 掌握鲲鹏数学库的性能测试过程，对比 KML 优化前后的性能。

1.1.3 实验流程

1.1.4 软件介绍

表1-1 软件链接

软件名称	下载链接
Putty	https://hcia.obs.cn-north-4.myhuaweicloud.com/v1.5/putty.exe

1.2 购买 ECS 并登陆系统

1.2.1 购买 ECS

步骤 1　浏览器输入 www.huaweicloud.com，登录华为云官网。

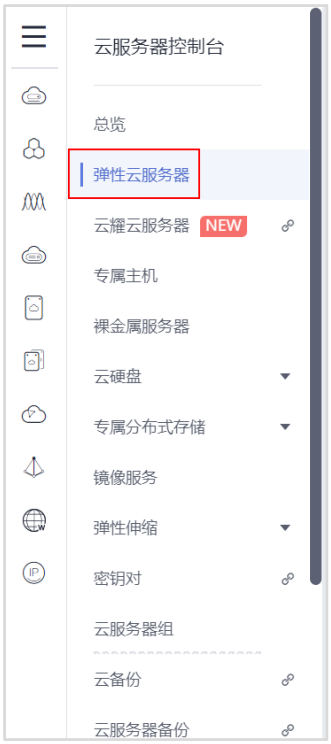
步骤 2　点击右上角的“登录”，打开登录窗口。

步骤 3　按要求输入账号密码，登录华为云。

步骤 4　点击右上角的“控制台”，打开控制台界面。

步骤 5　切换区域为“华北-北京四”。

步骤 6　点左侧导航栏的“弹性云服务器 ECS”，进入 ECS 控制台。



步骤 7　在 ECS 控制台界面点击右上角的“购买弹性云服务器”。

步骤 8　按照如下表配置弹性云服务器的基础配置参数。

表1-2 弹性云服务器基础配置参数

参数	配置
计费模式	按需计费
区域	华北-北京四
可用区	随机分配
CPU 架构	鲲鹏计算
规格	鲲鹏通用计算增强型 | kc1.large.2 | 2vCPUs | 4GB
镜像	公共镜像 | openEuler | openEuler 20.03 64bit with ARM(40GB)

系统盘	高 IO | 40GB

步骤 9　配置完成后点击“下一步：网络配置”，进入网络配置，按下表配置网络参数。

表1-3 弹性云服务器网络参数

参数	配置
网络	vpc-default | subnet-default |自动分配 IP 地址
安全组	default
弹性公网 IP	现在购买
线路	全动态 BGP
公网带宽	按流量计费
带宽大小	5Mbit/s

步骤 10　配置完成后，点击“下一步：高级配置”，按如下表配置 ECS 高级配置参数。

表1-4 弹性云服务器高级配置

参数	配置
云服务器名称	ecs-kp-test
登录凭证	密码
密码	请输入 8 位以上包含大小写字母、数字和特殊字符的密码
确认密码	请再次输入密码
云备份	暂不购买
云服务器组	不配置
高级选项	不勾选

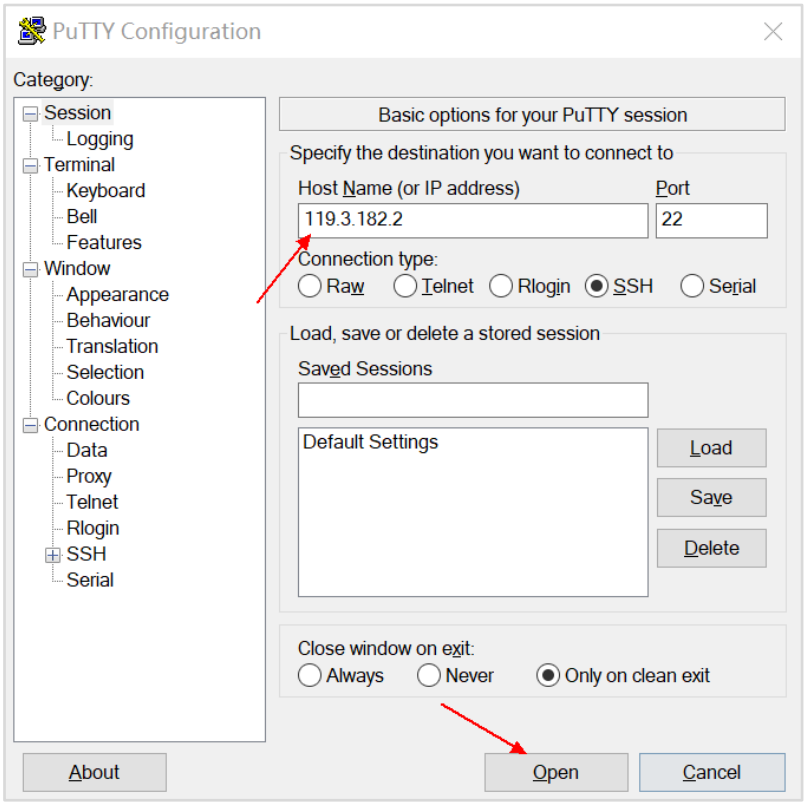
步骤 11　配置完成后点击右下角“下一步：确认配置”。勾选同意协议，然后点击：立即购买。

步骤 12　在提交任务成功后，点击“返回云服务器列表”，返回 ECS 控制台。

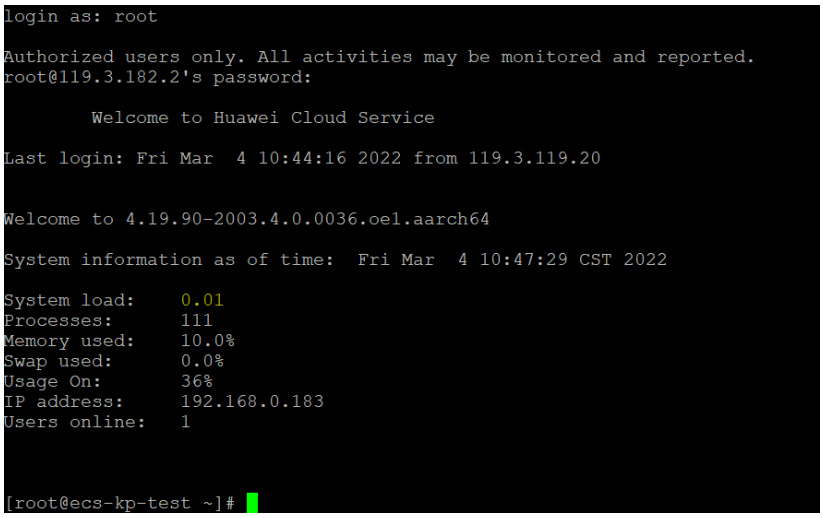
1.2.2 通过 ssh 登录系统

步骤 1　在 ECS 控制台查看 ECS 弹性 IP 地址。

步骤 2　打开电脑的 putty 软件，在“Host Name (or IP address)”栏输入上一步查找到的弹性 IP 地址，然后点击“open”。



步骤 3　在弹出的提示框中点击“Yes”，此时 putty 窗口中会显示登录界面。在 login as 后面输入登录用户名 root，按回车键；在 password 后面输入 root 用户密码，按回车键，登录系统。（此处输入密码不会有任何显示）



1.3 鲲鹏数学库实践

1.3.1 数学库安装

步骤 1　进入/home 目录下，在此目录下，下载实验所需的所有安装包并安装 KML。

```
[root@ecs-kp-test ~]# cd /home
[root@ecs-kp-test ~]# yum install --nogpgcheck git
[root@ecs-kp-test home]# git clone https://codehub.devcloud.cn-north-4.huaweicloud.com/cp2k00001/cp2k.git
[root@ecs-kp-test home]# cp cp2k/boostkit-kml-1.4.0-1.aarch64.rpm /home
[root@ecs-kp-test home]# rpm -ivh boostkit-kml-1.4.0-1.aarch64.rpm
```

```
[root@ecs-kp-test home]# rpm -ivh boostkit-kml-1.4.0-1.aarch64.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:boostkit-kml-1.4.0-1             ################################# [100%]
```

安装结束后，系统在环境变量 LD_LIBRARY_PATH 中自动添加 lib 文件夹所在目录即“/usr/local/kml/lib”。

步骤 2　执行 source 命令或重新登陆终端让环境变量生效。

```
[root@ecs-kp-test home]# source /etc/profile
```

步骤 3　安装后验证。

查看环境变量 LD_LIBRARY_PATH 是否包含 KML 的安装路径“/usr/local/kml/lib”：

```
[root@ecs-kp-test home]# env | grep LD_LIBRARY_PATH
```

如果变量包含安装路径，说明安装成功。

```
[root@ecs-kp-test home]# env | grep LD_LIBRARY_PATH
LD_LIBRARY_PATH=/usr/local/kml/lib/kspblas/multi:/usr/local/kml/lib/kspblas/single:/usr/local/kml/lib/kvml/multi:/usr/local/kml/lib/kvml/single:/usr/local/kml/lib/kblas/locking:/usr/local/kml/lib/kblas/omp:/usr/local/kml/lib/kblas/pthread:/usr/local/kml/lib/kblas/nolocking:/usr/local/kml/lib:
```

安装成功后在安装路径（默认路径是“/usr/local/kml”）下生成相应文件，其中，include 文件夹包含子库的头文件，lib 文件夹包含了数学库的动态库文件。

1.3.2 性能测试（KML_VML）

初始化长度为 100000 的向量 src，分别用两种方法求向量中每个元素的正弦函数值：

1、循环使用系统函数库的 sin 函数求解

2、调用 KML_VML 提供的向量三角函数 vdsin 求解

分别用计时器记录两种方法消耗的时间，对比 KML_VML 与系统函数库的性能。

步骤 1　编写测试代码，新建 test_sin.c 文件。

```
[root@ecs-kp-test home]# vim test_sin.c
```

按“i”进入编辑模式，并将以下代码复制粘贴至文件内：

```
#include <stdio.h>
#include <sys/time.h>
#include <math.h>
#include "kvml.h"
#define LEN 100000

int main()
{
    //数据初始化
    double src[LEN] = {0};
    double dst1[LEN] = {0};
    double dst2[LEN] = {0};
    for (int i = 0; i < LEN; i++) {
        src[i] = i;
    }
    struct timeval start, end;
    long t;
gettimeofday(&start, NULL);
//使用系统数学库对向量中每个元素求正弦值
    for (int i = 0; i < LEN; i++) {
        dst1[i] = sin(src[i]);
    }
gettimeofday(&end, NULL);
//记录计算时间
    t = 1000000 * ( end.tv_sec - start.tv_sec ) + end.tv_usec - start.tv_usec;
    printf("Calculate Time without KML_VML: %ld us\n", t);

gettimeofday(&start, NULL);
//使用 KML_VML 提供的函数计算向量中每个元素的正弦值
    vdsin(LEN, src, dst2);
    gettimeofday(&end, NULL);
    t = 1000000 * ( end.tv_sec - start.tv_sec ) + end.tv_usec - start.tv_usec;
    printf("Calculate Time with KML_VML: %ld us\n", t);

    return 0;
}
```

按“Esc”键，输入:wq!，按“Enter”保存并退出编辑。

步骤 2　配置环境变量并编译文件。

```
[root@ecs-kp-test home]# export LD_LIBRARY_PATH=/usr/local/kml/lib/kvml/single:$LD_LIBRARY_PATH
```

编译时添加动态库和头文件所在路径，并链接系统数学库和 KML_VML 动态库。此实验案例使用单线程版本。

```
[root@ecs-kp-test home]# gcc test_sin.c -lm -o test -L /usr/local/kml/lib/kvml/single -lkvml -I /usr/local/kml/include
```

使用 ldd 指令检查程序依赖库是否准确链接。

```
[root@ecs-kp-test home]# ldd test
```

```
[root@ecs-kp-test home]# ldd test
        linux-vdso.so.1 (0x0000fffbeec10000)
        libm.so.6 => /lib64/libm.so.6 (0x0000fffbeeb30000)
        libkvml.so.1.4.0 => /usr/local/kml/lib/kvml/single/libkvml.so.1.4.0 (0x0000fffbeead0000)
        libc.so.6 => /lib64/libc.so.6 (0x0000fffbee940000)
        /lib/ld-linux-aarch64.so.1 (0x0000fffbeec20000)
```

步骤 3　性能测试。

执行可执行文件，进行性能对比。

```
[root@ecs-kp-test home]# ./test
```

```
[root@ecs-kp-test home]# ./test
Calculate Time without KML_VML: 2922 us
Calculate Time with KML_VML: 783 us
```

步骤 4　结果分析。

结果显示，对于一个长度为 100000 的数组，用 C 语言的 for 循环实现求正弦函数值，需要 2922 微秒，而使用 KML_VML 仅需要 783 微秒，性能提升 3 倍。

1.3.3 性能测试（KML_BLAS）

初始化规模为 1000*300 的矩阵 A，长度为 300 的向量 x，长度为 1000 的向量 y1 和 y2。分别用两种方法求矩阵-向量乘加运算，即 y=alpha*A*x+beta*y：

1、按照矩阵-向量的乘加规则实现算法求解；

2、调用 KML_BLAS 提供的函数 cblas_dgemv 求解。

分别用计时器记录两种方法消耗的时间，对比 KML_BLAS 与手动实现矩阵乘加的性能。

步骤 1　编写测试代码，新建 test_gemv.c 文件。

```
[root@ecs-kp-test home]# vim test_gemv.c
```

按“i”进入编辑模式，并将以下代码复制粘贴至文件内：

```
#include <stdio.h>
#include <sys/time.h>
#include "kblas.h"
#define M 1000
#define N 300

int main()
{
    //数据初始化
    double A[M * N] = {0};
    double x[N] = {0};
    double y1[M] = {0};
    double y2[M] = {0};
    for (int i = 0; i < M * N; i++) {
```

```
        A[i] = i;
    }
    for (int i = 0; i < N; i++) {
        x[i] = i;
    }
    for (int i = 0; i < M; i++) {
        y1[i] = 1;
        y2[i] = 1;
    }
    double alpha = 1.2;
    double beta = 2.5;

    struct timeval start, end;
    long t;
    gettimeofday(&start, NULL);
    //按照矩阵-向量乘法规则实现 y1=alpha*A*x+beta*y1
    for (int i = 0; i < M; i++) {
        double tmp = 0;
        for (int j = 0; j < N; j++) {
            tmp += (A[i * N + j] * x[j]);
        }
        y1[i] = alpha * tmp + beta * y1[i];
    }
    gettimeofday(&end, NULL);
    t = 1000000 * ( end.tv_sec - start.tv_sec ) + end.tv_usec - start.tv_usec;
    printf("Calculate Time without KML_BLAS: %ld us\n", t);

    gettimeofday(&start, NULL);
    //调用 KML_BLAS 提供的 gemv 函数实现 y1=alpha*A*x+beta*y1
    cblas_dgemv(CblasRowMajor, CblasNoTrans, M, N, alpha, A, N, x, 1, beta, y2, 1);
    gettimeofday(&end, NULL);
    t = 1000000 * ( end.tv_sec - start.tv_sec ) + end.tv_usec - start.tv_usec;
    printf("Calculate Time with KML_BLAS: %ld us\n", t);

    return 0;
}
```

按“Esc”键，输入:wq!，按“Enter”保存并退出编辑。

步骤 2　配置环境变量并编译文件。

```
[root@ecs-kp-test home]# export
LD_LIBRARY_PATH=/usr/local/kml/lib/kblas/nolocking:$LD_LIBRARY_PATH
```

编译时添加动态库和头文件所在路径，并链接 KML_BLAS 动态库。

```
[root@ecs-kp-test home]# gcc test_gemv.c -g -o test2 -L /usr/local/kml/lib/kblas/nolocking -lkblas -I
/usr/local/kml/include
```

使用 ldd 指令检查程序依赖库是否准确链接。

```
[root@ecs-kp-test home]# ldd test2
```

```
[root@ecs-kp-test home]# ldd test2
        linux-vdso.so.1 (0x0000fffc054c0000)
        libkblas_armv8s_v1.4.0.so => /usr/local/kml/lib/kblas/nolocking/libkblas_armv8s_v1.4.0.so (0x0000fffc04420000)
        libc.so.6 => /lib64/libc.so.6 (0x0000fffc04290000)
        libm.so.6 => /lib64/libm.so.6 (0x0000fffc041c0000)
        /lib/ld-linux-aarch64.so.1 (0x0000fffc054d0000)
```

步骤 3　性能测试。

执行可执行文件，进行性能对比。

```
[root@ecs-kp-test home]# ./test2
```

```
[root@ecs-kp-test home]# ./test2
Calculate Time without KML_BLAS: 1720 us
Calculate Time with KML_BLAS: 212 us
```

步骤 4　结果分析。

结果显示，计算一个 1000*300 的矩阵-向量乘加运算，用 C 语言的 for 循环实现，需要 1720 微秒，而使用 KML_BLAS 仅需要 212 微秒，性能提升 7 倍。

1.4 思考题

如果在使用 java 语言开发的软件中想使用 BLAS 和 VML 库该如何操作?

答：使用 KML_JAVA 库。它是基于 JNI 技术，用 Java 语言封装的 KML 数学库。详细使用方法请参考：

https://support.huaweicloud.com/devg-kml-kunpengaccel/kunpengaccel_kml_16_0313.html