doi: 10.7690/bgzdh.2022.01.013

# 一种基于 QML 和 GPU 编程的实时显示软件

王　华，施　斌，李　阳，朱　瑾
（中国卫星海上测控部技术部，江苏 江阴 214431）

**摘要：**为实现国产自主可控，基于国产 CPU 和麒麟操作系统，采用 Qt 元语言（qt meta language，QML）和图形处理器（graphics processing unit，GPU）编程的模式设计一种跨平台的实时显示软件。详细阐述基于 QML 的软件架构设计，并对软件设计中的 QML、C++及 GPU 的通信模式、QML 下 CPU+GPU 编程实时显示性能优化方法以及 QML 显示控件设计等关键技术进行说明，验证其应用。结果证明：该模型具有可靠、稳定、高效和跨平台等特点，满足实时显示应用要求。

**关键词：**麒麟操作系统；Qt 元语言；图形处理器；实时显示；性能优化方法
**中图分类号：**TP31　　**文献标志码：**A

# A Real-time Display Software Based on QML and GPU Programming

Wang Hua, Shi Bin, Li Yang, Zhu Jin
(*Technology Department*, *Satellite Maritime Tracking & Control Department of China*, *Jiangyin* 214431, *China*)

**Abstract:** In order to achieve domestic autonomy and controllability, based on domestic CPU and Kylin operating system, a cross-platform real-time display software is designed by using qt meta language (QML) and graphics processing unit (GPU) programming mode. The software architecture design based on QML is described in detail, and the key technologies in the software design, such as the communication mode of QML, C + + and GPU, the real-time display performance optimization method of CPU + GPU programming under QML, and the design of QML display control are described in detail, and then the verification application is carried out. The application results show that the model is reliable, stable, efficient and cross-platform, and meets the requirements of real-time display applications.

**Keywords:** Kylin operating system; Qt meta language; graphic processing unit; real-time display; performance optimization method

## 0　引言

在国产自主可控的要求下，海上测控实时显示软件要求具备在国产 CPU 核心器件和麒麟操作系统下运行的能力，同时满足实时刷新显示，保证软件运行稳定、可靠，具有跨平台等要求，其功能要求适用性强、易于操作、人机交互友好、自动化程度高，实时数据显示信息丰富、即时性好。由于国产 CPU 显示终端和麒麟操作系统的应用还不够成熟，整机的性能管理和优化还有很大的提升空间，因此软件的架构设计，特别是性能设计尤为重要，要针对数据的图形化显示，重点减少内存占有率和 CPU 开销，此外还要满足高密度测控使用要求。笔者基于国产 CPU 和麒麟操作系统，采用 Qt 元语言（QML）和图形处理器（GPU）编程的模式，设计一种跨平台的实时显示软件。

## 1　QML 和 GPU 编程

### 1.1　QML 编程

QML 是一种用来描述应用程序界面的声明式脚本语言，具有良好的易读性，以可视化组件及其交互和相互关联的方式来描述界面，使组件能在动态行为中互相连接，并支持在用户界面上方便地复用和定制组件。Qt 是一个跨平台应用的框架，Qt Quick 是 Qt 为 QML 提供的一套类库，由 QML 标准类型和功能组成，包括可视化类、交互类型、模型和视图、粒子系统和渲染效果等，可迅速开发高品质、流畅的 UI 界面[1-2]。

在应用中，后台应用逻辑和业务逻辑运算采用 Qt C++，前台可视化界面交互的开发使用 QML。

### 1.2　GPU 编程

GPU 适合处理大量类型统一的数据。由于图形渲染的高度并行性，GPU 在处理能力和存储器带宽上相对于 CPU 有明显优势，一般而言适合 GPU 运算的应用有运算密集、高度并行、控制简单、分多个阶段执行等特点[3]。主流 GPU 编程接口有 CUDA、OpenGL、DirectCompute。CUDA 是英伟达公司开发专门针对 N 卡进行 GPU 编程的接口；OpenGL

**收稿日期：**2021-09-08；**修回日期：**2021-10-28
**作者简介：**王　华（1977—），女，河南人，硕士，高级工程师，从事航天测控显示系统架构设计及新技术研究与应用等研究。
E-mail: whwhindcolor@126.com。

(open graphics library)开源的 GPU 编程接口，作为一个 3D/2D 图形 API，具有硬件的无关性和跨操作系统平台的特点，其核心库具功能丰富、强大而又易用，是 C 语言编写的图形开发包，其 C/S 模式工作，在网络环境和集群环境下，可通过显卡加速，OpenGL 的绘图、渲染速度会极大提高，发挥强大的运算威力，OpenGL 成为高性能图形和交互性场景处理的行业标准[4]；DirectCompute 是微软开发的 GPU 编程接口，只能用于 Windows 系统。

## 2 基于 QML 的实时显示软件架构设计

实时显示软件运行于基于国产 CPU 的计算机上，采用国产 Kylin 操作系统，软件架构设计如图 1 所示[5]。

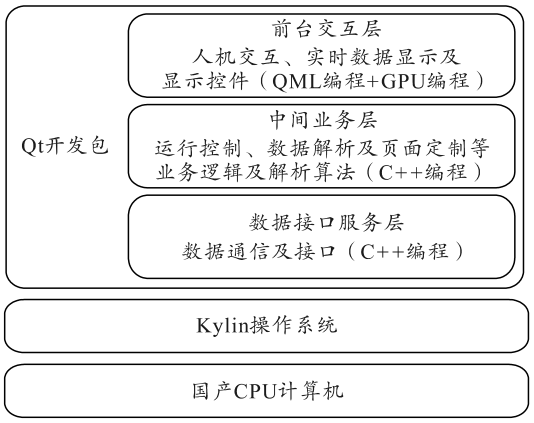


图 1 软件架构设计

### 2.1 前台交互层

前台交互层主要是前台人机交互、实时数据显示及各类显示控件，采用 QML 编程+GPU 编程的模式实现。主要包括软件运行、停止、发布、更新、页面定制等人机交互，所有显示控件均具有控件大小、位置、颜色和字体等静态属性以及显示的信源、信宿、信类和代号等数据源属性根据需要和显示要求进行定制，可满足今后显示一体化的要求。

### 2.2 中间业务层

中间层进行运行控制、数据解析及页面定制等业务逻辑及解析算法，采用 C++编程的模式实现。后台运行控制逻辑主要负责后台处理与显示页面的控制响应，包括对应前台命令解析、后台逻辑解析 CRP(compact result protocol)数据流产生 CRP 数据对象，形成数据表以及触发数据触发器，用于触发显示控件以及连接对象管理器，触发状态机切换程序运行状态；页面定制用于显示界面注册、显示控件注册、触发器注册、定制页面注册、绑定数据的显示控件等。

### 2.3 数据接口服务层

数据接口服务层包括后台数据通信及接口，采用 C++编程的模式实现。数据接口服务层用于与网发网收数据接口，与软件配置参数、理论数据、显示数据包、显示控件包、软件升级包文件、数据备份文件、CRP 数据接口文件等各类数据文件的接口。

## 3 QML 和 GPU 编程的实时显示方法设计

### 3.1 QML、C++及 GPU 的通信模式

QML 的引擎由 C++实现，但是在 QML 中不能对 C++中的对象直接访问，在 2 种运行环境间相互访问需要建立访问通道。Qt 提供了 2 种方式实现 QML 与 C++之间的交互：1) 在 C++中设计一个类，注册为 QML 环境的类型，在 QML 环境中使用该类型创建对象实例，在 Qt 中可使用 Q_INVOKABLE 宏来定义一个类成员函数让其被元对象系统调用，在 QML 中就可直接访问该函数[6]；2) 在 C++构造对象，将这个对象直接设置成 QML 的属性，在 QML 环境中直接使用。具体实现方式需使用 Qt 中的信号与槽机制，实时数据显示中的大多数应用需求是 QML 访问 C++中的对象[7]。

在海上测控实时显示软件数据刷新显示时，实时数据通信流程如图 2 所示。

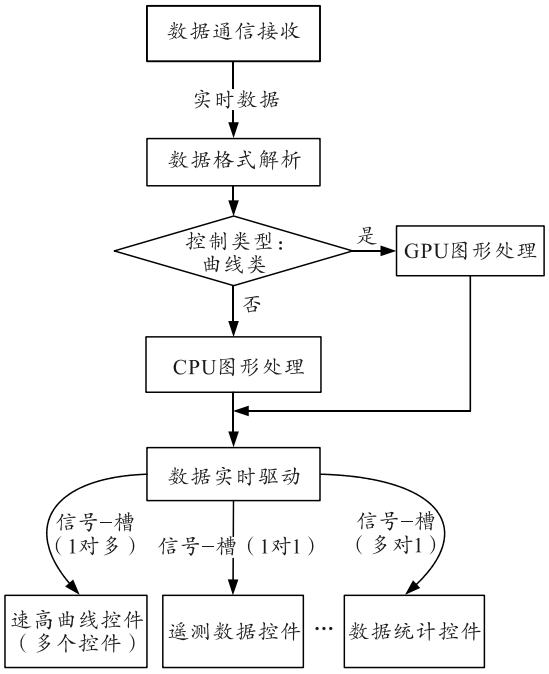


图 2 实时数据通信流程

首先 QML 访问 C++中的对象，通过后台的 C++编程部分进行数据的接收、数据格式的自动解析，

判断数据类型，如果是曲线类等数据进行 GPU 图形处理，否则进行 CPU 图形处理，处理完毕后通过“信号槽”（signal and solt）的通信机制，通知对应的数据显示控件，当指定的实时数据信号触发发射时，与该信号相关联的所有显示控件槽函数将会立刻执行，软件中通过使用 QObject 对象的 connect()函数来实现信号与槽的连接，可以将单个信号与多个槽连接，比如弹道数据要求显示在不同的控件中，当弹道数据信号发生变化发射时，相关的速高控件、星下点等多个显示控件槽函数会被对应调用，并顺序执行；也可以将多个信号与单个槽连接，比如数据统计控件包含多类数据统计，当其中任意数据信号发生变化，发射出新数值时，该数据统计槽函数就会被调用；甚至还可以将一个信号与另外一个信号相连接。

### 3.2　QML 下 CPU+GPU 编程实时显示性能优化

海上测控实时显示软件中，性能要求高的是实时数据能准确、高效图形化显示，比如实时数据曲线刷新显示策略、图形放大缩小模式、图形的前后滚屏显示等，在 QML 下实时数据显示的优化，采用 GPU+ CPU 分工协作处理的处理机制，显示控件中对于数值、列表显示及算法复杂的、精度要求高的等都直接进行 CPU 显示计算和逻辑运算，GPU 进行大量的曲线、图形学计算及并行处理显示，通过 CPU 串行计算、指令的逻辑控制及 GPU 的并行计算融合、协同工作，可提高 QML 下实时显示的性能和效率。

图 3 所示为基于 CPU+GPU 的实时数据显示流程[8]。实时数据利用后台信号-槽后台数据驱动机制，将数据同步到各类显示控件，一方面 CPU 同步进行相关显示数据的缓存，另一方面前台实时数据的显示刷新、区域数据图形的放大缩小及根据时间轴进行数据图形的前后滚屏显示，分别根据不同的显示策略进行设计驱动，CPU 和 GPU 进行不同类型数据通信，最后利用 GPU 进行并行绘制显示，GPU 编程接口采用开源的 Open GL 实现[9-10]。

基于 GPU 的 2 维显示控件图形绘制，应用于飞腾 1500A 处理器国产平台，在处理高频率数据、多线层任务时处于高负荷运行状态。为了提升实时显示整体性能和图形加速效果，使用 OpenGL 硬件加速的绘图引擎绘制图形。具体实现采用图形卡驱动程序的形式，将 OpenGL 调用传递给硬件驱动，直接与图形显示硬件进行通信。

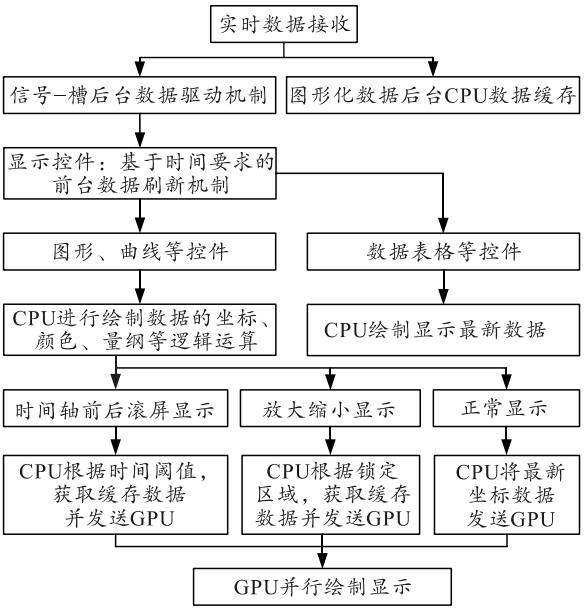


**图 3　CPU+GPU 的实时数据显示流程**

GPU 的 2 维显示控件图形绘制基本处理流程：首先，CPU 程序通过 OpenGL 接口函数对系统框架的软、硬件环境进行初始化，并动态编译 Cg 程序；其次，把待处理的图像数据打包成纹理流加载到 GPU 图形绘制流水线中；再次，执行 OpenGL 绘制操作，相应的顶点程序和片段程序就会在 GPU 上对图像进行并行化处理；最后，GPU 程序结束处理流程。

具体绘制中实现 Qt 和 OpenGL 之间图形接口机制的前提条件是设置像素格式、OpenGL 建立 RC、OpenGL 调用绘图原语绘图以及通过信号槽机制关联 DC（Device Context）与 RC（Rendering Context）。Linux API 提供了几个操作 RC 的函数，包括建立、复制、使用、删除、查询等。

GPU 的 2 维显示控件图形绘制具体代码实现如下：

1）设置 Include 文件及全局变量，添加 OpenGL 库文件。每个程序开始 4 行包括了窗口及 OpenGL 所使用的库的头文件：

```
#include GLinux.h    //视窗文件
#include g1\gl.h     //OpenGL 库文件的头文件
#include g1\glu.h    //GLu32. lib 的头文件
#include g1\glaux.h //Glaux. lib 的头文件
```

接下来定义在程序中计划使用的所有变量：

```
HGLRC hRC=NULL;     //定义渲染环境 HDC
hDC=NULL;                //私有的 GDI 设备环境
HWND hWnd=NULL;          //得到窗口句柄
HINSTANCE hlnstance; //得到程序的例子
```

2) 设置窗口大小、斜率及全屏标志变量等像素格式。变量 fullscreen 在全屏模式下值为 TRUE，在窗口模式下值为 FALSE，其参数设置可使程序的每个过程和函数都知道程序是否运行在全屏模式，具体代码如下：

```
RECT rect;
Intsw = 640;
Intsh=480
Boolfullscreen = 1
Glfloat  aspect;
```

3) 创建 Qt 与 OpenGL 的应用接口，进行不同显示方法的绘制编程。实时点绘制代码如下：

```
void ChartDataSet::addSeries(QAbstractSeries *series)
{
if (m_chart&&m_chart->chartType() ==QChart::ChartTypePolar) {//m_chart 为绘制图表指针变量；
    if (!(series->type() == QAbstractSeries::SeriesTypeArea || series->type() == QAbstractSeries::SeriesTypeLine || series->type() == QAbstractSeries::SeriesTypeScatter || series->type() == QAbstractSeries:: SeriesTypeSpline)) {
        qWarning()<<QObject::tr("Can not add series. Series type is not supported by a polar chart.");
        return; }
    // Disable OpenGL for series in polar charts
    series->setUseOpenGL(false);
    series->d_ptr->setDomain(new XYPolarDomain());
    // Set the correct domain for upper and lower series too
    if (series->type() == QAbstractSeries::SeriesTypeArea) {//QabstractSeries 为自定义散点图类；
        foreach (QObject *child, series->children()) {
            if (qobject_cast<QAbstractSeries *>(child)) {QAbstractSeries *childSeries = qobject_cast< QAbstractSeries *>(child);
                childSeries->d_ptr-> setDomain(new XYPolarDomain()); }
                                                    }
                }
    series = chart.createSeries(type, name + " " + chart.count);
    multiplier = 10; //设置多重绘制数，优化绘制效率；
    if (openGl) {//是否采用 openGl 绘制的 BOOL 全局变量；
        series.useOpenGL = true;
        multiplier = 100; }
    for (inti = chart.axisX().min * multiplier; i<chart.axisX().max * multiplier; i++) {
        double y = Math.random() * (chart.axisY().max - chart.axisY().min) + chart.axisY().min;
        double x=(Math.random()+i)/ multiplier;
    series.append(x, y);
        } else { series->d_ptr->setDomain(new XYDomain()); }
    series->d_ptr->initializeDomain();
    m_seriesList.append(series);
    series->setParent(this);
    series->d_ptr->m_chart = m_chart;
    emit seriesAdded(series);}
```

### 3.3 QML 显示控件设计

使用 QML 语言开发的控件，文件格式为*.qml 文本文件。运行时通过 QML 引擎加载 qml 格式字符串，实现动态快速创建控件。如控件需要修改，也只需更动文本内容，再重新加载即可，对于控件的更动及测试十分灵活、便捷。配置设计如图 4 所示。

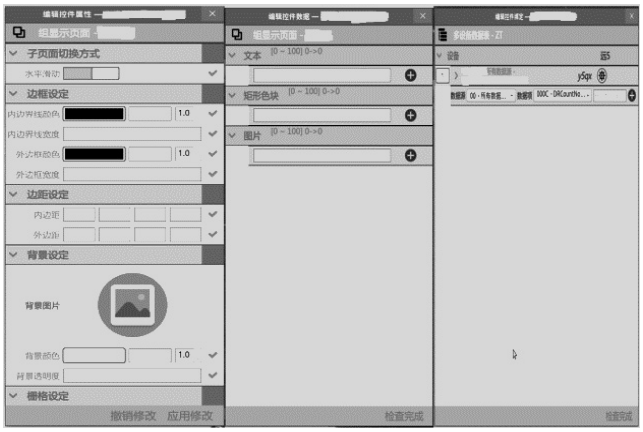

图 4 QML 显示控件之双轴曲线控件配置设计

QML 显示控件模型设计解决统一定制、组装以及数据来源等问题，具体如下：

1) 控件属性组，设计为若干属性组，是为了方便定制而将相关的属性进行的分类。实时显示控件设计为通用 3 类属性组，分别为显示属性组、显示数据单元属性组、显示数据源与数据项属性组。比如对于箭下点轨迹控件，可能的属性组有世界地图设定属性组、测站显示设定属性组等。

2) 控件属性组下属性项，设计为若干属性项，

每个属性项定义了属性的值类型，是否非空等属性规范，每个属性项有一个当前值。

显示控件显示属性组属性项设计，属性项包括绘制数据点数、抗锯齿、GPU、底图颜色、绘制区颜色、图标题(名称、题颜色、字体、大小)、瞄准线(颜色、线型、线宽、端点、连接)等。在属性中，设计为性能优化包括CPU或GPU显示，默认为GPU显示。

显示控件显示数据单元属性组属性项设计，数据单元属性项是设置对应的显示模式及显示值类型，不同控件其属性项不同，比如双轴曲线控件，其对应属性项设计为横轴为时间线属性项，V 为左轴数据线属性项，H 为右轴数据线属性项，为后续的数据源和数据项配置提供配置基础。

显示控件的显示数据源与数据项属性组属性项设计，该属性组用于绑定实时数据，其属性项设计，包括显示数据单元配置完成后，配置对应的显示数据源，再根据数据源的字段信息配置需显示的最终数据，数据配置完成。

3) 控件可以有若干附属控件与之关联。为了联合查看和实时数据比对的需要，应将多个相关控件显示有机结合起来，这些控件放置于主控件容器中。为了适应多输出设备不同分辨率的显示需求，将附属控件的显示区域与主控件相关联。各附属控件在主控件上透明度是不同的，深度也是有前有后，避免了遮蔽副作用，可以动态调整可视性对某附属控件进行显示屏蔽。

## 4　实际应用

根据以上设计，基于 QML 和 GPU 编程的实时显示软件运行于国产麒麟操作系统下。图 5 是对显示的各类数据进行实时表格化帧统计。

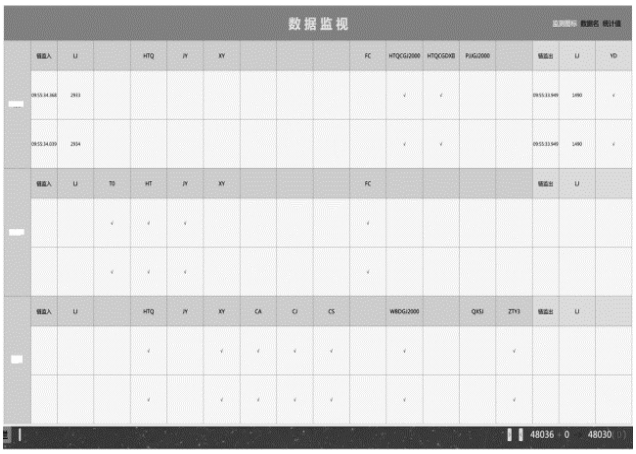


图 5　帧统计显示

图 6 是图形化仪表及数据显示的惯导设备状态，圆形码表分别显示设备的纵摇、横摇及航速，由实时数据驱动而动态变化，下方表格显示更详细的设备数据。

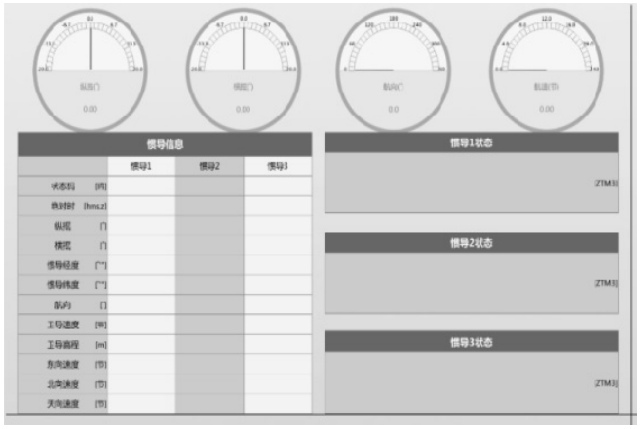


图 6　图形化仪表及数据显示

图 7 是图形化设备及实时数据显示的变形设备状态情况，可通过图形化的设备结构，实时显示各部分设备状态情况。

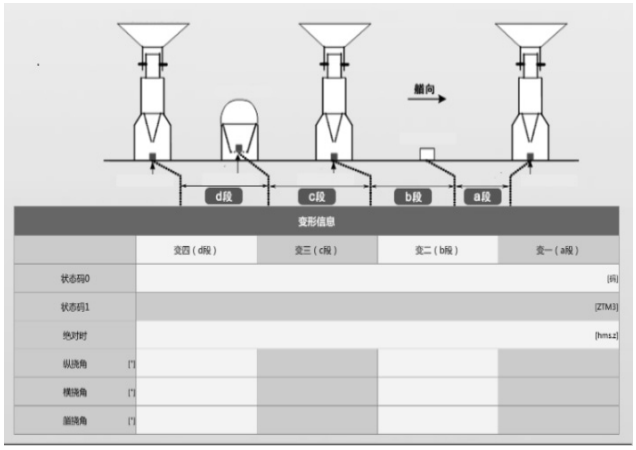


图 7　图形化设备及实时数据显示

实时显示软件配置 25 个实时显示页面，运行微机配置为国产 CPU(飞腾 1500A 4 核)、DDR3 8GB 内存、1TB 硬盘、2 个千兆网口、24" LCD(分辨率为 1 920×1 200)、银河麒麟桌面操作系统及独立显卡 R5235，仿真实际数据情况对软件性能进行监控。首先，打开“系统监视器”查看整机运行性能。如图 8 所示，实时数据流量为 11.2 KB/s，4 核 CPU 单核最高占用 35.5%、最低 14.7%，8G 内存已用 3.3GB。其次，打开终端下，用“top”命令查看实时显示软件(即框中“app”)的运行性能。如图 9 所示，CPU 占用 26.4%，内存占用 9.3%，切换页面正常，图形化曲线、数据刷新显示流畅，满足海上测控任务实时显示需求。

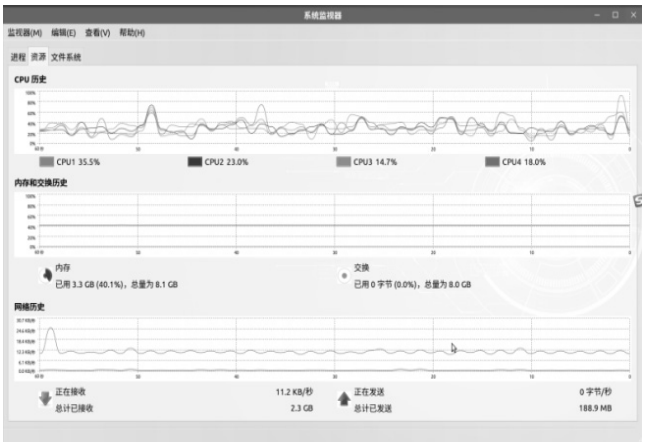


图 8 国产微机运行实时性能监控

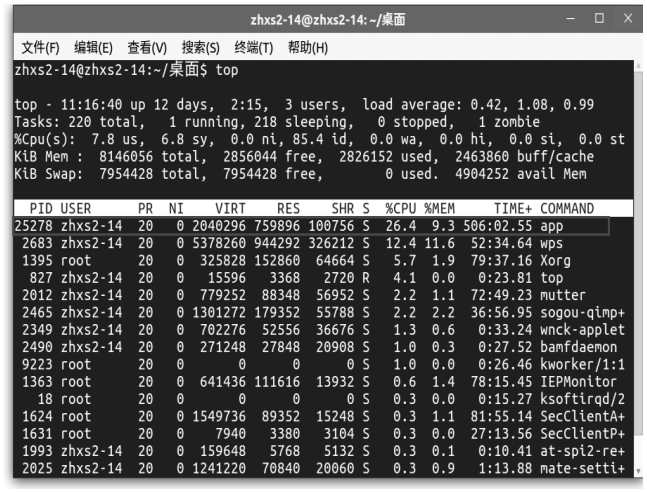


图 9 实时显示软件运行性能显示

## 5 结束语

笔者采用 QML 和 GPU 编程的模式，设计一种跨平台的实时显示软件，并进行验证应用。应用结果证明：该设计具有可靠、稳定、高效和跨平台等特点，满足海上测控实时显示应用要求。在我国计算机软件系统自主可控的大趋势下，文中采用的国产化软件设计模式和技术，具有一定的参考和使用价值，既可为航天系统同类的国产化软件的设计开发提供有效的解决方案和技术支持，又可为相关行业实时监视显示软件系统的国产化提供具体的样本示例和设计思路。

## 参考文献：

[1] 陆文周. Qt5 开发及实例[M]. 北京: 电子工业出版社, 2017: 580-582.

[2] Mark Summerfield. Qt高级编程[M]. 北京: 电子工业出版社, 2011: 56-57.

[3] 仇德元. GPU 编程技术[M]. 北京: 机械工业出版社, 2011: 5-7.

[4] Dave Shreiner. OpenGL 编程指南[M]. 北京: 机械工业出版社, 2010: 1-2.

[5] 覃征, 刑剑宽, 董金春, 等. 软件体系结构[M]. 北京: 清华大学出版社, 2008: 50-52.

[6] JasminBlanchette. C++ GUI Qt 4 编程[M]. 北京: 电子工业出版社, 2013: 366-371.

[7] 王维波, 栗宝鹃, 侯春望. Qt 5.9 C++开发指南[M]. 北京: 人民邮电出版社, 2018: 344-350.

[8] 吴凡贤. 基于 GPU 的地震数据图形快速绘制[D]. 成都: 电子科技大学, 2013: 6-29.

[9] 仲芊芊. 基于 Qt 的地震数据可视化技术的研究及应用[D]. 成都: 电子科技大学, 2010: 9-15.

[10] 夏海宏. 图像缩放及其 GPU 实现[D]. 杭州: 浙江大学, 2010: 8-10.