USER FRIENDLY EDGE COMPUTING IN FTI NETWORKS

Diarmuid Collins
Curtiss Wright
Dublin, Ireland
dcollins@curtisswright.com

ABSTRACT

Flight Test Instrumentation (FTI) networks typically require low power, small form factor equipment which can capture and process data in rugged environments.
Edge computing is a computing paradigm that brings data processing closer to the source of the data. In recent years, advances in low power processors and the proliferation of user friendly, single-board computers, has made edge computing more accessible and powerful.
These advances can be leveraged and integrated into an FTI network to provide for flexible data processing, both reducing post processing requirements and providing services on the FTI network that were previously not feasible. In addition to this, standards like IRIG-106 Chapter 7 allow for more flexible transfer of native network traffic to the ground, allowing these edge computing devices to pass the processed data, transparently to the ground. This paper will address some of the approaches, showing how user-friendly data processing can be brought to the FTI network, within existing power, weight and size constraints.

INTRODUCTION

For many years FTI applications have been required to capture and store of a significant volume of parameters. Typically this data is recorded for later processing with a small subset transferred to the ground for real time analysis. Edge computing is a paradigm that push data processing closer to the source, resulting in improved efficiency, real-time analysis and faster decision making.

This has significant implications for IoT and autonomous vehicles and can be leveraged for FTI applications. However, the environment in which flight test equipment is operated poses challenges with respect to wide temperature ranges, ruggedness and low power requirements.

This paper will consider edge computing, how it can help in FTI networks and how it can be easily adopted by Flight Test Instrumentation engineers.

EDGE COMPUTING

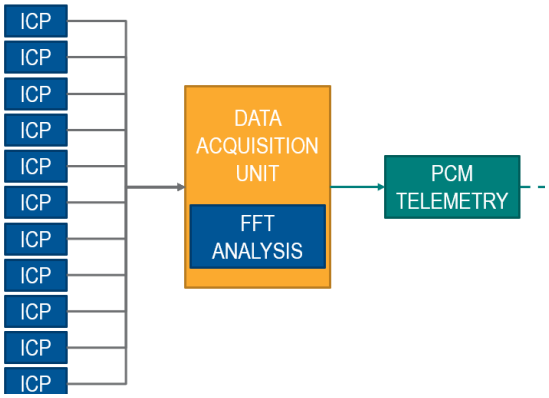
In traditional computing the data is sent from the acquisition site to a centralised server while in edge computing the data is processed at the periphery of the network. The movement of the data processing close to the sensor allows for a number of advantages.

Advantages

- Improved Efficiency and Cost Reduction: By processing data at the edge of the network, greater efficiency can be achieved by reducing the volume of data to be passed over the network. This also applies to data storage and processing costs, which can also be reduced by edge processing.
- Real time analysis: The local nature of the data processing allows for faster decision making and immediate action which is critical for time sensitive applications.
- Data latency reduction: As the sensor data is processed local to the acquisition any action resulting from the processing will have effect with reduced latency.
- Increased Data Security: By reducing the data being sent to a central recorder or over a wireless link, the security of the data can be improved.
- Flexibility and customised solutions: Edge computing allows for more flexibility when designing and deploying FTI acquisition systems. A more customised and specific solution can be applied on a per sensor site basis.

FTI Applications

These advantages can be leveraged in FTI networks in several ways. For analog measurements, the data processing and reduction that would normally be carried out on the ground or in post processing, can now be done on the acquisition hardware. Instead of recording a high sample rate accelerometer data, this data could be processed at the edge, for example performing frequency analysis on the data. Such type of edge computing has already been implemented in commercially available acquisition modules[1]



Data compression on analog samples could also be performed. For some analog sampled data, relatively high levels of compression could be achieved, reducing storage and bandwidth analysis. The following shows the compression ratio that were achieved on some typical analog data using various compression systems.

System	Ratio[1]	Relative Time[2]
Zip	2.68	1
Gzip	2.68	1
Lzma	6.11	19.9

Bzip2	5.51	3.2
Zstd	3.2	0.3

1.Compression ratio from the original data

2. The CPU effort relative to a baseline of zip.

For data acquired from digital buses, similar data compression could be used to reduce data volumes. In addition, actions based on events occurring on these digital buses could be implemented. For example, alert generation for critical events could be implemented at the acquisition site. This would result in lower response latency to events.

Finally edge computing can add "smarts" to older FTI equipment. Many flight test teams may have acquisition systems that were designed and purchased many years ago. This equipment can still perform the original specified acquisition many years later but do not have the features and "smarts" of more modern equipment. In these scenarios, the addition of edge computing equipment can allow the FTI team to use the older equipment to perform the data acquisition while processing the data on the latest edge computing equipment.

FTI SPECIFIC REQUIREMENTS

Flight Test Networks place a combination of requirements on the equipment that can be used for edge computing. As a result, this equipment should be carefully designed to meet these requirements.

Low Power: Edge computing equipment in FTI networks, should be low power devices. By minimising the power usage in the edge device, the power dissipation in the equipment is kept low. This minimises the self-heating effect and allow the equipment to operate in higher ambient temperatures.

Ruggedness: Like most FTI equipment, edge computing devices will be exposed to harsh environments. This means a high vibration, high temperature. In addition to the self-heating effects, this can bring component temperatures to 125°C.

Network Interface: FTI networks over time have moved to an ethernet based networking infrastructure. The processing elements in such an environment will require native networking support. Ideally Gigabit networking should be supported.

Hardware Acceleration: Certain application specific hardware accelerators can help meet power to performance metrics, that may not be possible with general purpose processors. This can range from graphics processing, network processing to machine learning. System-On-Chip devices, are generally designed including these co-processors.

A typical FTI module on which edge computing could be implemented could be based around an ARM or RISC-V SOC. They are generally lower power than the x86 based

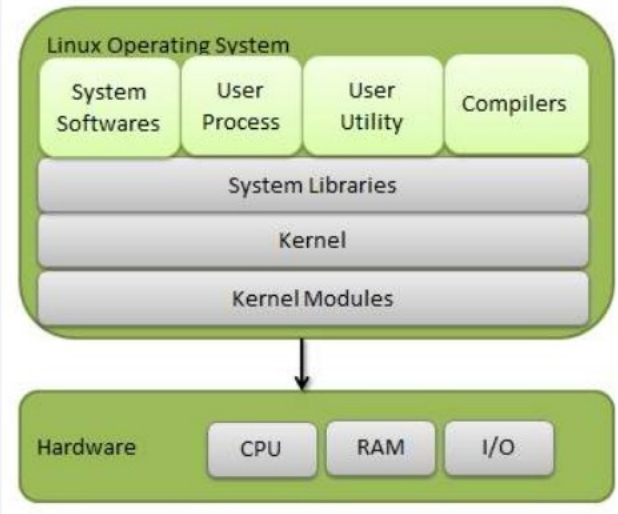
equivalent devices, ensuring the edge device can operate in the harsh environments mention previously.

LINUX BASED

There are many approaches to implementing edge computing in FTI equipment. Custom logic can be implemented in FPGAs which processes and analyses the data on acquisition. Bare metal software can be run on embedded processors. However, when the goal is to support user friendly computing on embedded devices, then the choices can be significantly reduced. A user should require the minimum of programming knowledge to implement the specific custom logic required on the acquisition module.

The starting point is the decision to use an Operating System on the hardware. With this choice the user is abstracted away from concerns about low level hardware control, memory management, and a host of other challenges. This decision does have a computational and memory overhead which has to be accounted for in the device design.

Linux is the most obvious choice of an operating system for use in edge devices, providing a wide array of benefits that are highly desirable for edge applications.



Software and Tools: Linux has an extensive ecosystem of software and tools, including open-source libraries, drivers, and applications. This provides engineers with a wide pool of resources for product development. This ecosystem accelerates development cycles, reduces time-to-market, and enhances product functionality.

Hardware Support: Linux also offers excellent hardware support with a wide range of device drivers for different hardware architectures. This broadens the choice of components that can be used to build the edge computing device.

Open Source: Being an open-source OS, Linux eliminates the need for expensive licensing fees, reducing overall development costs while running efficiently on low-power and

resource-constrained hardware. This makes it a cost-effective choice for embedded products with limited hardware resources.

Reliability: Linux is widely recognized for its exceptional stability, making it an ideal choice for embedded applications that require reliable and robust performance. The mature and time-tested design of Linux, developed and refined over many years ensures that the OS is stable and resilient to errors and crashes. The separation of user space and kernel space in Linux prevents processes from impacting the overall system stability. A wide network of engineers are working constantly resolving bugs and adding features to the kernel and system.

USER FRIENDLY PROGRAMMING

While powerful hardware in a rugged package is important, support for user-friendly programming is the critical aspect of any solution. FTI engineers require access to an acquisition module that will allow programming of applications that are easy to understand, use, and maintain. The choice of programming language is a significant decision in development, as it impacts the performance, maintainability, and scalability of the final product.

Some of the factors that contribute to the selection include, platform compatibility, community support and most importantly, expertise.

Compiled and interpreted languages are two different approaches that are used in writing programming code. In compiled languages, the code is translated into machine code by a compiler before it is executed. The resulting compiled code is typically faster and more efficient.[2][3] Compiled languages include, C, C#, Pascal among others.

On the other hand, interpreted languages do not require compilation, and the code is directly executed by the run time interpreter. This allows for faster development cycles, as changes to the code can be immediately executed without the need for compilation. Bash, Python, Javascript are some examples of interpreted languages.

After selecting Linux as the OS for the platform, the choice of programming language is broad. Each language comes with specific advantages and disadvantage but one metric to judge a language is the popularity. Using this, the ranked languages would generally include the following languages.[4]

- Java
- Python
- Javascript
- C/C++
- PHP
- SQL
- C#

PYTHON PROGRAMMING

Why Python?

While there's no one correct answer, one sensible choice is to use one of the most common interpreted language, Python.

By many metrics, Python is the most popular programming language in the world.[4] It is known for its simplicity, versatility, and extensive libraries. This combination makes it a good choice for FTI applications. One of the key advantages of Python is its easy-to-read syntax, which makes it accessible to beginners and experienced engineers alike.

Python has a large standard library with a "batteries included" approach. and a rich ecosystem of third-party libraries that support data analysis and data manipulation extensively. Python's cross-platform compatibility enables code to be written once and run on different operating systems, making it highly portable. This feature is especially useful in edge computing applications, where the development may occur on a Windows desktop computer and then be run to an ARM-based Linux device in the flight test chassis.

Performance

Python's interpreted nature and dynamic typing make it a highly productive language, allowing engineers to write code quickly and efficiently. The simple, readable syntax helps promote fast development cycles. Automatic garbage collection help prevent memory leaks and make it a reliable choice for long-running applications.

As previously mentioned, because Python is interpreted it can result in slower execution speeds compared to compiled languages. This can be particularly evident in computationally intensive tasks, where the overhead of interpreting each line of code can impact performance. This can be mitigated by using compiled libraries for these computationally intensive tasks. This approach is used widely in packages such as SymPy where the computational core is implemented in Fortran and C

Finally, Python's Global Interpreter Lock (GIL), an internal implementation detail that protects access to Python objects, can limit the performance of multi-threaded applications. The GIL restricts Python threads from utilizing multiple CPU cores fully, which can impact performance in CPU-bound applications that could otherwise benefit from parallel processing.

PYTHON DEBUGGING

A significant advantage of using an interpreted language is that the source code is portable across platforms. This allows engineers to develop and debug a Python application on a Windows PC and migrate it to a Linux platform with very limited changes.

Using off the shelf applications like git, samba and ssh, python code can easily be moved from a development PC to an FTI chassis.

While many applications can be developed and debugged in a local Windows of Linux desktop computer, interactive debug on the target chassis may sometimes be necessary. To do this there are several approaches.

PDB

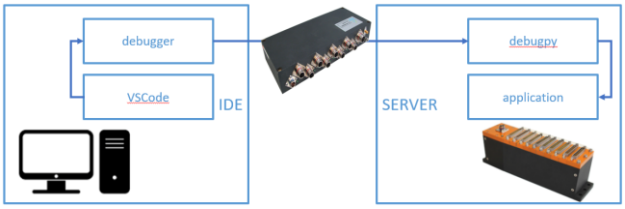
The Python debugger, commonly known as pdb, is a built-in module that allows developers to debug their Python code interactively.

pdb provides a set of commands that can be used to set breakpoints, step through code line by line, inspect variables, and execute arbitrary Python statements during runtime. When a breakpoint is hit, pdb stops the execution of the code and gives the developer control to examine the current state of the program and step through the code to understand its behaviour.

While pdb is powerful, being a command line tool, it is not the most user-friendly way to debug an application. It is run directly on the edge computing device, which means the user has to have remote access to the device, typically via ssh or serial ports.

Debugpy

Debugpy is a popular Python debugger written by Microsoft, that provides powerful debugging capabilities for Python code in Visual Studio Code (VSCode). The debugger is written around an open protocol called Debug Adapter Protocol. This allows other IDEs to use this same module to debug Python application.



Like pdb it allows engineers to set breakpoints, step through code, inspect variables, and perform other debugging tasks within their Python code directly from VSCode.

Debugpy supports both local and remote debugging, making it versatile for various debugging scenarios. This is particularly useful when debugging an application running on an acquisition chassis. Debugpy has a user-friendly interface, with easy-to-use commands and intuitive visualizations, making it a popular choice for developers who use VSCode or other compatible IDEs for Python development.

CHALLENGES

While edge computing does provide some significant advantages in an FTI network, it is not without its challenges. By providing a general computing platform in the FTI chassis, there is a power consumption overhead compared to dedicated hardware. An FPGA can be

used to design the very specific set of functions required, where as a CPU on which Linux will run, is designed to support a wide range of functions but in a less specific manner.

In addition, by selecting a user-friendly operating system on which to run the system, the determinism of such a system is reduced. The operating system abstracts the hardware through numerous layers of software, making the processing of the sensor data less deterministic. Real Time Operating Systems mitigate this to a point, but this in turn requires a lower level of expertise on the engineer's part, as well as limiting the programming languages available to use

By adding modules to the FTI network that require programming, expertise is required in the language domain selected. While this can be mitigated by the use of a well-supported and user-friendly language, some expertise is still required. One way to mitigate this risk is to outsource the development of the edge computing application.

CONCLUSIONS

The use of edge computing devices in an acquisition network opens possibilities for FTI engineers to implement custom processing of data, at the point of acquisition. The choice of widely used and supported programming languages on a familiar platform such as Linux, reduces the barrier to entry to such custom applications

REFERENCES

[1] Optimizing PCM Bandwidth Usage in Flight Test by Real-time Data Analysis During Flight, ETTC 2022, Pat Quinn
[2] Qualitative Assessment of Compiled, Interpreted and Hybrid Programming Languages. Ampomah, Ernest & Mensah, Ezekiel & Gilbert, Abilimi. (2017). Communications on Applied Electronics. 7. 8-13. 10.5120/cae2017652685.
[3] Ranking the Performance of Compiled and Interpreted Languages in Genetic Algorithms. Juan-Julian Merelo-Guerv, Israel Blancas-Alvarez, Pedro A. Castillo, Gustavo Romero , Pablo Garciıa-Sanchez , Victor M. Rivas, Mario Garc´ıa-Valdez, Amaury Hernandez- ´ Aguila and Mario Roman
[4] TIOBE Index https://www.tiobe.com/tiobe-index/