Panther Storage Acceleration SDK: Simplifying Hardware and Software Integration

WHITE PAPER

Author:

Pinaki Chanda
Director of Software Engineering

Introduction

MaxLinear's Panther storage accelerators and the accompanying MaxLinear storage acceleration software development kit (SDK) provide a solution to the growing demand for secure and efficient data storage. As organizations grapple with increasing data volumes, the Panther platform and SDK offer a robust combination of data reduction, encryption, and deduplication capabilities, enabling an impressive data reduction ratio while significantly reducing capital expenditures (CAPEX) and improving data protection.

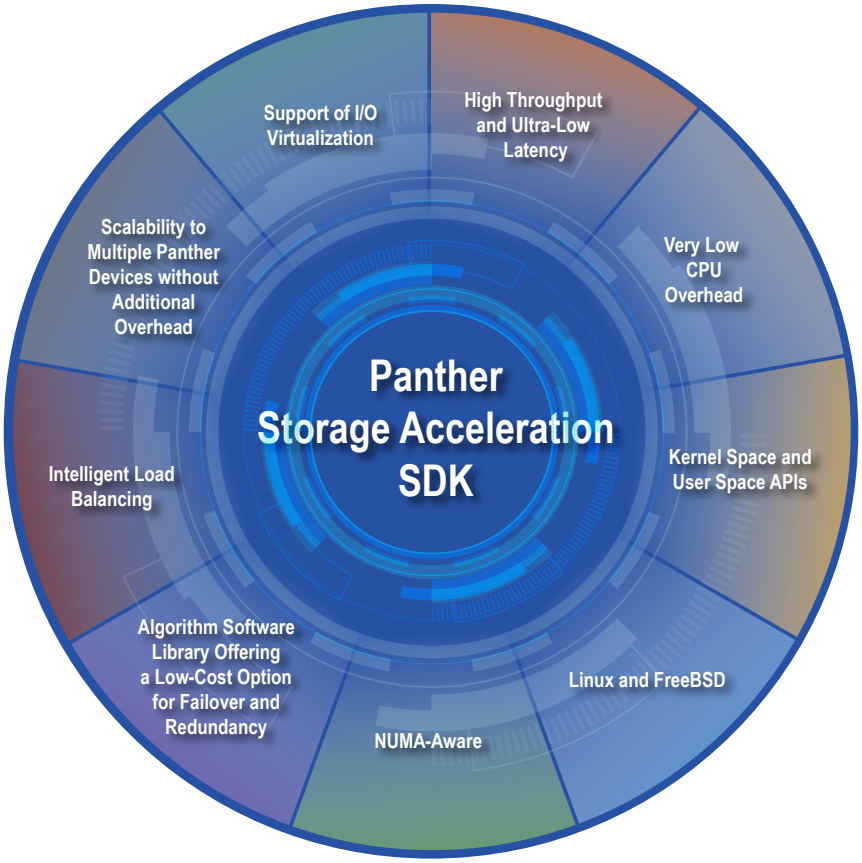

Figure 1: Storage Acceleration SDK Key Features

The SDK supports all transformation operations, including data transformation chaining for the Panther accelerator, through a simple set of APIs for initialization, session opening, command submission, session closing, and deinitialization. MaxLinear's innovative approach uses a single set of APIs, facilitating both software implementation of the algorithms and efficient offloading of hardware implementation. This streamlines software integration for transformation operations and enables flexible hardware offloading afterwards. This one-time integration with existing storage software ensures minimal disruption and smooth operations. The architecture adapts to high concurrency and throughput, delivering ultra-low latency for demanding applications while maintaining low CPU usage, making it scalable across various deployment scenarios.

The SDK is optimized for both Linux and FreeBSD on AMD and Intel platforms, allowing developers to fully leverage the hardware capabilities of the Panther accelerators.

Key Features

- **Unified APIs—**A streamlined set of APIs enables seamless integration and flexibility, allowing you to choose between software implementation or hardware offload as needed.
- **Transformation operation chaining—**The SDK leverages the unique capabilities of Panther acceleration, allowing you to define a sequence of transformation operations on a data block submitted from your application—all in a single command. This approach results in ultra-low latency and high throughput for data transformation operations.
- **One-time integration—**You can integrate the solution with existing storage software once, with the ability to offload tasks as needed to Panther storage accelerator hardware, ensuring minimal disruption and operational continuity.
- **Comprehensive transform support—**All transformation operations supported in the Panther architecture are available in optimized software implementations, ensuring consistent results.
- **High concurrency and throughput—**The solution supports a high level of concurrency, enabling high throughput and ultra-low latency for demanding applications.
- **Low CPU overhead—**The architecture is optimized for minimal CPU usage, making it highly scalable for a pool of offload accelerators.
- **Kernel and user space APIs—**The solution offers both kernel space and user space APIs, enabling versatile application integration.
- **Synchronous and asynchronous modes—**The solution supports both synchronous and asynchronous data transformation modes, meeting various operational needs.
- **NUMA awareness—**The design is non-uniform memory access (NUMA) aware, optimizing memory access patterns for better performance on AMD and Intel platforms.
- **Intelligent load balancing—**The solution includes intelligent load balancing mechanisms, ensuring efficient resource utilization across the system and transparent support of up to 16 panther accelerators (3.2Tbps).

Key Benefits

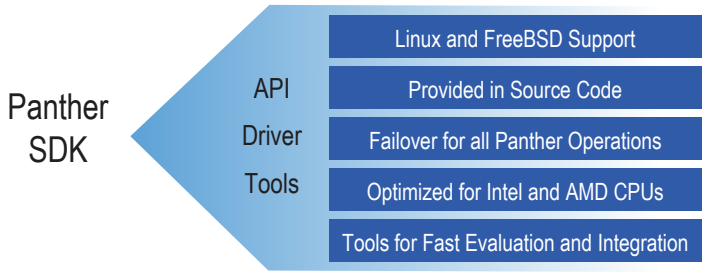

Figure 2: Storage Acceleration SDK Highlights

Accelerated software development and product integration

- The SDK and performance tuning tools allow software developers to quickly start working on software development and product-level integration of the Panther storage accelerators on their platforms.
- The SDK source code is distributed under a dual BSD/GPLv2 license, enabling projects to reach a wider audience.
- The reference applications provided in the SDK reduce the learning curve and make it easier to develop applications that leverage hardware capabilities.
- The SDK's Algorithm Software Library (ASL) enables parallel software development without the need for Panther hardware. This setup enables seamless plug-and-play integration with the Panther accelerator once development is complete.

Optimized performance and increased hardware utilization

- The SDK makes the most of the Panther V hardware features, including the highest acceleration throughput and ultra-low latency, resulting in optimal hardware utilization.

Documentation and examples

- The SDK includes several documents such as the *MxL890x Software Development Kit User Guide* (209UG), *MxL890x Software Development Kit (SDK) Getting Started User Guide* (210-CUG), and *MxL890x Raw Acceleration Application Program Interface (API) Reference Guide* (200AG), as well as example use cases using all hardware features. This saves time and reduces the risk of errors and incompatibilities.

User feedback

- MaxLinear periodically updates the SDK based on feedback from developers on performance, functionality support of new kernel versions and OS distributions, and user experience with the use cases.

SDK Overview

SDK Architecture

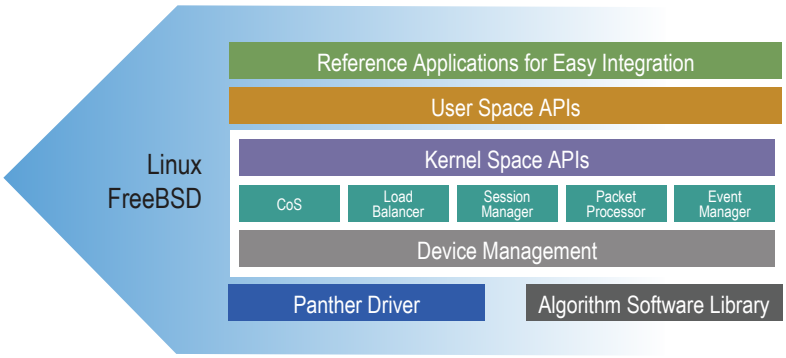

Figure 3: Storage Acceleration SDK Components

The SDK is structured into kernel-space and user-space components, giving access to accelerator features such as NVMe Protection Information (PI) verification, compression/decompression, deduplication, encryption, and cryptographic hash operations.

The Service Framework module facilitates the management of sessions, load balancing, user callbacks, and devices without relying on specific hardware. Its main features include:

- **Session management—**Handled by the session manager.
- **Device interface—**Established by the device manager to interface with the Panther accelerator's specific driver or ASL.
- **Command processing and result retrieval—**The packet processor creates commands for device execution, while the result-retrieval modules provide outcomes to user applications.
- **Load Balance—**The load balancer within the Service Framework efficiently distributes software and hardware resources for data processing, reducing command contention and enhancing throughput while promoting equitable resource sharing.

Device-specific drivers can be Panther hardware drivers or a virtual driver, which is an algorithm library that supports all transformation operations, offering a cost-effective redundancy option.

The Service Framework, aided by device-specific drivers, monitors hardware for problems such as overheating and ECC errors to ensure smooth operation.

The SDK supports multiple applications and uses on-chip memory in Panther devices for faster operations, with APIs available for on-chip caching.

The SDK enables I/O virtualization in Panther using a physical function driver and a virtual function driver.

SDK Reference Applications

The SDK includes several applications to demonstrate the capabilities of the Panther storage accelerator.

Table 1: SDK Applications

Application	Kernel / User Space	Purpose
demo	Kernel and user space	Demonstrates functionality and performance.
sdemo	User space	Enables low-level control on transform operations for unit tests.
example	Kernel and user space	Provides developer training.

SDK Configuration

The SDK provides a high degree of configurability to set up the use cases and optimal resource utilization on a platform. This may include, but not limited to, command ring configuration, load balance algorithm selection, result-retrieval mode, temperature monitoring, firmware flashing options, data protection configuration, and enabling real-time verification. You can also configure different class of services using the SDK configuration files.

MaxLinear recommends that you derive some of these parameters by running the Performance Evaluation tool on your platform.

Performance Tuning Tools

The MaxLinear Performance Evaluation tool helps you configure the SDK and its applications with the key software and hardware configuration parameters required to get the optimal performance from the Panther device and the SDK based on your server configuration/topology, such as the number of CPU cores, NUMA architecture, and CPU clock frequency, as well as use case requirements such as the data block size.

MaxLinear recommends that you use this tool on an evaluation platform to determine the correct set of performance-determining parameters and run the benchmarking tests.

Programming Model

The storage acceleration SDK APIs are called Raw Acceleration APIs. An acceleration session is initiated to submit source data to the SDK for transformation operations and retrieve the transformed data from the Panther device or ASL. You can configure a raw acceleration session to perform multiple data transformation tasks, such as NVMe PI processing, compression, encryption, or hashing in a single command, although all commands in a session must use the same algorithms.

To create a session, you must specify the class-of-service attributes, options for on-chip memory usage, and consistent operation parameters (for example, encryption keys or initial vectors). A session functions as a command processing context, where commands are submitted with source and destination buffer locations. Once the operation is complete, the transformed data is written to the destination buffers.

Commands can be submitted synchronously or asynchronously, and this submission mode must be specified when opening a session.

Using the Raw Acceleration API is simple as it only requires three main steps and the use of three APIs.

The basic processing steps for using the Raw Acceleration API are as follows.

Note: The code in this section is provided for illustrative purposes only and is intended to describe the programming model. It is not functional code for actual implementation.

Synchronous mode of execution

1. Open a session for the synchronous mode of command submission.

```
DRE_rawSessHandle sessId = 0;
DRE_status ret = 0;
DRE_COSType cos = DRE_COS_TYPE_ASSURED_FWD;
DRE_rawSessCompParam cmpParam;
DRE_compAlgoParam cmpAlgParam;

/* set compression parameters */
cmpParam.compAlgo = DRE_DEFLATE;
cmpParam.compAlgoParam = &cmpAlgParam;
cmpAlgParam.deflate.winSize=DRE_DEFLATE_WIN_SIZE_32KB;
```

```
ret = DRE_rawSessOpen(&sessId,      // Handle to the session instance
                      cos,          // Class of service
                      DRE_TRUE,     // Encode direction
                      DRE_FALSE,    // Use on-chip memory to cache command metadata
                      0,            // Flags for the session being opened
                      NULL,         // Callback function-used for async mode of submission
                      &cmpParam,    // Parameters for compression algorithm
                      NULL,         // Parameters for padding
                      NULL,         // Parameters for encryption
                      NULL,         // Parameters for hash algorithms
                      NULL,         // Parameters for NVME PI
                      0);           // Id of the device
if (DRE_IS_RESULT_ERR(ret))
{
    DRE_osalPrint(DRE_DEBUG_LEVEL_ERR,"Failed to open encode session, error code: 0x%08x\n",
    ret);
    exit(-1);
}
```

2. Submit one or more commands with data to the session and retrieve the result when the submitted command(s) is/are completed.

```
DRE_dataDesc srcBuf = {0};
DRE_dataDesc dstBuf = {0};
DRE_rawSyncOpData opData;
DRE_u32b dstLen, inLen, outLen;

/* src  */
srcBuf.ptr = inBuff;
inLen = COMPRESS_BUFF_SIZE;
srcBuf.len = inLen;
/* dst  */
dstBuf.ptr = outBuff;
outLen = COMPRESSED_BUFF_SIZE;
dstBuf.len = outLen;
/* opData  */
opData.dstLen = &dstLen;

ret = DRE_rawSessSubmitSync(sessId,
                            &opData,
                            &srcBuf,
                            1,
                            &dstBuf,
                            1,
                            NULL);

if (DRE_IS_RESULT_ERR(ret))
{
    DRE_osalPrint(DRE_DEBUG_LEVEL_ERR,"Command submission fails, error code: 0x%08x\n", ret);
    exit(-1);
}
```

3. Once the data transformation operations are complete, close the session and the session will no longer be used for additional transform commands.

```
DRE_rawSessClose(sessId);
```

Asynchronous mode of execution

In asynchronous mode, when opening a session, you must indicate that this session is used for the asynchronous mode of command submission by passing the callback function pointer. The callback function will be executed once the command has been submitted.

A unique cookie can be passed as an argument in each asynchronous command submission API call. When each command is completed, the SDK executes the callback function, providing the corresponding cookie as an argument. This enables the callback to access contextual information about the command when it runs, enabling it to perform its task more efficiently without relying on global variables.

1. Define your callback function that will be executed at the end of every command passing the callback cookie passed later in asynchronous API submission calls.

```
DRE_rawSessHandle sessId = 0;
DRE_status ret = 0;
DRE_COSType cos = DRE_COS_TYPE_ASSURED_FWD;
DRE_rawSessCompParam cmpParam;
DRE_compAlgoParam cmpAlgParam;

/* set compression parameters */
cmpParam.compAlgo = DRE_DEFLATE;
cmpParam.compAlgoParam = &cmpAlgParam;
cmpAlgParam.deflate.winSize=DRE_DEFLATE_WIN_SIZE_32KB;

/* Callback function for the async commands */
static inline DRE_status DRE_exCallback( DRE_sessHandle sessId,
                                         DRE_u32b       dstLen,
                                         DRE_u08b       nextHeader,
                                         DRE_status     status,
                                         DRE_uptr       cbID        // Cookie passed to callback
                                        )
{
   DRE_ex_correlator *myInfo = (DRE_ex_correlator *)cbID;
   if (myInfo)
   {
      myInfo->status = status;
      myInfo->dstLen = dstLen;
      DRE_osalSemRelease(myInfo->sem);
   }
   return DRE_OK;
}
```

```
 ret = DRE_rawSessOpen(&sessId,          // Handle to the session instance
                       cos,              // Class of service
                       DRE_TRUE,         // Encode direction
                       DRE_FALSE,        // Use on-chip memory to cache command metadata
                       0,                // Flags for the session being opened
                       DRE_exCallback,   // Callback function-used for async mode of submission
                       &cmpParam,        // Parameters for compression algorithm
                       NULL,             // Parameters for padding
                       NULL,             // Parameters for encryption
                       NULL,             // Parameters for hash algorithms
                       NULL,             // Parameters for NVME PI
                       0);               // Id of the device

if (DRE_IS_RESULT_ERR(ret))
{
    DRE_osalPrint(DRE_DEBUG_LEVEL_ERR,"Failed to open encode session, error code: 0x%08x\n",
    ret);
    exit(-1);
}
```

2. Submit one or more commands with data to the session and retrieve the result when the submitted command(s) is/are completed.
   Note: In the case of asynchronous command submission, the callback function is executed by the SDK once the command has been completed. Typically, many commands are submitted after a session is created.

```
DRE_dataDesc srcBuf = {0};
DRE_dataDesc dstBuf = {0};
DRE_rawSyncOpData opData;
DRE_u32b dstLen, inLen, outLen;
DRE_ex_correlator  usrInfo = {0};

/* src  */
srcBuf.ptr = inBuff;
inLen = COMPRESS_BUFF_SIZE;
srcBuf.len = inLen;
/* dst  */
dstBuf.ptr = outBuff;
outLen = COMPRESSED_BUFF_SIZE;
dstBuf.len = outLen;
/* opData  */
opData.dstLen = &dstLen;

/* Create a semaphore that' unlocked after execution of callback.
   So that command submission thread can get a notification waiting
   on completion of this command.
*/

DRE_osalMemset(&usrInfo, 0, sizeof(DRE_ex_correlator));
DRE_osalSemCreate(&usrInfo.sem, 0, 1, (DRE_u08b*)"deflate sem"); // Application can wait on this
                                                                  semaphore to get notification
```

```
ret = DRE_rawSessSubmitAsync( sessId,
                              &opData,
                              &srcBuf,
                              1,
                              &dstBuf,
                              NULL,
                              (DRE_uptr)&usrInfo);

if (DRE_IS_RESULT_ERR(ret))
{
    DRE_osalPrint(DRE_DEBUG_LEVEL_ERR,"Command submission fails, error code: 0x%08x\n", ret);
    exit(-1);
}
```

3. Once the data transformation operations are complete, close the session and the session will no longer be used for additional transform commands.

```
DRE_rawSessClose(sessId);
```

For more information on using the Raw Acceleration API, refer to the *MxL890x Raw Acceleration Application Program Interface (API) Reference Guide* (200AG).

Performance

MaxLinear benchmarks the performance of the SDK on different platforms for different use cases and data frame sizes. Key performance metrics include throughput in different use cases, command execution latency, and CPU core usage. For more information on benchmarking, refer to the *MxL890x Linux Performance Application Note* (294AN) and *MxL890x FreeBSD Performance Application Note* (297AN).

Supported Platforms

Table 2: Supported Platforms

Operating Systems	Platforms
CentOS	Intel and AMD CPUs
Rocky Linux	Intel and AMD CPUs
CentOS Stream	Intel and AMD CPUs
Ubuntu	Intel and AMD CPUs
RedHat	Intel and AMD CPUs
Anolis	Intel and AMD CPUs
FreeBSD 12.x, 13.x, and 14.x	Intel and AMD CPUs

Conclusion

MaxLinear's Panther storage accelerators address the increasing demand for secure, efficient data storage by delivering extensive data reduction, encryption, and deduplication capabilities, achieving an industry-leading data reduction ratio while reducing capital expenditures.

The storage acceleration SDK simplifies hardware and software integration, offering:

- Unified APIs for seamless implementation and hardware offloading.
- One-time integration with existing systems.
- Support for multiple transformation operations ensuring consistent performance.
- High concurrency and throughput for low-latency applications.
- Minimal CPU overhead for scalability.
- Kernel and user-space APIs for versatile integration.
- Flexible command submission with both synchronous and asynchronous modes.
- NUMA awareness for optimized memory access.
- Intelligent load balancing for fair resource utilization.

The SDK also includes configuration tools, performance tuning resources, and reference applications to help developers. Optimized for Linux and FreeBSD on AMD and Intel platforms, it improves performance and efficiency. Comprehensive documentation and regular updates based on user feedback further support developers, driving innovation and operational efficiency in data storage solutions.

References

- *MxL890x Software Development Kit (SDK) Getting Started User Guide* (210-CUG).
- *MxL890x Software Development Kit User Guide* (209UG).
- *MxL890x Raw Acceleration Application Program Interface (API) Reference Guide* (200AG).
- *MxL890x Storage Accelerator User Guide* (204-UG).
- *MxL890x Performance Evaluation Tool User Guide* (219UG).
- *MxL890x Linux Performance Application Note* (294AN).
- *MxL890x Linux Performance Tuning Application Note* (298AN).
- *MxL890x Software Development Kit Linux Release Notes* (202-CRN).
- *MxL890x FreeBSD Performance Application Note* (297AN).
- *MxL890x FreeBSD Performance Tuning Application Note* (299AN).
- *MxL890x Software Development Kit FreeBSD Release Notes* (209-CRN).

MaxLinear, Inc.
5966 La Place Court, Suite 100
Carlsbad, CA 92008
Tel.:+1 (760) 692-0711
Fax: +1 (760) 444-8598
www.maxlinear.com

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment, representation, or warranty by MaxLinear, Inc. or any of its affiliates (collectively, "MaxLinear"). MaxLinear assumes no responsibility or liability for any errors, omissions, or inaccuracies that may appear in the informational content contained in this document.

Reproduction, distribution, modification, or creation of derivative works, in part or in whole, without the express prior written consent of MaxLinear is prohibited. MaxLinear, the MaxLinear logo, and any other MaxLinear trademarks (including but not limited to MxL, Full-Spectrum Capture, FSC, AirPHY, Puma, AnyWAN, VectorBoost, MXL WARE, and Panther) are all property of MaxLinear and/or its subsidiaries in the U.S.A. and other countries. All rights reserved. All third-party marks and logos are trademarks™ or registered® trademarks of their respective holders/owners.

© 2026 MaxLinear, Inc. All rights reserved.