# NASA and Blue Origin Collaborative Assessment of Precision Landing Algorithms and Computing

David Rutishauser[1]
*NASA Johnson Space Center, Houston, Texas, 77058, United States*

Ray Ramadorai[2]
*Blue Origin, Kent, WA 98032, United States*

John Prothro[3]
*Avenue Technologies and Commodities Inc., Ponca City, Oklahoma, 74604, United States*

Thadeus Fleming[4]
*Odyssey Space Research, Houston, Texas, 77058, United States*

Peter Fidelman[5]
*Blue Origin, Kent, WA 98032, United States*

**NASA's Safe and Precise Landing Integrated Capabilities Evolution (SPLICE) project is developing sensor, algorithm, and compute technologies for precision landing and hazard avoidance. These technologies are being tested as an integrated Precision Landing and Hazard Avoidance (PL&HA) system on Blue Origin's New Shephard suborbital vehicle. A key goal for the computing element of this technology development is to characterize the performance of the SPLICE software workloads on the project's Descent and Landing Computer (DLC). The DLC is a multi-core processor designed as a surrogate for NASA's High-Performance Space Computer (HPSC). Measurements of the SPLICE workload performance on the DLC provides NASA insight on how PL&HA capabilities will perform on the HPSC, and guidance on how the SPLICE algorithms can be implemented to best utilize the DLC platform. This insight can also be used to derive requirements to guide trade studies on candidate computing architectures, for use on platforms like Blue Moon. NASA and Blue Origin are collaborating under an agreement to pursue this mutual benefit. Performance metrics collected are based on measurement of common compute resources such as percentage used of memory bandwidth, I/O utilization, interrupt latency, and kernel vs. user space code residency. Where possible existing performance counters and metrics that are part of the operating system kernel are used. As the design has a significant FPGA component, performance counters are identified and instantiated in the fabric to measure DMA performance and interface metrics. Collection of metrics is performed on the DLC with a representative workload that simulates a full landing cycle of the Blue Origin New Shepard vehicle. Consideration is given to the other compute implementations and whether they can run SPLICE algorithms at the same rate and with the same latency as the DLC. One option being considered is the use of a RISC-V soft core instantiated in a radiation resilient FPGA fabric such as the Xilinx KU60. Select algorithms from the SPLICE code will be run for comparison with the DLC. This paper describes how the DLC is instrumented to collect performance measurements of the SPLICE workloads, preliminary results from these measurements, and their implications**

[1] SPLICE Avionics Lead, Avionic Systems Division.
[2] Principal Technologist Avionics.
[3] Software Engineer, Avionic Systems Division, NASA JSC.
[4] Software Engineer, Aeroscience and Flight Mechanics Division, NASA JSC.
[5] Software Engineer, Advanced Development Programs.

**on SPLICE algorithm implementation. The results of experimentation to derive candidate requirements for architecture trades on a PL&HA computing system are also presented.**

## I. Introduction

The High-Performance Space Computer (HPSC) project is a NASA agency initiative to provide a high-performance multi-core processing capability for space missions [1]. Multicore processing or distributed processing will be needed to meet the computational demands of future space missions, specifically for the Precision Landing and Hazard Avoidance (PL&HA) capability that the Safe and Precise Landing, Integrated Capabilities Evolution (SPLICE) project is developing [2]. For the test flights on the New Shephard vehicle, NASA has developed a Descent and Landing Computer (DLC) as the compute platform to integrate the SPLICE technologies under test. The DLC and its interfaces are shown in Figure 1, and an overview of the DLC is provided in [3]. The processor used is a Xilinx Ultrascale+ Multi-Processing System on a Chip (MPSoC) [4]. The MPSoC has four ARM A53 processors. The HPSC uses two of these ARM quad cores interfaced with a coherent cache to provide an eight-core system. The similarities in processing architecture allow the MPSoC to be used as a surrogate for the HPSC. Future SPLICE tests plan to replace the surrogate MPSoC SBC with the HPSC SBC. The DLC MPSoC is hosted on a VPX standard single board computer developed by NASA. Another NASA developed Single-Board Computer (SBC) hosting a Xilinx Kintex FPGA provides the interfaces to the SPLICE sensor suite. Currently all the SPLICE algorithms and flight software run on the MPSoC A53 quad core. The SPLICE software uses the Core Flight System (cFS) framework, also discussed in [3]. Algorithm technologies that will run on the DLC are an extended Kalman Filter guidance, Dual Quaternion Guidance (DQG), and Terrain Relative Navigation (TRN). A future capability that will be added is Hazard Detection and Avoidance (HDA). Traditionally, the guidance and navigation algorithms are not developed for a specific target flight processor, or for parallel execution, and this is consistent with the approach for the algorithms in the SPLICE project. Measurements of how these workloads use shared system resources, such as cache and RAM memory and processor utilization, will be valuable for several reasons. (1) The measurements identify constrained shared resources that may be causing a performance issue, and system parameters or code changes can be guided and improved with this data. (2) The measurements provide insight as to how the SPLICE workloads will perform on the HPSC when available. And (3) the measurements can be used to derive architecture specifications for computing platform trades such as Blue Origin is performing for their Blue Moon lunar lander vehicle.

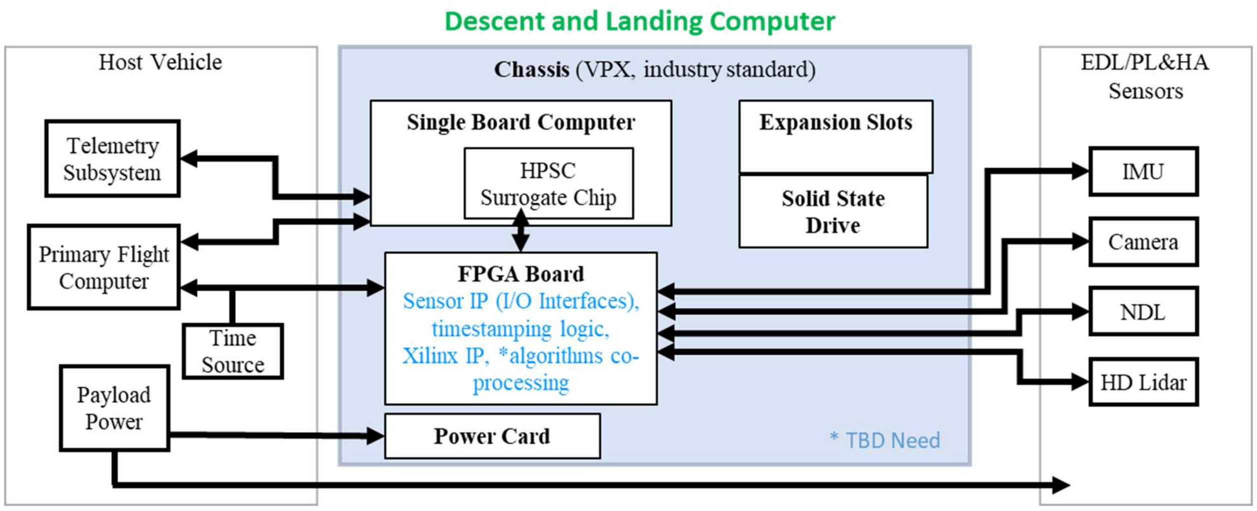


**Fig. 1 SPLICE Flight Test System Configuration.**

Performance testing may reveal that parallelization is needed to meet the control rates for a given algorithm. This is especially true of HDA which is expected to need significant compute resources. At a minimum, parallelization is required to efficiently utilize the DLC multicore platform. A manual parallelization of a single-thread code often requires a substantial effort. Part of this work is to investigate the effectiveness of achieving parallel execution with OpenMP directives, and compiler options to enable vectorization via the NEON Single-Instruction, Multiple Data (SIMD) co-processor available in the MPSoC processing core. Code that is written for parallel or vector compilation will benefit most from these enhancements, so the trade of how much effort is required in software modifications verses performance gain is desirable. Metrics of system performance are valuable for this type of trade study.

Blue Origin is investigating the use of a RISC-V soft processor instantiated in a radiation-tolerant FPGA as the processing core for their Blue Moon vehicle. The RISC-V core is open source, has options for co-processors and extensions to its instruction set, and tools for development and integration into FPGA vendor tools [5]. Blue Origin plans to support PL&HA capabilities in their chosen processor, so SPLICE system measurements of compute resource utilization are valuable for guiding the RISC-V development and trade study.

## A. SPLICE Workloads

High-level performance measurements are made of the integrated flight software system, and targeted measurements evaluate the performance of key SPLICE algorithm technologies. An overview of these algorithms follows.

### 1. *Navigation (NLP and NLU)*

Navigation consists of two cFS applications, NLP and NLU. NLP runs at 50 Hz and handles state propagation, and NLU runs at 5 Hz and calculates updates to covariance and filter states based on processed sensor measurements. Currently a 41x41 double-precision matrix is maintained for these algorithms; a future update will only carry non-zero values. Both NLP and NLU use a 1st order Taylor series-based expansion, with most operations performed on linear approximations of dynamics. Neither NLP nor NLU require disk access during execution, and the data logged from the algorithms is about 25 MB. Parallelization has not been examined for these algorithms and may not yield much of a performance improvement since much of these algorithms require sequential execution.

### 2. *Terrain Relative Navigation (TRN)*

TRN is a single cFS application that executes at 1 Hz. It uses TRN software developed by Draper Laboratory to match features in an image captured with an onboard camera with those stored in an onboard database. The database is saved as a binary file and loaded at startup. It is multiple megabytes in size and stored as a persistent global variable within the application. It does not access the disk during execution, and storage requirements are estimated at 15.5 GB for the imagery and 250 MB for other. Imagery is logged to disk for post-flight processing. Some parallelization is implemented in this code.

### 3. *Dual Quaternion Guidance (DQG)*

DQG is active during the final phases of flight (powered descent). It is a single cFS application currently executing at 0.2 Hz with a goal of 1 Hz. It utilizes a second order cone program to solve a non-linear optimization problem to determine optimal trajectory from powered descent through touchdown. DQG is not currently parallelized but may need parallel execution to meet the desired 1 Hz execution rate. DQG doesn't currently access the disk during execution but this may be needed in future versions. The storage requirement is currently estimated as 15 MB. The number of iterations DQG requires in the optimizer is non-deterministic.

## B. System Metrics

The performance parameters measured are described in this section. Figure 2 shows a block diagram of the DLC components, the functions allocated to each part of the architecture, and the measurements collected. The measurements were selected because they provide insight into how the SPLICE software utilizes the DLC compute resources. The metrics to be collected are also summarized in Table 1, along with attributes and notes on each measurement. A description of these measurements and how they are collected follows.

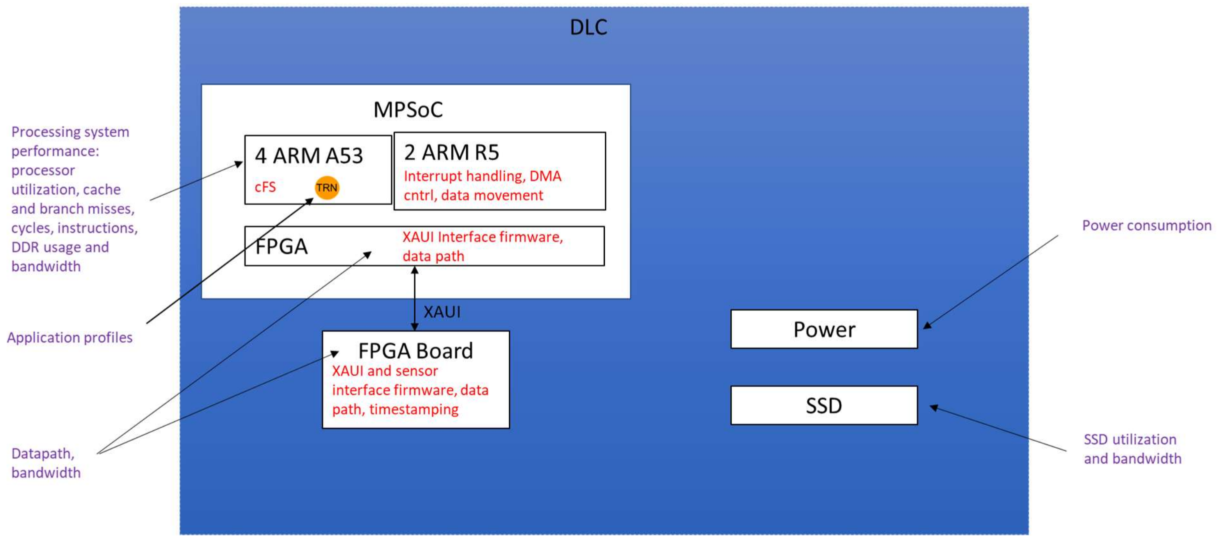

DLC
MPSoC
4 ARM A53
cFS
TRN
2 ARM R5
Interrupt handling, DMA cntrl, data movement
FPGA
XAUI Interface firmware, data path
XAUI
FPGA Board
XAUI and sensor interface firmware, data path, timestamping
Power
SSD
Processing system performance: processor utilization, cache and branch misses, cycles, instructions, DDR usage and bandwidth
Application profiles
Datapath, bandwidth
Power consumption
SSD utilization and bandwidth

**Fig. 2 DLC Structure, Function, and Measurements.**

| Measurement | Units | Frequency | HWIL sim | Flight |
|---|---|---|---|---|
| Processor utilization | % | 1 Hz | x | x |
| SSD usage | % | 1 Hz | x | |
| SSD bandwidth | Gbits/sec | Offline measurement | x | |
| DDR usage | % | 1 Hz | x | |
| DDR bandwidth | Gbits/sec | 1 Hz | x | |
| Cache misses | count | 1 Hz | x | x |
| Branch misses | count | 1 Hz | x | x |
| Instructions | count | 1 Hz | x | x |
| Cycles | count | 1 Hz | x | x |
| Application profiling | n/a | One-time measurement | x | |
| Application speedup | % | One-time measurement | x | x |
| High-speed serial link bandwidth | Mbits/sec | 156.25 MHz | x | x |

**Table 1. Performance Measurement Summary**

### *1. Processor utilization*

The processor utilization provides information on how the SPLICE workloads are balanced on the 4 ARM A53 MPSoC cores and the extent to which they are compute-bound. It also shows how much reserve in processing capacity remains for future SPLICE capabilities like hazard detection and avoidance. In the simulation, this data is collected with the *mpstat* utility, and in flight it is collected through the Linux *procfs* API. Most SPLICE algorithms are not currently written for parallel execution, but individual apps are assigned their own thread in cFS, so the Linux scheduler distributes them among the processors available. Some algorithm parallelization may be implemented by the second test flight.

### 2. *Solid State Drive (SSD) Usage and Bandwidth*

SSD usage is a sizing metric for in-flight data collection. The current usage is 16 GB. No SPLICE algorithms access the disk during computation, but a profile of the disk usage over the flight time is examined. This is collected in the simulation configuration only using *iotop*.

### 3. *DDR Usage and Bandwidth*

DDR usage is a sizing metric. It is also sampled during runtime to see a usage profile. This is collected in the simulation configuration only using *free*. DDR bandwidth provides insight on the extent the SPLICE workloads are memory bound. Currently we have not identified a means to measure this bandwidth.

### 4. *Cache misses*

Cache misses provide insight on the working set size of the SPLICE applications and how adequately the DLC caches are sized for them. In the simulation, this data is collected with the perf command line utility, and in flight it is collected with an API to the perf utility.

### 5. *Branch Misses*

Branch misses provide insight on how efficiently the SPLICE applications are executing on the A53 processors as a result of successful branch predictions. In the simulation, this data is collected with the perf command line utility, and in flight it is collected with an API to the perf utility.

### 6. *Instructions*

Instruction count provides insight on the amount of work an application is doing at a given time. Converting this metric to a rate of instructions per second, or cycle, provides an execution speed measurement. In the simulation, this data is collected with the perf command line utility, and in flight it is collected with an API to the perf utility.

### 7. *Cycles*

Cycle count also provides insight on the amount of work an application is doing at a given time. In the simulation, this data is collected with the perf command line utility, and in flight it is collected with an API to the perf utility.

### 8. *Application Speedup*

Application speedup is computed from the cycle measurement difference before and after a performance enhancement is made to the code when running the same mission profile.

### 9. *Application Profiling*

The Linux perf utility is used to generate call graphs that report the amount of execution time a SPLICE application spends in each of its constituent parts. This is performed for an application as it executes in the integrated flight software, in the HWIL configuration only.

### 10. *High-Speed Serial Link Bandwidth*

The DLC sensor input stream bandwidth is known, but the data path can pause if a buffer fills, or the high-speed serial link between the FPGA and MPSoC boards loses sync. Counters will be inserted to track number of paused clock cycles of the data stream. With this information, actual bandwidth will be computed. This measurement will provide insight as to the effectiveness of the DLC data path implementation. This measurement is planned for future work.

## II. Experimental Design

System metrics presented in this paper are measured in the SPLICE Hardware-in-the-Loop (HWIL) simulation. A subset of these measurements will also be collected during a second test flight planned for 2021. An overview of the HWIL capability for SPLICE is in [3]. The HWIL simulation uses a nominal profile representative of the suborbital flight tests flown and planned of the SPLICE hardware on Blue Origin's New Shephard vehicle. The SPLICE HWIL configuration at the time of this writing did not have a simulated or real TRN camera interface to the DLC available. To stimulate the TRN algorithm, recorded camera images were read from the DLC's SSD and provided to the software application that consumes the real camera data in flight. In the simulation, the Linux *perf stat* command, the *perf record* command and the *perf report* command, as well as other Linux command line utilities are used to collect performance data. The following procedure is used for command line measurements:

1. Start the SPLICE flight software and take note of the relevant thread id for the application/dynamic shared object (DSO) to be examined. The thread id can be obtained using the Linux *ps* command.
2. Start simulation. Verify that the simulation runs to completion successfully. Verify that the console does annunciate the different modes.
3. Repeat steps 1 and 2 varying the measurement of interest, on the process or thread of interest. The preference is to execute only one collection method and utility at a time to reduce the overhead and interference induced by the data collection itself.

A measurement application is also used within the SPLICE flight software to collect metrics using an Application Programmers Interface (API) to the performance counters for *perf*. The *perf* API used is the *perf_event_open*() API. This call allows for duplicating in a customized application, every detail of any given usage of the Linux *perf stat* command line utility. The measurements are constrained to those in the *perf* event list [6], and the measurements chosen are shown in Table 1. These are the measurements that will be collected in flight. An advantage of the software measurement application is that it can be triggered by the same cFS scheduler event that "wakes up" the process of interest to execute at its designed frequency. This prevents significant measurement time when the process is not active, avoiding error in the measurement.

## III. Results

Figure 3 shows the results of running the *mpstat –P ALL 1* at the command line during a HWIL simulation run. The utilization of all 4 CPUs is shown for the applications (*%usr*). The lines that go to 100% utilization are an overlay showing flight software mode transitions. Immediately apparent is that most of the times of high utilization are on a single processor, confirming that the current level of parallelization of the SPLICE software is at the coarse application level. Finer grained parallelization within each application would provide the opportunity for a more balanced load across the processors. Further evidence of this is shown in Figure 4, where the same data is plotted as a stacked bar graph. The area is zoomed in and shows the CPU utilization (first CPU2 and then CPU0) increases to 80% each while total CPU utilization stays the same (~150-200%). This suggests some heavy single-threaded workloads that start near the end of the run.

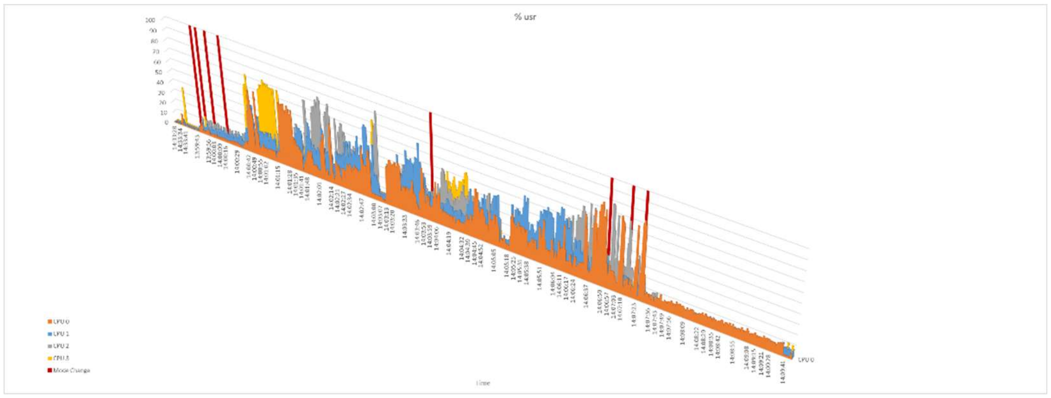


**Fig. 3 Top-level SPLICE software processor utilization.**

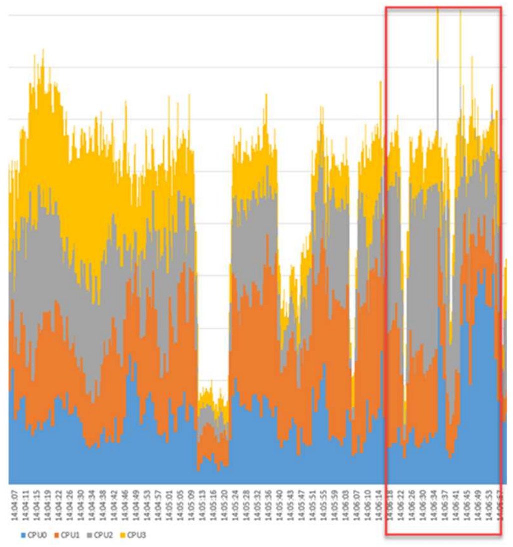


**Fig. 4 Top-level utilization, high single-threaded allocation.**

Figure 5 shows the average CPU utilization for all categories. As shown, the average utilization is low, which indicates processing potential for future planned upgrades such as hazard detection. This assumes the work can be distributed such that the worst-case utilization is less than 100%. Figure 6 is the data from executing an *iotop -obt* command and plotting the SSD read and write bandwidth. Note that this plot shows additional disk reads that are an artifact of the simulation configuration, described in the Experimental Design section. The spikes in disk writes that occur roughly every 5-10 seconds are due to saving data from DQG, which runs at that interval.

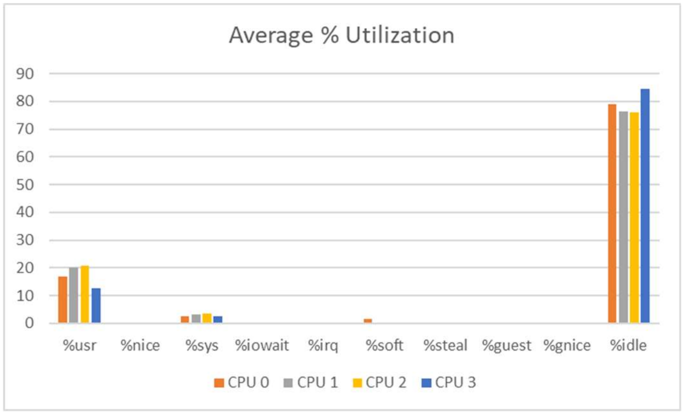


**Fig. 5 Average top-level SPLICE software processor utilization.**

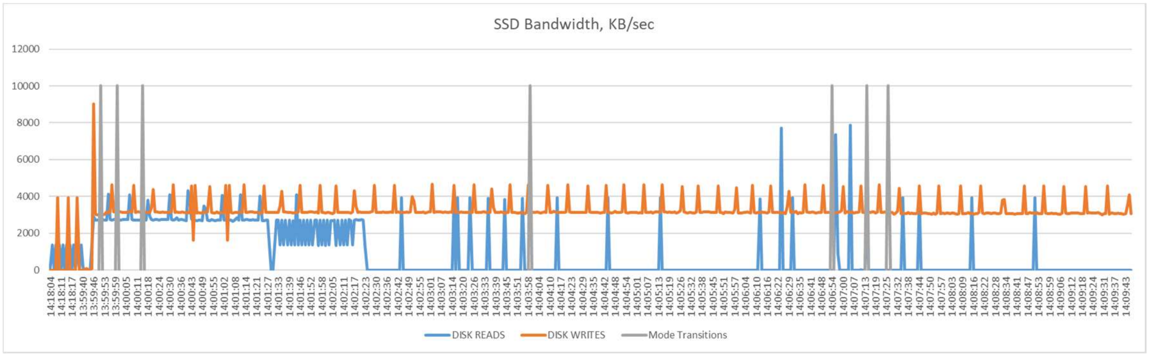


**Fig. 6 Top-level SSD bandwidth.**

Figure 7 shows the top-level cache misses for the SPLICE software. This data is produced with the command *perf stat –e cache-misses,cache-references –p 1913 –I 1000.* The trend shows a typical cache behavior of decreasing misses as the more used cache lines are loaded. The flight mode transition is roughly in the middle of the plot, and from the small number of misses during flight the data suggests the DLC's caches are sized adequately for the SPLICE workload.

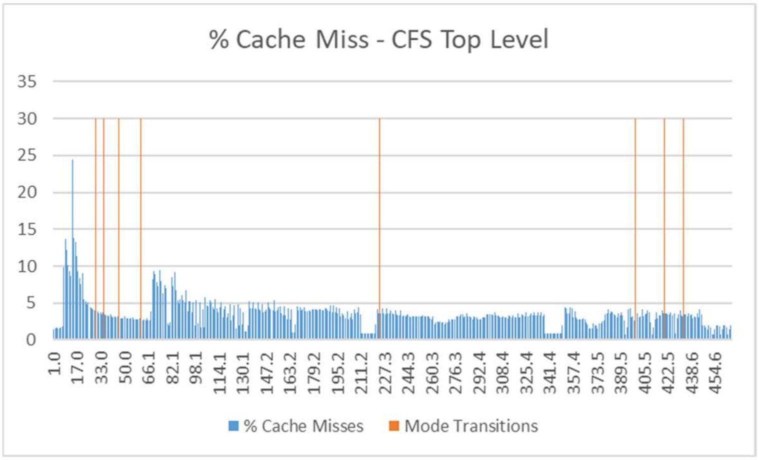


**Fig. 7 Top-level cache misses.**

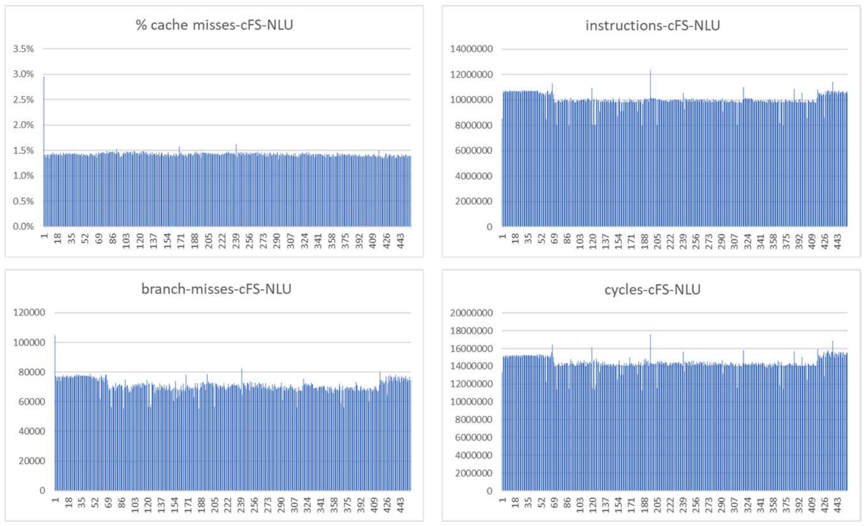
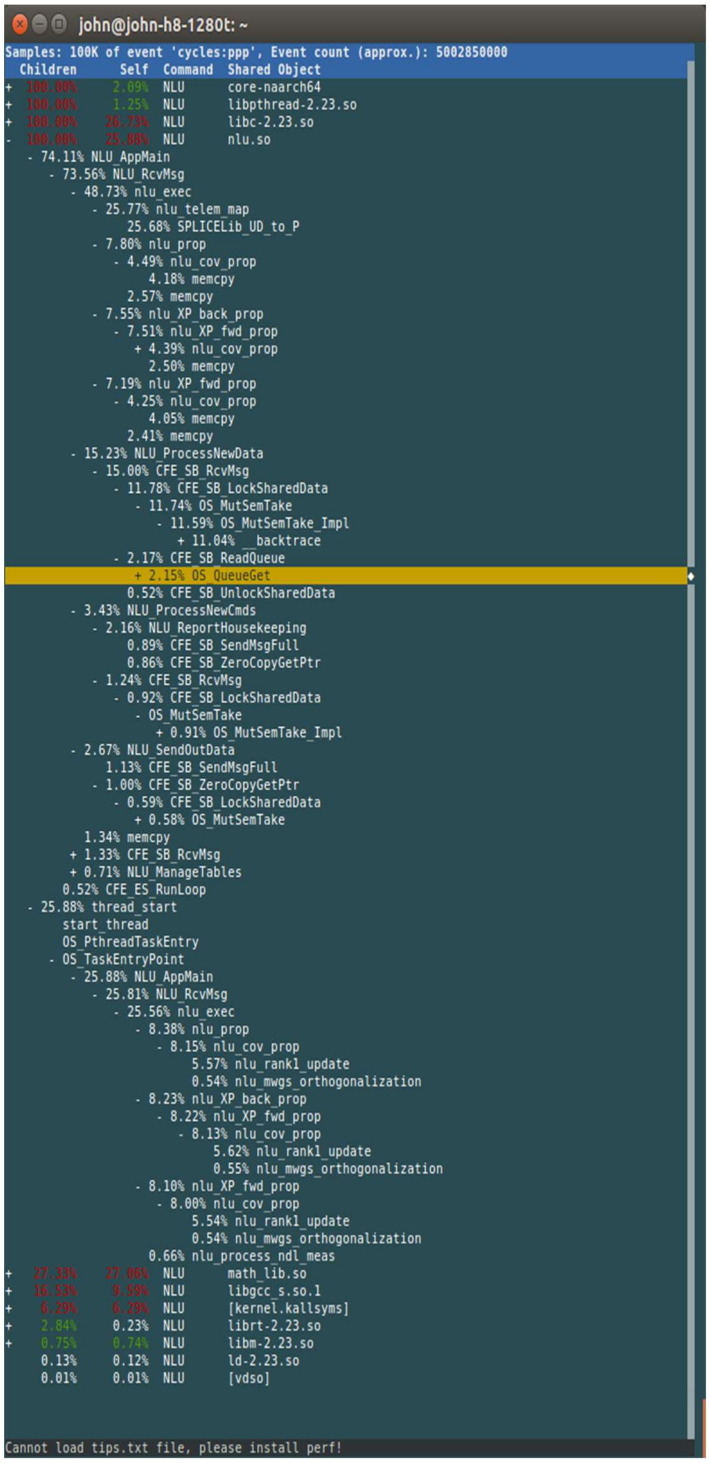


**Fig. 8 CFS measurement application metrics for NLU.**

Figure 8 shows metrics collected for the NLU application, using the cFS measurement application. Note that the measurement application targets the one thread ID, providing specific measurements of one application of interest as it executes in the integrated software environment. NLU does not show much variation in these metrics during execution. The low percentage of cache misses reflects that this measurement is a small contributor to the top level cFS process measurement of the same metric. The measurements of instructions and cycles yield an execution rate of about 0.7 instructions per cycle. The ARM A53 is an in order partial dual-issue machine. With its two-issue pipeline, instructions per cycle could be as high as 2 if the workload is appropriate [7]. This metric can be used post performance-enhancing code changes to measure the effect. Some of these optimizations may be pursued prior to the second flight test. Figure 9 is a perf call graph for the same application, providing a profile of the time spent in each subpart of this application. This is useful for the algorithm implementation team to identify the areas of the application that would have the largest impact on the overall performance if their execution time was reduced. For example, a substantial amount of time is spent inside of memcpy, suggesting that the team may be able to achieve noticeable speed increases by accessing data in place rather than copying it.



```
john@john-h8-1280t: ~
Samples: 100K of event 'cycles:ppp', Event count (approx.): 5002850000
  Children      Self  Command  Shared Object
+  100.00%     2.09%  NLU      core-naarch64
+  100.00%     1.25%  NLU      libpthread-2.23.so
+  100.00%    26.73%  NLU      libc-2.23.so
-  100.00%    25.88%  NLU      nlu.so
   - 74.11% NLU_AppMain
      - 73.56% NLU_RcvMsg
         - 48.73% nlu_exec
            - 25.77% nlu_telem_map
                 25.68% SPLICELib_UD_to_P
            - 7.80% nlu_prop
               - 4.49% nlu_cov_prop
                    4.18% memcpy
                 2.57% memcpy
            - 7.55% nlu_XP_back_prop
               - 7.51% nlu_XP_fwd_prop
                  + 4.39% nlu_cov_prop
                    2.50% memcpy
            - 7.19% nlu_XP_fwd_prop
               - 4.25% nlu_cov_prop
                    4.05% memcpy
                 2.41% memcpy
         - 15.23% NLU_ProcessNewData
            - 15.00% CFE_SB_RcvMsg
               - 11.78% CFE_SB_LockSharedData
                  - 11.74% OS_MutSemTake
                     - 11.59% OS_MutSemTake_Impl
                        + 11.04% __backtrace
               - 2.17% CFE_SB_ReadQueue
                  + 2.15% OS_QueueGet
                 0.52% CFE_SB_UnlockSharedData
         - 3.43% NLU_ProcessNewCmds
            - 2.16% NLU_ReportHousekeeping
                 0.89% CFE_SB_SendMsgFull
                 0.86% CFE_SB_ZeroCopyGetPtr
            - 1.24% CFE_SB_RcvMsg
               - 0.92% CFE_SB_LockSharedData
                  - OS_MutSemTake
                     + 0.91% OS_MutSemTake_Impl
         - 2.67% NLU_SendOutData
              1.13% CFE_SB_SendMsgFull
            - 1.00% CFE_SB_ZeroCopyGetPtr
               - 0.59% CFE_SB_LockSharedData
                  + 0.58% OS_MutSemTake
           1.34% memcpy
         + 1.33% CFE_SB_RcvMsg
         + 0.71% NLU_ManageTables
        0.52% CFE_ES_RunLoop
   - 25.88% thread_start
        start_thread
        OS_PthreadTaskEntry
      - OS_TaskEntryPoint
         - 25.88% NLU_AppMain
            - 25.81% NLU_RcvMsg
               - 25.56% nlu_exec
                  - 8.38% nlu_prop
                     - 8.15% nlu_cov_prop
                          5.57% nlu_rank1_update
                          0.54% nlu_mwgs_orthogonalization
                  - 8.23% nlu_XP_back_prop
                     - 8.22% nlu_XP_fwd_prop
                        - 8.13% nlu_cov_prop
                             5.62% nlu_rank1_update
                             0.55% nlu_mwgs_orthogonalization
                  - 8.10% nlu_XP_fwd_prop
                     - 8.00% nlu_cov_prop
                          5.54% nlu_rank1_update
                          0.54% nlu_mwgs_orthogonalization
                    0.66% nlu_process_ndl_meas
+   27.33%    27.06%  NLU      math_lib.so
+   16.53%     9.59%  NLU      libgcc_s.so.1
+    6.29%     6.29%  NLU      [kernel.kallsyms]
+    2.84%     0.23%  NLU      librt-2.23.so
+    0.75%     0.74%  NLU      libm-2.23.so
     0.13%     0.12%  NLU      ld-2.23.so
     0.01%     0.01%  NLU      [vdso]

Cannot load tips.txt file, please install perf!
```

**Fig. 9  Perf call graph for NLU application.**

A concern for flight test is the overhead that running the cFS measurement application introduces. At a minimum, the overhead changes the performance measurements. The worst case is the overhead changes the desired functionality of the SPLICE software. For example, the extra processing required to process the measurements causes SPLICE applications to not complete operation on schedule. To characterize this overhead, a top-level processor utilization was collected while both running and not running a cFS measurement application. The difference in the two runs is plotted in Figure 10. As shown, there is significant overhead measured for at least one of the processors. No errors in the operation of the SPLICE software were observed, so this is likely limited to a consideration in the accuracy of the performance measurements.

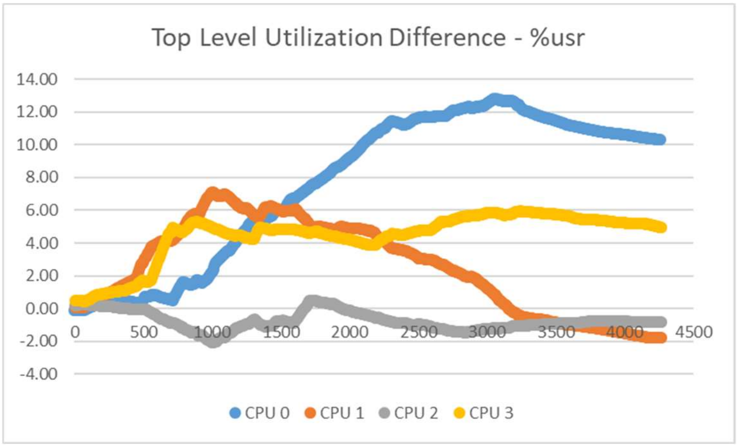


**Fig. 10 Processor utilization increase due to cFS measurement application.**

## IV. Conclusions and Future Work

The results presented here represent the beginning of efforts to analyze valid measurements from our DLC instrumentation approach. Future work includes working with the algorithm and software designers to understand the implications of the data in more detail. We will use this information to guide performance enhancements and will have a baseline with this work to measure those enhancements against. Flight data will be collected on the second test planned for the coming year. Additional measurements we are considering are interrupts to the A53 and network performance. The DLC data path for sensor I/O is architected to prevent the A53 processors that are running the SPLICE software from being interrupted for sensor data. But other interrupts such as for telemetry will impact performance, and a characterization of all the impacts will produced a more complete performance assessment. Additional future work will be to test applications of OpenMP parallelization, manual parallelization, and vectorization of the SPLICE software with this measurement approach.

## Acknowledgements

The authors wish to acknowledge Matthew Fritz of Draper Labs for consultation on the specific SPLICE algorithm technologies. This work was performed under Space Act Agreement # ACO-2019-051 between NASA and Blue Origin.

## References

[1] Richard Doyle, Raphael Some, Wesley Powell, Gabriel Mounce, Montgomery Goforth, Stephen Horan, and Michael Lowry, “High Performance Spaceflight Computing; Next-Generation Space Processor: A Joint Investment of NASA and AFRL,” International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS 2014), Montreal, Canada, June 2014.

[2] J.M. Carson, M.M. Munk, et al., "The SPLICE Project: Continuing NASA Development of GN&C Technologies for Safe and Precise Landing”, Proceedings of the AIAA 2019 SciTech/GN&C Conference, San Diego, CA, January 2019.

[3] D.K. Rutishauser and T. Tse, “Hardware-in-the-Loop Testing for Suborbital Flights of the Safe and Precise Landing Integrated Capabilities Evolution (SPLICE) Project Technologies”,, Proceedings of the AIAA 2020 SciTech Conference, Orlando, FL, January 2020.

[4] Xilinx MPSoC Web site: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[5] Chipyard Web site: https://chipyard.readthedocs.io/en/latest/Tools/index.html

[6] Linux Man page https://linux.die.net/man/1/perf-list

[7] Arm Developer Web site: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53