In Silico Methods for Space System Analysis: Optical Link Coding Performance and Lunar Terrain Masks

Carlyn Lee*, Hua Xie†, and Charles H. Lee‡
NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA

Dmitry A. Lyakhov§ and Dominik L. Michels¶
King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia

As deep space links migrate toward higher frequency bands like K_a and optical, thorough trade-space exploration becomes increasingly valuable for designing reliable and efficient communications systems. In this contribution, we leveraged high-performance, concurrent simulations when the run-time complexity of simulation software overwhelms capabilities of ordinary desktop machines. The first part of this manuscript describes how to run error correcting code simulations concurrently on a high-performance supercomputer. The second part of this study describes a framework to produce azimuth and elevation terrain masks from imagery of the Lunar South Pole.

I. Nomenclature

E_b/N_0	=	bit energy to noise-spectral density ratio
K_s	=	signal power
K_b	=	average number of detected background photons in one slot
M	=	PPM order
K_s/M	=	signal photons/slot

Deep space optical communication suffers from presence of non-Gaussian and non-linear fading effects caused by pointing errors, atmospheric losses, optical turbulence, and other losses [1]. In this type of power constrained optical channel, we look to data-link layer protocols to aid in reliable communications links. In particular, a Pulse Position Modulation (PPM) is used for error correction and concatenated with a recursive accumulator, a simple short constraint length convolutional code, and bit interleaver [2]. This serially concatenated PPM, or SCPPM, was adopted as the standard for error-correction coding in the Consultative Committee for Space Data Systems (CCSDS).

On a modern desktop computer, calculating the operating regimes corresponding to bit error rates (BER) 10e-5 and lower with SCPPM simulation software requires days or weeks. Virtualization enabled our simulations to run in 24-hrs on ShaheenII, King Abdullah University of Science & Technology's (KAUST) 36-cabinet Cray XC40 supercomputer, and helped identify signal and noise characteristics reaching BER 10e-9. We explore SCPPM simulations of rate 2/3 coded 16-SCPPM and 128-SCPPM, and demonstrate how to use this insight for channel design based on Psyche optical comm predicts. Psyche will host the first deep space optical comm payload, DSOC. This approach can also be useful when exploring higher layer protocols, such as the network-layer Automatic Repeat Request (ARQ), which introduces a dimension of latency to the comm system design trade-space, in addition to traditional considerations of mass, power, and available spectrum.

Part two demonstrates how to develop a framework to produce azimuth and elevation terrain masks from imagery of the Lunar South Pole. By leveraging Shaheen II, this framework produced masks for 360 degrees views around each pixel in a grid of 850x1001 pixels, each representing a 20m resolution. In this manuscript, we discuss how to use a Jupyter notebook to assess parallel design considerations for this particular algorithm.

*Engineering Applications Software Engineer, Telecommunications Architectures, Mail Stop 238-420.
†Telecommunications Engineer, Telecommunications Architectures, Mail Stop 238-420.
‡Telecommunications Engineer, Telecommunications Architectures, Mail Stop 238-420.
§Postdoctoral Researcher, Computational Sciences Group.
¶Assistant Professor, Computational Sciences Group.

II. Introduction

A. Deep Space Optical Comm. Serially Concatenated Pulse Position Modulation

The development of error-control codes is attributed to advancements in deep-space exploration missions. Extreme loss of signal power over interplanetary distances and limited power and mass availability aboard space probes motivated development of SCPPM, an error-control coding technique described in [2]. In a power constrained optical channel, SCPPM adds redundancy to received data in order to recover messages from DSOC's faint laser signals. This data-link layer protocol ensures that data in a communications channel is received reliably by encoding the transmitted message such that it can be extracted correctly at the receiving end, even in the presence of noise. The software used in this study simulates the effect of noise on the communications channel, and implements the encoder and decoder operations.

In 2022 NASA will launch a Deep Space Optical Communication (DSOC) payload aboard the Psyche spacecraft. The flight-ready payload will use lasers in the near-infrared region of the electromagnetic spectrum. The ground data system is comprised of a ground laser receiver and transmitter that uses existing ground assets. Compared to conventional systems of comparable mass and power, DSOC is capable of delivering information at least 10 times faster. In this manuscript we analyse Psyche DSOC as a pedagogical example of our software simulation strategy.

For RF links, the coding channel performance is described by a BER curve, which is a function of SNR, E_b/N_o. However for DSOC, bit error rate is a function signal power (K_s) and background noise power (K_b). In our simulations of the coding channel, K_s and K_b correspond to the signal and noise part of the SNR measured by the photons per one cycle (slotwidth), or $10 * log_{10}(\#\ of\ photons/slot)$. The background noise is measured with the absence of a signal. Insight on the tail of the noise distribution demonstrates the transition from a signal dominant regime to a noise dominant regime, so we run simulations until asymptotic behavior is observed. This is computationally expensive and may take days or weeks, if at all, to complete. In order to explore the trade-space for a wide range of design tolerances, our simulations were left to run over long periods of time.

The projected supportable data rates for Psyche, K_s, K_b are shown in figure 1. These projections were generated by the Strategic Optical Link Tool (SOLT) using methods described in [3]. In our simulation we explore the noise figures, K_b, projected in the time-frame starting at the end of 2022 until beginning of 2023, and is highlighted in the graph of K_b shown in the top figure.

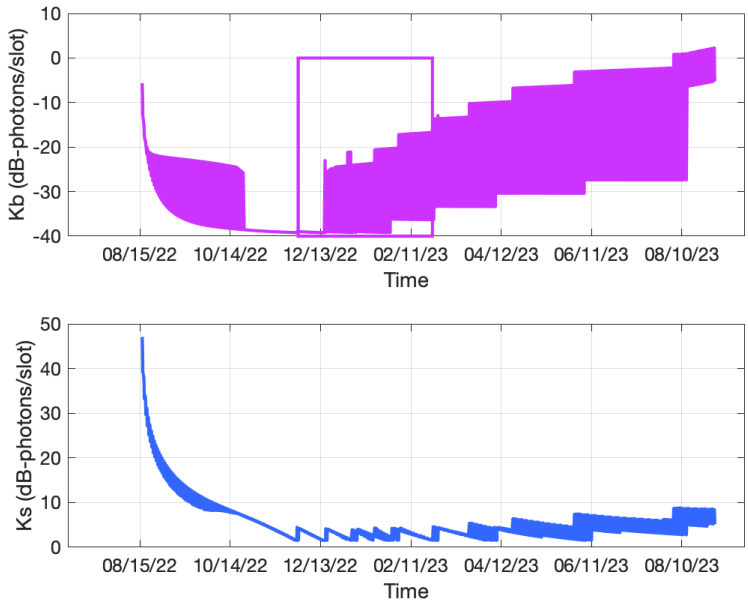

Fig. 1 Psyche DSOC projected supportable K_s and K_b.

Figure 2 shows projected supportable data rates, ranges, and Sun-Earth-Probe angles during the start of the Psyche mission. In this study we simulate coding channels using SOLT projections and varying configurations for the Psyche DSOC demo when the range of the spacecraft is between 0.25 AU and 1AU. SCPPM configurations $M = 16, 128$ and

$rates = 1/3, 1/2, 2/3$ were selected for this study based on SOLT recommendations. Simulations were also run for $M = 8, 16, 32, 64, 256$ but are not discussed in this manuscript.

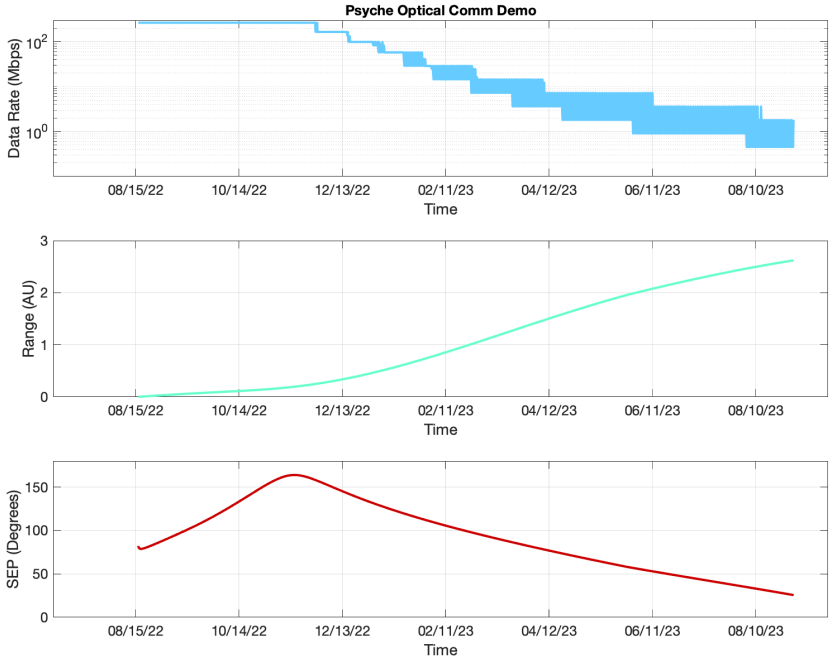

Fig. 2 Projections and supportable data rates for Psyche DSOC demo.

B. Lunar South Pole Terrain Masks

The main purpose of simulating the Lunar terrain mask is to determine line-of-sight visibility with another object (Sun, Earth, spacecraft, etc.) at a given location, taking into account the surrounding terrain. The terrain mask model helps to determine the communication coverage and the Sun's coverage of a surface element maneuvering on the surface of the Lunar South Pole [5].

1. Lunar Orbiter Laser Altimeter Data Set

The data are collected from the Lunar Orbiter Laser Altimeter (LOLA) on NASA's Lunar Reconnaissance Orbiter (LRO) [6] and are available from the LOLA Data Archive: `http://imbrium.mit.edu/BROWSE/LOLA_GDR/POLAR/SOUTH_POLE/`. For this work we are using polar stereographic projections at -80 degrees to the Lunar South Pole. The original data are gridded data records in a binary file, which are read in as 30,400x30,400 single precision floating point numbers. Each pixel represents a 20 meters resolution and are relative to a radius, $r_{moon} = 1737.4km$. For communications coverage analysis on the Lunar South Pole, we will assume a height, $P_h = 2m$, for a DSOC terminal mounted to a rover or any extravehicular activity, and a height, $P_h = 10m$, for a fixed station. Elevation masks were computed at every pixel this region at heights of 2m and 10m and spanning 150 km in each azimuthal direction.

2. Terrain Mask Algorithms

The terrain mask algorithm consists of a series of geometric transformations to different coordinate reference systems. Each pixel index pair (k_x, k_y) is transformed to planetocentric latitude and longitude, by first scaling them to the resolution, S, for image size , n, and applying the following transformations:

$$\begin{aligned} x &= (k_x + 1 - n/2 - .5) \cdot S \\ y &= (k_y - 1 - n/2 - .5) \cdot S \end{aligned} \tag{1}$$

$$P_{lat} = -90 + 180/\pi \cdot 2 \cdot arctan\left(0.5 \cdot \frac{\sqrt{x^2 + y^2}}{1737400}\right)$$
$$P_{lon} = arctan2(x, y) \cdot 180/\pi \tag{2}$$

The terrain altitude, P_{alt}, is the half the value at index location found in the mapping of (P_{lat}, P_{lon}) to corresponding (i, j) index pairs:

$$R = 2 \cdot r_{moon} \cdot tan((90 + P_{lat}) \cdot \pi/360) \tag{3}$$

$$\begin{aligned} i &= round(R \cdot sin(P_{lon} \cdot \pi/180)/S + n/2 + 1/2) - 1 \\ j &= round(R \cdot cos(P_{lon} \cdot \pi/180)/S + n/2 + 1/2) + 1 \end{aligned} \tag{4}$$

For each pixel in the image, the spherical coordinates, $(P_{lon}, P_{lat}, r_{moon} + P_{alt} + P_h)$, were transformed to Cartesian coordinates $\vec{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix}$, where

$$\begin{aligned} P_x &= (r_{moon} + P_{alt} + P_h) \cdot cos(P_{lat}) \cdot cos(P_{lon}) \\ P_y &= (r_{moon} + P_{alt} + P_h) \cdot cos(P_{lat}) \cdot sin(P_{lon}) \\ P_z &= (r_{moon} + P_{alt} + P_h) \cdot sin(P_{lat}) \end{aligned} \tag{5}$$

Using the normalized vector $\hat{P} = \frac{\vec{P}}{\left\|\vec{P}\right\|}$ and unit vector $\vec{N} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ apply the transformation to planetocentric coordinates:

$$\begin{aligned} \vec{P}_E &= \vec{N} \times \hat{P} \\ \vec{P}_N &= \hat{P} \times \vec{P}_E \end{aligned} \tag{6}$$

$$\tilde{Q} = \vec{P} + \vec{d}(cos(Az)\vec{P}_E + sin(Az)\vec{P}_N) \tag{7}$$

for all azimuthal degrees $Az = \{1, 2..360\}$ to generate all surrounding horizon masks for the array of distances, $\vec{d} = \left[d_1, d_2, \ldots, d_D\right]$, up to distance, d_D, separated by intervals determined by pixel resolution. Geometric transformations in eqs. 1-5 are applied on the observation points, $\tilde{Q}$. Each row in $\tilde{Q}$ are transformed to spherical coordinates using eq. 2 to find $(\tilde{Q}_{lon}, \tilde{Q}_{lat})$. Following this we apply eqs. 3-4 to find the index pair, $(\tilde{Q}_i, \tilde{Q}_j)$, in our image which corresponds to altitude information for $(\tilde{Q}_{lon}, \tilde{Q}_{lat}, \tilde{Q}_{alt} + r_{moon})$. Using the polar coordinates we find the Cartesian coordinates $\vec{Q} = (Q_x, Q_y, Q_z)$:

$$\begin{aligned} Q_x &= \left(r_{moon} + \tilde{Q}_{alt}\right) \cdot cos(\tilde{Q}_{lat}) \cdot cos(\tilde{Q}_{lon}) \\ Q_y &= \left(r_{moon} + \tilde{Q}_{alt}\right) \cdot cos(\tilde{Q}_{lat}) \cdot sin(\tilde{Q}_{lon}) \\ Q_z &= \left(r_{moon} + \tilde{Q}_{alt}\right) \cdot sin(\tilde{Q}_{lat}) \end{aligned} \tag{8}$$

All elevation angles, β, along points in the array of distances, $\vec{d}$, are determined by view of $\vec{Q}$ from $\vec{P}$ for each azimuth direction Az:

$$\begin{aligned} \hat{PQ} &= \frac{\vec{Q} - \vec{P}}{\left\|\vec{Q} - \vec{P}\right\|} \\ \beta(Az, \vec{d}) &= arccos(\hat{P} \cdot \hat{PQ}) \end{aligned} \tag{9}$$

Finally, the elevation mask, α, along each azimuth angle, Az, is the largest elevation in view at distance, $\vec{d}$ from the observer:

$$\alpha(Az) = \max_{\vec{d}}(\pi/2 - \beta(Az, \vec{d})) \tag{10}$$

III. Implementation

A. SCPPM Concurrent Software Simulations

Figure 3 illustrates steps in the SCPPM algorithm to communicate data with PPM order $M = 16$. The data flow example in this figure shows the encoding and decoding operations in the SCPPM algorithm, which are implemented in the software to simulate bit error rates representative of the channel configuration in a real-world scenario. Following the example from [4], an error correction code of length $log_2(16)$ is concatenated with the data and used to decode at the receiver.

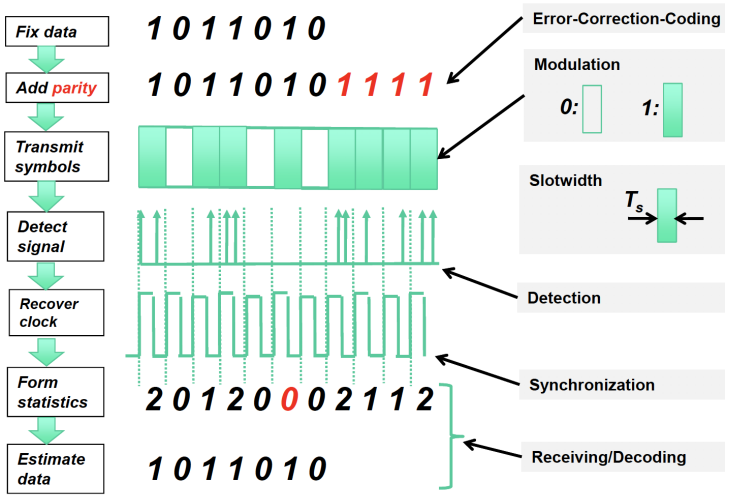

Fig. 3 Signal processing steps to communicate data from [4].

Figure 4 illustrates only the SCPPM encoder described in [2], which consists of an outer convolutional encoder, an interleaver, an accumulator, and an inner modulation encoder. The accumulate PPM (APPM) operation highlighted in the schematic requires the most time to complete in the software simulation. In the APPM, mapping of bits to PPM symbols is represented by a finite directed graph, resembling a trellis structure. The PPM order determines the number of trellis edges for convolutional encoding and dominates the run time of the simulation. Consequently the run-time increases as the PPM order increases.

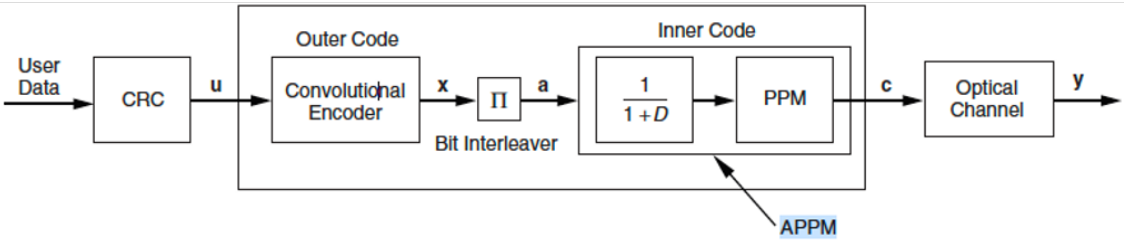

Fig. 4 SCPPM encoder.

A dtrace of the SCPPM software in figure 5 shows 86% of the time is spent in accumulate PPM, and about 50% of those hits are in maxstar. In the function call appm_bcjr we have an upper bound described by $O(N * M * log(M))$ where N is the number of symbols and M is the PPM order.

The inner code traverses through the trellis using the maximum likelihood path shown in figure 6. This pipe-lined structure is used in [2] to illustrate FPGA implementation, however all computations in this software simulation are serial. As shown in the pseudocode below, nothing is pipelined or done in parallel natively. Instead, we virtualize the existing software to run all instances simultaneously.

Fig. 5 Dtrace on SCPPM software.

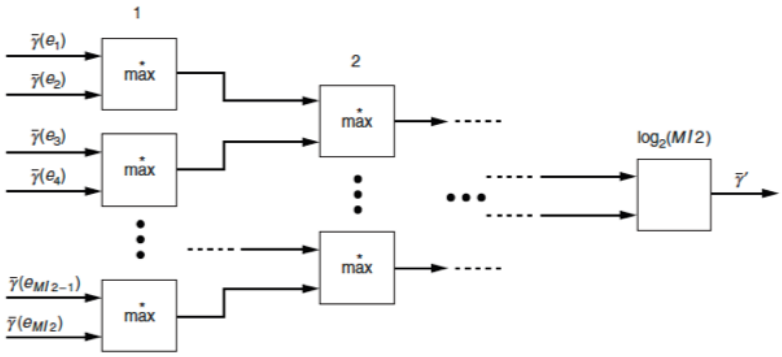

Fig. 6 Max-likelihood path pipe lined FPGA implementation.

```
// appm_bcjr maxstar serial operations
for (j=i=0;i<SYMS;i++) /* Compute extrinsic output */
   for (k=0;k<LOGM;k++)
      z=o=-BIGNUM;
      for (d=0;d<PPM;d++)
         if ((d>>k)&1) /* d has 1 in kth position */
            o = maxstar(o,a[i][0] + g[i][0][d] + b[i+1][ns[0][d]]);
            o = maxstar(o,a[i][1] + g[i][1][d] + b[i+1][ns[1][d]]);
         else
            z = maxstar(z,a[i][0] + g[i][0][d] + b[i+1][ns[0][d]]);
            z = maxstar(z,a[i][1] + g[i][1][d] + b[i+1][ns[1][d]]);
      e = o-z - extin[j];
      extout[j]= (e>QMAX) ? QMAX : (e<-QMAX) ? -QMAX : e; /* clip */
      j++;
```

Singularity [7] was used to build an image containing SCPPM software. The image was run on Shaheen rather than building and running the executable natively. Simulations for all configurations were run for 22 hours and the control scripts were used to launch multiple singularity containers. Figure 7 outlines the virtual machine architecture. On the left hand side the control script runs on the host operating system. This process manages the container environment on the right hand side and launches SPPM software with varying K_s and K_b values. Each container consists of the SCPPM object code and any library or system dependencies.

B. Psyche Trade-Space Exploration

From the simulation results, we are able to see the operational signal, K_s and noise regimes, K_b, corresponding to bit error rates within specified design tolerances. Our simulations focus on the operational phase for PPM M=16 and M=128 and slotwidth, T_s, 2.0 ns. The supportable PPM orders are determined using SOLT and are shown in figure 8. Projected supportable PPMs and slot width configurations for DSOC during the early mission phase were selected for simulation, which starts toward the end of 2022 through the beginning of 2023. The highlighted time-frame in the figure shows the region which we are generating the code rate map.

The SCPPM simulations started with $K_s = -6$dB-photons/slot and incremented by 0.1 dB when the number of code word errors detected reached 100. Each configuration was run twice and allowed to run for 22 hours, using a new random seed each time. This section describes results from running simulations for PPM-16 and PPM-128.

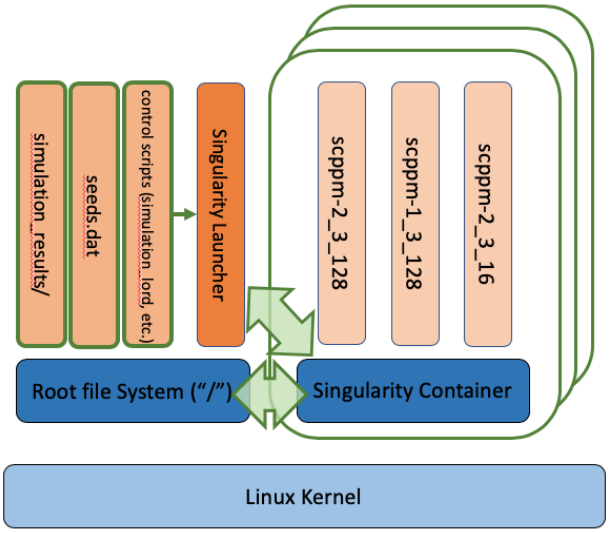

Fig. 7 SCPPM container architecture.

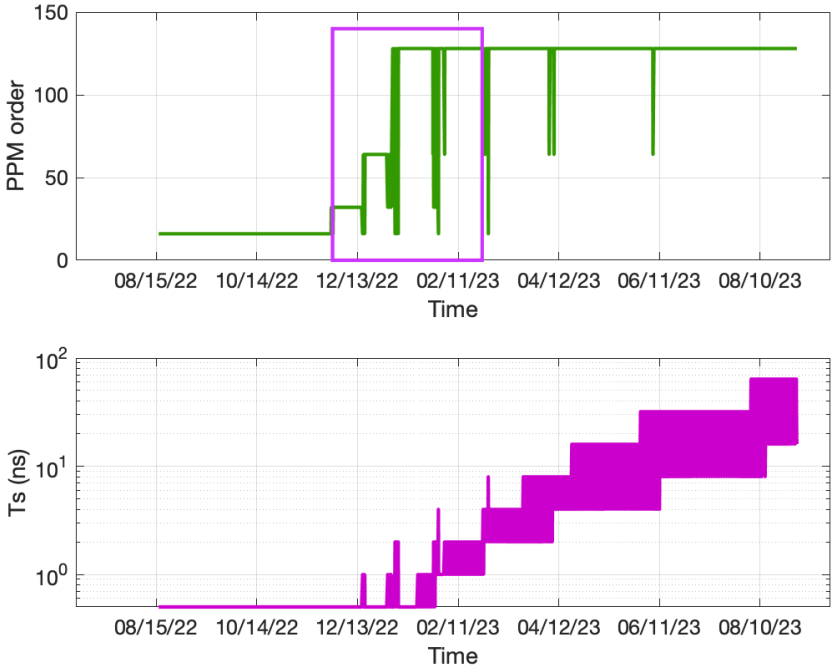

Fig. 8 Supportable PPM and slot widths projected for Psyche DSOC demo.

1. PPM 16

The SCPPM simulations from Shaheen were validated on for M=16 and rate 2/3 by comparing with existing data from previous simulations in figure 9. The bold curves are results from simulations in previous studies, and we confirmed the results produced from the virtual machines fall around existing plots and follow a similar trend.

For these BER curves, we averaged K_s over M slots and plot on a log scale for a quick visual inspection of inflection points and long tail end of the BER curves. Normalizing over all M slots steepens the slope and lengthens the BER curve tail-end to show the lowest achievable bit error rates in the asymptotic region. The 2-d BER curves show the asymptotic behavior at the tail-region around 10e-8, where we stopped the simulations.

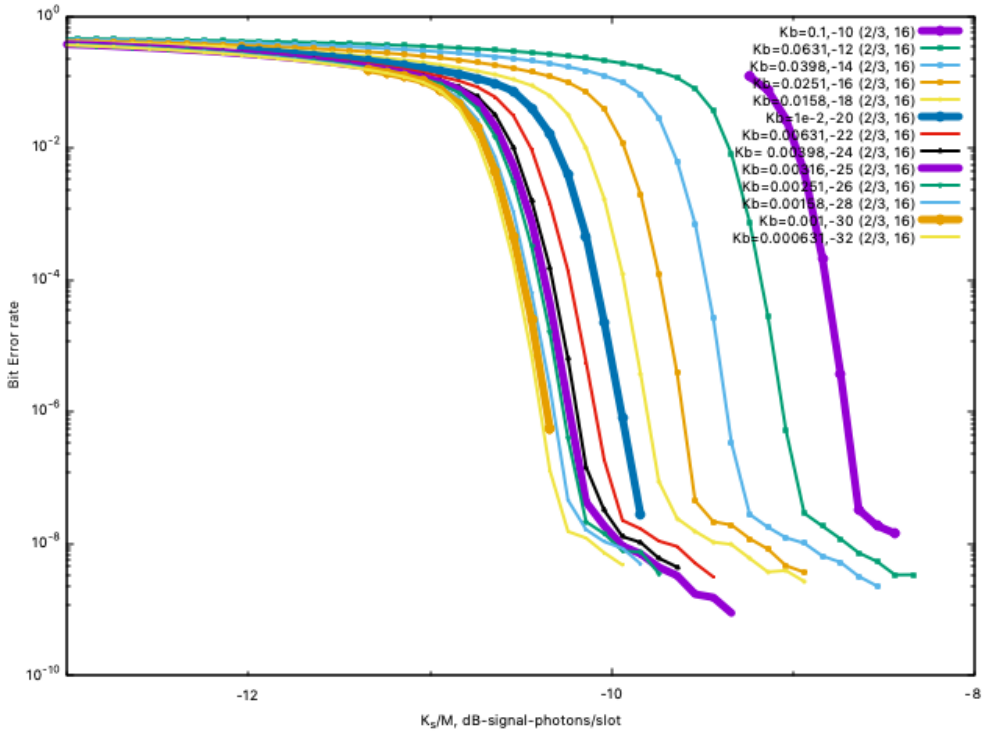

Fig. 9 BER curves for PPM 16, rate 2/3.

The surface plot in figure 10 is interpolated from the BER curves from figure 9. From this waterfall plot we can see the K_b and K_s values that produce bit error rates within specified design tolerances for this particular PPM order and rate. For example, the contour line below in figure 10 shows the operating regimes which achieve BER 10e-6. The curve shows a transition from signal dominant to noise dominant regime for this bit error rate using PPM-16 rate 2/3.

2. PPM 128

Figure 11 shows SOLT projected supportable K_b values for Psyche DSOC. K_b (dB-photons/slot) was adjusted by assuming PPM-128. SCPPM simulations were repeated for PPM-128 rate 2/3 using these K_b values and BER curves plotted in figure 12. The resulting 2-d BER curves show an asymptotic behavior near BER 10e-7. We average over all M slots and plot on log scale for a quick visual representation of the asymptotic region. From looking at the tail region of these curves, we can glean insight to K_b and K_s regimes that produce bit error rates as low as 10e-7.

The surface plot shown in figure 13 was generated using the BER curves for PPM128, rate 2/3. From this surface plot we can trace the contours along the bit error rates corresponding to mission design tolerance. For instance in the bottom of figure 13, we traced the contour along the surface plot corresponding to BER 10e-6 to find corresponding signal and noise regimes for this configuration.

3. Generating Code Rate Maps

We repeat this process for all configurations, and plot all corresponding curves for code word error rate 10e-4 in figure 14. On the resulting code rate map, we plot the time varying K_s and K_b projected for Psyche DSOC which starts at the end of 2022 until beginning of 2023. The open circles along the projected path corresponds to approximately three days resolution. All map regions produced in figure 14 required less than 1,000 core hours from the project allocation on Shaheen II (each waterfall plot used approximately 130 core hours).

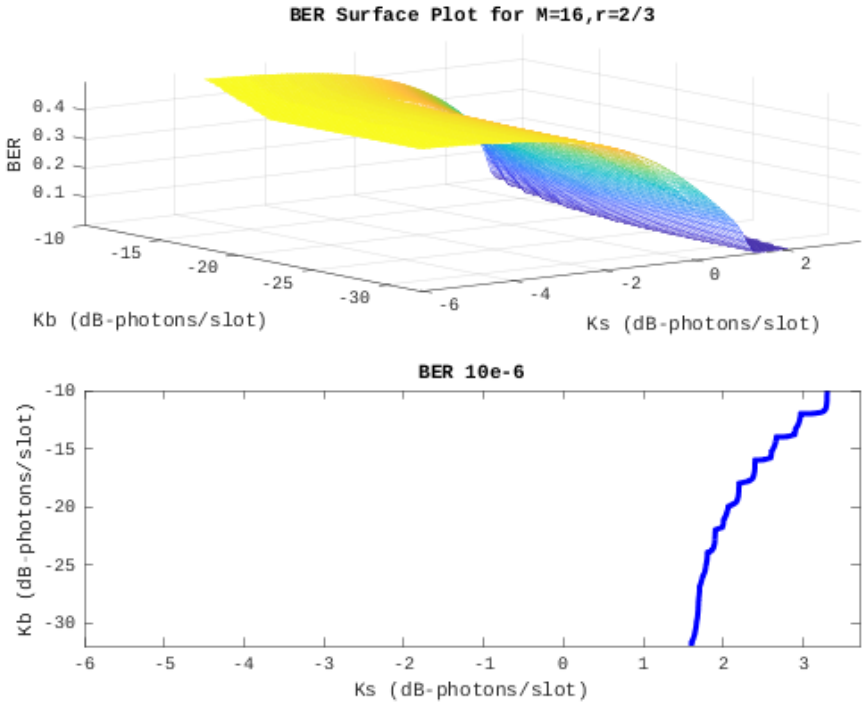

Fig. 10 Above: Surface plot interpolated from BER curves. Below: BER 10e-6 K_s and K_b regime.

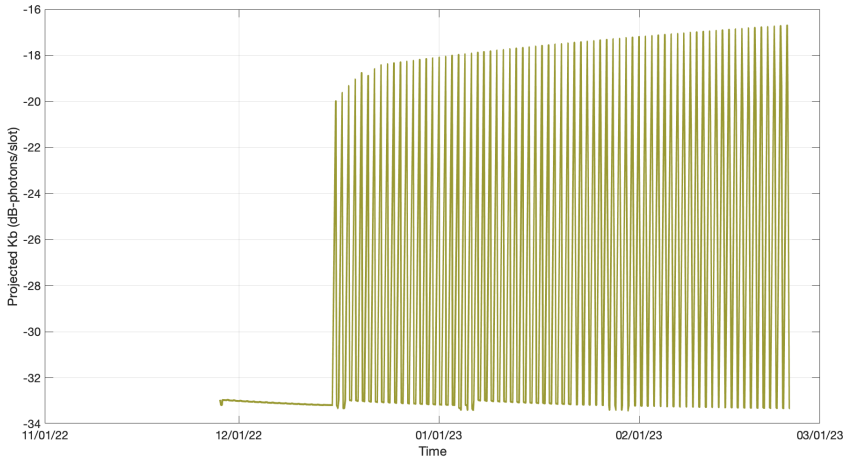

Fig. 11 Projected K_b adjusted assuming PPM-128.

C. Lunar South Pole Masks

In this section we discuss how to leverage a parallel worker pool to run algorithms in native Python, and to produce az-el masks for a large subset of pixels. Algorithm design and development was enabled in the Matlab environment. However initial attempts to produce az-el masks for a subset of 851x1001 pixels overwhelmed memory available on a personal computer (32GB RAM and 2.9 GHz Intel Core i9 Processor). The average wall clock time to produce az-el masks for 2,500 pixels was 0.785 seconds. A back of envelope calculation suggests a wall time of approximately 186 hours for the entire set. On Shaheen, producing az-el masks for this set required roughly 14 hours of wall time and required approximately 1,500 core hours from the project allocation.

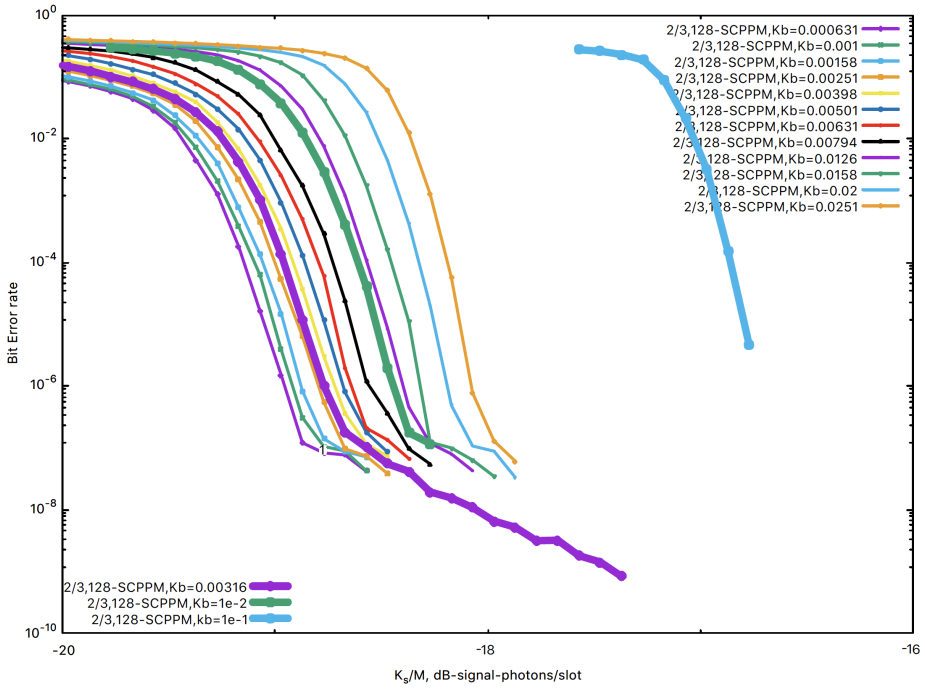

Fig. 12 BER curves for PPM 128, rate 2/3.

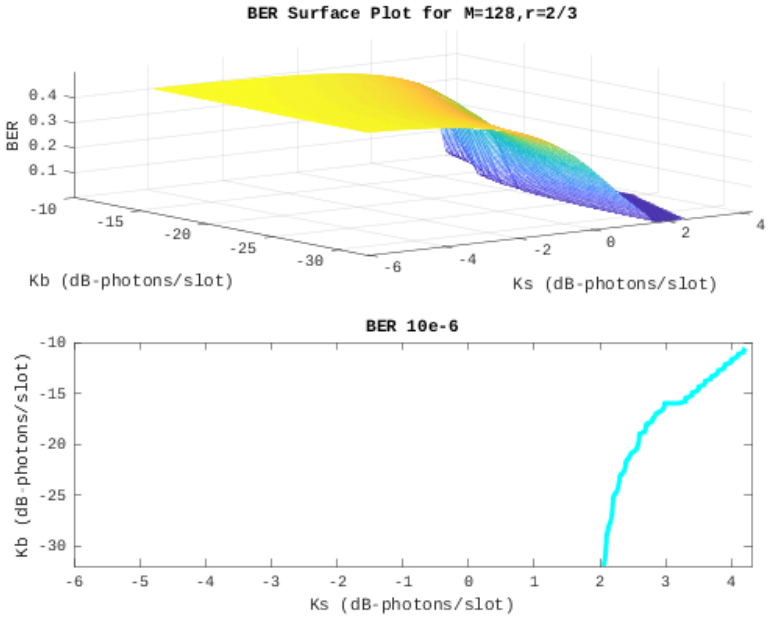

Fig. 13 Above: Surface plot interpolated from BER curves. Below: BER 10e-6 K_s and K_b regime.

In order to design a parallel workflow, a Jupyter notebook was set up on the production environment in debug mode. The notebook environment provided a framework to unit test sub tasks for parallel computing. Port forwarding was configured on Shaheen to enable an interactive notebook from localhost on a web browser. The Dask client enabled this application to run on 32 threads, and balance load and memory usage across all resources. For instance, in figure 15, a uneven task burden on one worker suggests to partition input sizes into smaller chunks.

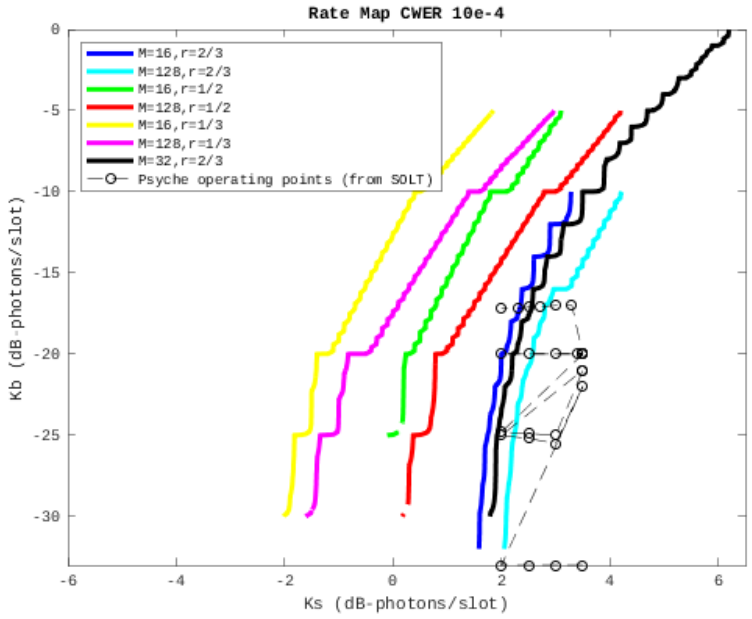

Fig. 14 Psyche DSOC code rate map path plan for CWER 10e-4.

Fig. 15 Dask dashboard.

Many branch points were considered and evaluated qualitatively using the Dask client viewer and Jupyter notebook. Because the software was developed directly in the production environment, we were able to refactor our code base for optimal scheduling and task or memory balancing. The flow chart in figure 16 shows the architecture used to produce all az-el masks in the region represented by 851x1001 pixels. In this approach, a job array was launched using 2 nodes and 16 cores per node with the following steps:

1) We partition data to submit multiple jobs, each processing 10,000 pixels. A Slurm job array assigns pixel

numbers for each job submission and passes them as arguments to the terrain mask calculation program.

2) For each set of 10,000 pixels,
    1) Each job array submission creates a Dask client with 32 threads. For each job,
    2) The base image is read and all pixels processed to produce az-el masks. For each pixel,
        1) Compute each element in the coordinate system transformations in parallel.
        2) Run mask caclulation algorithm.
        3) Each pixel produces two files: one file for elevation masks at all angles surrounding it, and another file with distances from that pixel to the elevation mask.

For the az-el masks produced in this work, over 1.7 million files and 154MB of data products were generated.

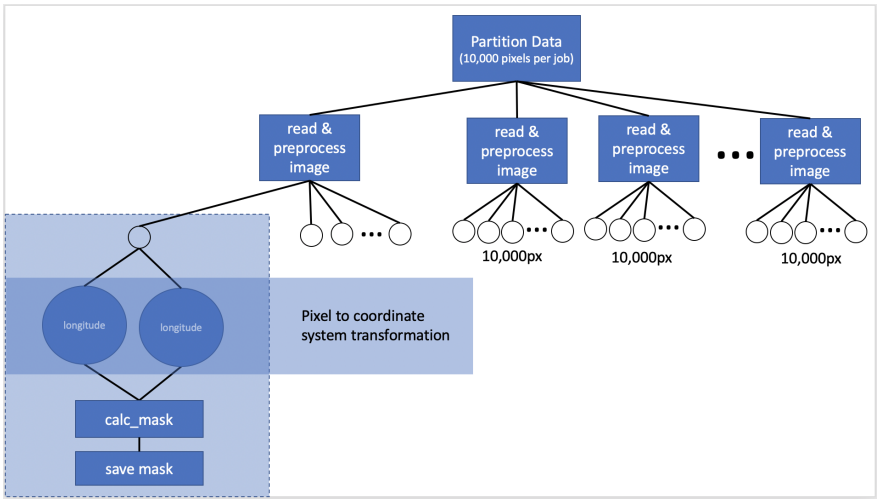

Fig. 16 Parallel flow chart.

IV. Conclusion

Our approach upgrades capabilities of existing DSOC software tools and algorithms to cover a larger scope of communications link and networking trade-spaces. Code rate maps can be extended to support longer operational periods and augmented with additional operational configurations. By simulating lower bit error rates, this approach allows for exploration of more stringent link margin requirements during critical mission phases. Similarly, terrain mask calculations accelerated with parallelized workloads explore larger coverage regions of the lunar surface, enabling roving missions, such as Lunar VIPER Mission, to identify regions with reliable communications coverage.

Acknowledgments

C. Lee thanks Mohsin Shaikh of King Abdullah University of Science and Technology for assistance with setup and troubleshooting applications on Shaheen and Harvey Newman of California Institute of Technology for insightful discussions about super computing applications. The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA). This research used the resources of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) in Thuwal, Saudi Arabia.

References

[1] Xie, H., and Cheung, K. M., "Statistical Optical Link Budget Analysis," *2019 IEEE Aerospace Conference*, 2019, pp. 1–6.

[2] Moision, B., and Hamkins, J., "Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation," *Interplanetary Network Progress Report*, Vol. 42-161, 2005, pp. 1–25.

[3] Xie, H., Heckman, D., and Breidenthal, J., "Link Characterization for Deep-Space Optical Communications," *Interplanetary Network Progress Report*, Vol. 42-205, 2016, pp. 1–33.

[4] Dolinar, S., Moision, B., and Erkmen, B., "Fundamentals of Free-Space Optical Communication," Keck Institute for Space Studies (KISS), Jun 2012.

[5] Lee, C. H., and Cheung, K. M., "Complete Lunar Exploration Coverage Analysis," *Interplanetary Network Progress Report*, Vol. 42-175, 2008, pp. 1–30.

[6] Smith, D. E., Zuber, M. T., Jackson, G. B., Cavanaugh, J. F., Neumann, G. A., Riris, H., Sun, X., Zellar, R. S., Coltharp, C., Connelly, J., Katz, R. B., Kleyner, I., Liiva, P., Matuszeski, A., Mazarico, E. M., McGarry, J. F., Novo-Gradac, A.-M., Ott, M. N., Peters, C., Ramos-Izquierdo, L. A., Ramsey, L., Rowlands, D. D., Schmidt, S., Scott, V. S., Shaw, G. B., Smith, J. C., Swinski, J.-P., Torrence, M. H., Unger, G., Yu, A. W., and Zagwodzki, T. W., "The Lunar Orbiter Laser Altimeter Investigation on the Lunar Reconnaissance Orbiter Mission," *Space Science Reviews*, Vol. 150, No. 1-4, 2010, pp. 209–241. https://doi.org/10.1007/s11214-009-9512-y.

[7] Kurtzer, G. M., Sochat, V., and Bauer, M. W., "Singularity: Scientific containers for mobility of compute," *PLOS ONE*, Vol. 12, No. 5, 2017, pp. 1–20. https://doi.org/10.1371/journal.pone.0177459, URL https://doi.org/10.1371/journal.pone.0177459.