

# Real-Time Wavefront Processors for the Next Generation of Adaptive Optics Systems: A Design and Analysis

Tuan Truong\*<sup>a</sup>, Mitchell Troy<sup>b</sup>, Gary Brack<sup>b</sup>, Thang Trinh<sup>a</sup>, Fang Shi<sup>b</sup>, Richard Dekany<sup>c</sup>  
<sup>a</sup>User Technology Associates, Inc.; <sup>b</sup>Jet Propulsion Laboratory; <sup>c</sup>California Institute of Technology

## ABSTRACT

Adaptive optics (AO) systems currently under investigation will require at least two orders of magnitude increase in the number of actuators, translating effectively to a  $10^4$  increase in compute latency. Since the performance of an AO system will in general improve as the compute latency decreases, it is important to study how today's computer systems will scale to address this expected increase. This paper answers the question by characterizing the performance of a single deformable mirror (DM) Shack-Hartmann natural guide star AO system based on the present-generation digital signal processors TMS320C6701 from Texas Instruments. We derive the compute latency function of such a system in terms of a few basic parameters, such as the number of DM actuators, how many data channels are used to read out the camera pixels, the number of processors, the available memory bandwidth as well as the inter-processor communication (IPC) bandwidth, and the pixel transfer rate. We show how the results would scale for future systems using multiple DMs and guide stars. We demonstrate the main bottleneck to performance of such a system is the available memory bandwidth of the processors and to some extent the IPC bandwidth. The paper concludes with suggestions for mitigating this bottleneck.

**Keywords:** Adaptive optics, wavefront processors, TMS320C6701

## 1. INTRODUCTION

Adaptive optics systems are now considered a standard instrument for all ground based telescopes. Using adaptive optics the effects of atmospheric turbulence can be ameliorated and the resolution benefits of larger telescope collecting apertures realized. Without the benefits of AO the typical size of an image is the observing wavelength ( $\lambda$ ) divided by the atmospheric coherence length ( $r_0$ ), where  $r_0$  is on the order of 75cm (at  $\lambda=1.5\mu\text{m}$ ). However, at wavelengths longer than  $1\mu\text{m}$ , diffraction limited images ( $\lambda/D$ ) are now routinely achieved using AO and ground based telescopes. This is over a factor of ten increase in resolution and a factor of 2 (at Palomar) to 4.0 (at Keck) better than the Hubble space telescope resolution. Hence, there is a desire to implement AO systems that operate on larger ground-based telescopes, at shorter wavelength and that provide wide field of views of correction, using multiple DMs and guide stars. These factors all increase the computational requirements.

Several AO systems currently under investigation [1] will require at least two orders of magnitude increase in the number of DM actuators. This translates to a  $10^4$  increase in compute latency, as most of this time is dominated by the reconstruction of the wavefront from slopes to DM commands that is traditionally carried out by a matrix of size  $N$  by  $2N$ , where  $N$  is the number of DM actuators. Since the performance of an AO system will in general improve as the compute latency decreases, it is important to study how today's computer systems will scale to address this high latency. This paper answers this question by characterizing the performance of a single DM Shack-Hartmann natural guide star AO system based on the present-generation digital signal processors (DSPs) TMS320C6701 from Texas Instruments (TI). We derive the overall compute latency of such a system and show how the results would scale for future systems (with multiple DMs and guide stars).

The rest of this paper is organized as follows: In section 2 we briefly review the data flow and algorithms implemented for the Shack-Hartmann AO system. Section 3 describes the hardware architecture selected to minimize the compute latency and discusses limitations imposed by the hardware selected. Section 4 details the predicted and realized performance of each algorithm and discusses the architectural constraints limiting it. Section 5 presents the total system compute latency. Section 6 works out the latency for two example AO systems. Section 7 concludes the paper.

## 2. DATA FLOW

Figure 2.1 below depicts the dataflow of a wavefront processor. As shown, each frame of pixels is first split by the camera hardware into equal and non-overlapping regions that are read out simultaneously using multiple dedicated channels. The pixels at *each* channel are then distributed and processed in a pipelined fashion by the wavefront processor using a number of DSPs (only one shown in Figure 2.1). Each DSP performs the same processing steps needed in a Shack-Hartmann adaptive optics system which include sky-background subtraction, flat fielding, centroiding, and wavefront reconstruction. The results from the reconstruction are merged (summed), again in a pipelined fashion, first from all DSPs within the same channel, then across all the channels - to form one true residual vector from which the servo commands to the DM and FSM (fast steering mirror) are constructed.

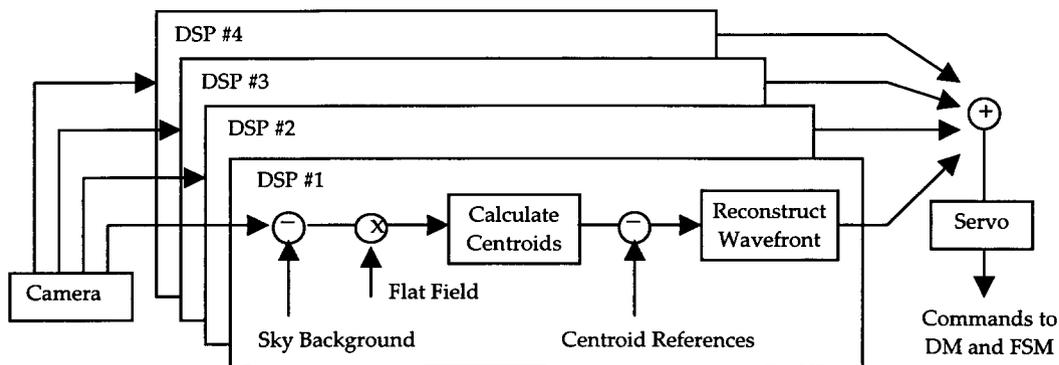


Figure 2.1: Data flow of a wavefront processor taking input from a camera with quad-channel simultaneous readout capability. Though figure shows only one, multiple DSPs can often be used to further reduce the compute latency.

### 3. HARDWARE ARCHITECTURE

The wavefront processing algorithms are performed by a number of TMS320C6701 DSPs configured in groups of 4 per board all housed in a VME-based control computer. In this section, we highlight the key architectural strengths of the 'C6701 that enable the application's high performance as well as the constraints that limit it. We present what we called the system-level constraints that are characteristic (and perhaps unique) of the boards we selected but that affect the overall application performance. This section ends with a description of the DSP network topology.

#### 3.1 TMS320C6701 DSP Architecture

The TMS320C6701 is TI's first generation floating point processor to use the high performance *VelociTI* architecture with its advanced very-long-instruction-word (VLIW) engine for instruction-level parallelism [2]. As shown in Figure 3.1, the 'C6701 core is divided into two data paths (A,B) each with the same set of four independent functional units, a register file with sixteen 32-bit general-purpose registers, and paths for addressing and moving data between memory and registers. All four functional units in each data path support integer operations. Except for the address generation unit (D), the other units also support both 32-bit and 64-bit floating point operations. The L and M units, in particular, are dedicated to floating point additions and multiplications, respectively, while the S unit is dedicated to floating point reciprocal, and reciprocal square root estimation. With three floating point units per data path, the 'C6701 is capable of up to six floating point operations or two multiply-accumulates (MACs), per clock cycle, for a total of 1 giga FLOPS or 334 million MACs, at a clock rate of 167 MHz.

The 'C6701 has 128 Kbytes of on-chip memory, evenly divided between program and data space. The on-chip program memory has a dedicated 256-bit path to the CPU core, allowing it to fetch an 256-bit VLIW instruction packet contained eight 32-word instructions, one per functional unit, every clock cycle. In contrast, all off-chip memory access, be it program or data fetch (or store), occurs over the 32-bit external-memory-interface (EMIF) bus, thus requiring at least eight cycles to fetch the same VLIW packet. Since the off-chip access rate is at least one cycle per 32-bit word, the 'C6701 must execute from on-chip program memory for good performance.

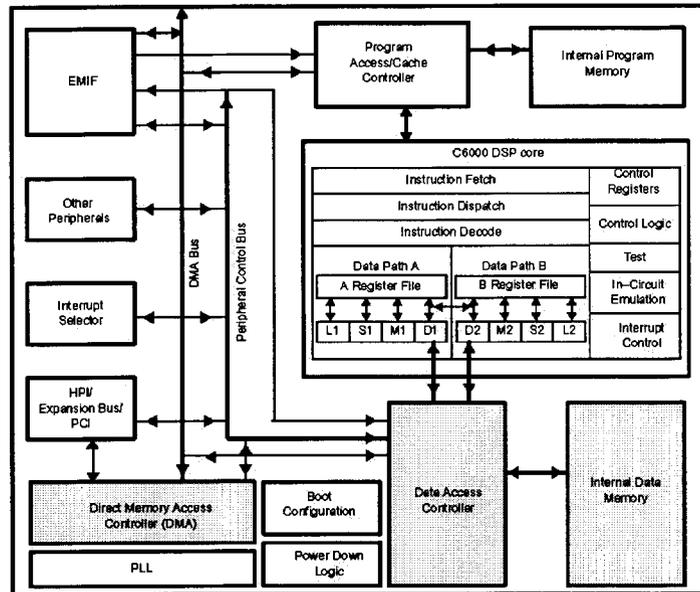


Figure 3.1: TMS320C620x/C670x Block Diagram. Reprinted by permission of Texas Instruments, Inc [3].

The 'C6701 on-chip data memory has two 32K-byte blocks each is organized as eight 4K banks of 16-bit halfwords that can be accessed simultaneously by both CPU and DMA without performance penalty, provided that different on-chip memory banks (of the same block or not) are used [3]. Since each of the two data paths is connected to the data memory by two 32-bit data buses, the 'C6701 can load two 64-bit words per instruction cycle. Coupled this with the EMIF bus, the maximum data accesses each cycle, defined herein as *memory bandwidth*, is one DMA access and two CPU accesses, where each CPU access may be a 32-bit store, a 32-bit load, or a 64-bit load. Hence, to obtain maximum memory bandwidth utilization, not only the 64-bit loads should be used (in place of the 32-bit loads), but concurrent background DMA transfer of external (off-chip) data via the EMIF bus should also be exploited.

It is clear that off-chip memory accesses incur severe performance penalty. However paramount, avoiding them may be impossible as the size of the data needed for the calculation increases (with more DM actuators or wavefront sensing cameras). The on-chip memory will inevitably be insufficient in size and the slower external memory hierarchy will need to be used. Once this occurs, the said memory bandwidth becomes the performance bottleneck. A simple way to mitigate this bottleneck is to use multiple DSPs and partition the dataset among the available on-chip memory areas. As the number of DSPs increases, the IPC bandwidth may become a limiting factor (as Section 5 shows). A tradeoff must eventually be performed between the use of slower external memories and further increasing the number of processors. The compute latency function derived in Section 5 simplifies this tradeoff process.

The 'C6701 provides 4 independent programmable DMA channels (each with separate context) allowing movement of data to and from internal (on-chip) memory and external peripherals to occur entirely in the background of CPU operation. Each DMA transfer (read or write) may be set up to synchronize separately with either an external signal (such as the start of a frame) or an internal event (such as a timer expiration or another DMA completion). We exploit this feature not only to avoid the high overhead of an interrupt service routine (ISR) traditionally associated with interrupting the CPU, when possible, but also to realize the theoretical peak throughput. For instance, to maximize the on-chip memory utilization, we pervasively load 64 bits of data at a time which required that interrupts be disabled for the duration of the instruction execution. Since these 64-bit loads are issued every cycle, this potentially leads to a prolonged period of interrupt lockout and hence high ISR dispatch latency. If the ISR had to execute to start another DMA transfer (such as in the case of pixel forwarding or when the FIFO buffer almost-full condition is detected and the pixels must be brought on-chip), then the latter transfer will be delayed, resulting in a sub-optimal overall latency. By using the completion event of one DMA transfer to automatically trigger the start of another without the CPU intervention of an ISR, we avoided the unnecessary overhead. We also made use of the DMA autoinitialization feature that enables a channel to automatically reinitialize itself for the next or repetitive transfers, with just an one-time setup. In short, we've found these architectural supports critical to enabling maximum data throughput and CPU performance.

### 3.2 Pentek 4291 Board Architecture

For the platform, we selected the high-performance DSP boards (Model 4291) from Pentek Inc [4]. As Figure 3.2 shows, each board has 4 identical DSPs each equipped with both private and shared resources. Following is a brief description of only the resources used by our application:

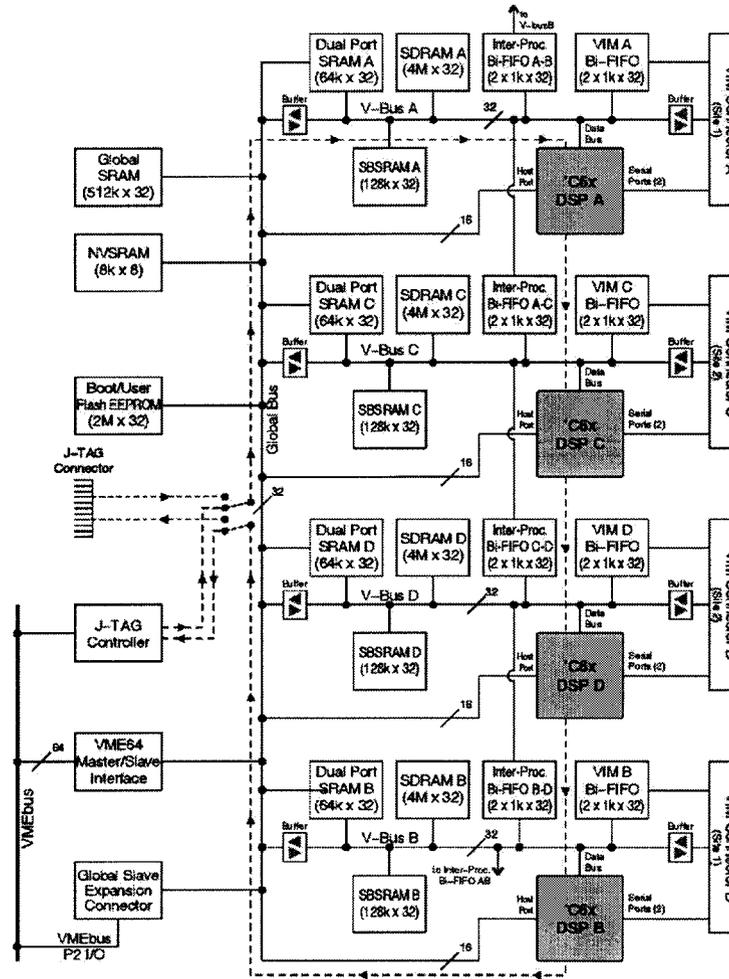


Figure 3.2: Functional Block Diagram – Model 4290/4291. Reprinted by permission of Pentek, Inc [4].

1. The 512 KB synchronous burst SRAM (SBSRAM) with 1-clock access, or 667 MB/sec, is the fastest of external memory available on the board. It is used to hold data that exceeds the amount of on-chip memory area, primarily the wavefront reconstruction matrix.
2. The 256 KB dual-port SRAM with 16-clock access (after arbitration which involves additional logic to obtain the semaphore) provided that access is made via the EMIF bus or 40-clock access if made via the shared global bus, is the fastest shared external memory that is accessible by both the VME host computer and the DSPs. We used this memory like a ring buffer for any real-time DSP data (such as the raw pixels, flat fields, xy-centroids, and DM residuals and positions) that we wish the VME host to save away for off-line analysis. We did not use the 2 MB global SRAM, which is the other shared resource available, for this purpose because of the potential bus contention on writes by the DSPs and of the slow access time, 35 clocks per read or write.
3. The three bi-directional FIFOs (two IP bi-FIFOs and one VIM bi-FIFO) with 9-clock access on reads and 8-clock access on writes, provide the fastest means of exchanging data with the two neighboring DSPs or with a

VIM-compatible I/O device. On the 4291 Standard Model, only one bi-FIFO is accessible at anytime – thus, to automatically store-and-forward the raw pixels using DMA, the VIM bi-FIFO must be used. Since our purchase, Pentek has released a more versatile version of the 4291 board (Option 330) that permits not only simultaneous access to all three bi-FIFOs but also has much lower latency for both reads and writes.

4. We used the front-panel VIM-compatible I/O connector for interfacing to the quad-channel readout EEV CCD39 wavefront sensing camera from SciMeasure, Inc. [5,6]

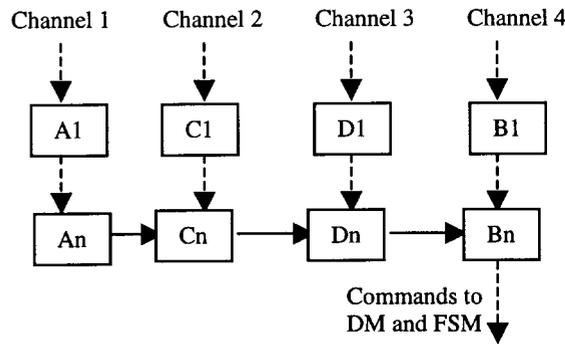
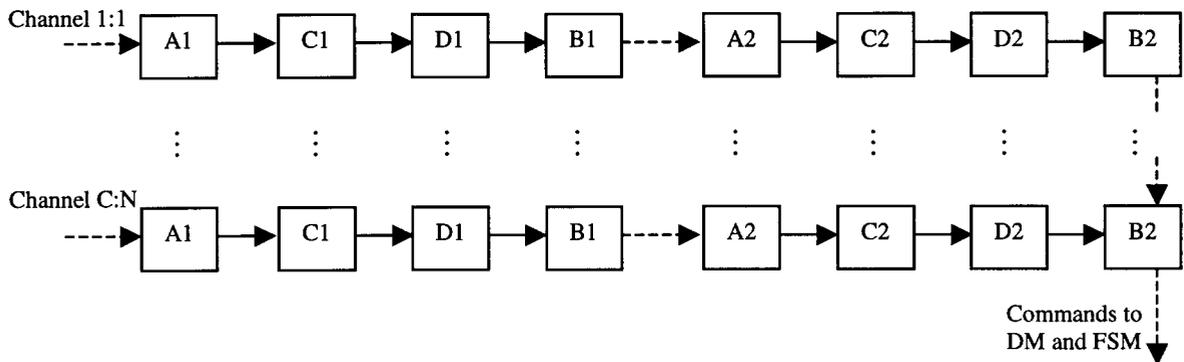


Figure 3.3(a) on the left shows a DSP network of n Pentek 4291-boards (standard model) driving one DM on data from one camera with 4-channel readout and n dedicated DSPs per channel, where  $n \leq 4$ . The dashed arrow utilizes the VIM bi-FIFO path while the solid utilizes the IP bi-FIFO path.

Figure 3.3(b) below shows a DSP network of 2NC 4291-boards (Option 330) driving one DM on data from N cameras each with C-channel readout and 8 dedicated DSPs per channel. The dashed arrow utilizes the VIM bi-FIFO path while the solid utilizes the IP bi-FIFO path.



### 3.3 DSP Network

The bi-FIFO limitation of the 4291 Standard Model limited us to the topology shown in Figure 3.3(a) for networking the DSPs. There, we dedicated one board to interface to the camera and a processor to each readout channel. The raw pixels from channels 1, 2, 3 and 4 are fed into the VIM FIFOs of processors A1, C1, D1, and B1, respectively. The start-of-frame sync signal from the camera synchronizes the processing of these four DSPs. When multiple DSPs are used to process a channel, these four are responsible for forwarding the pixels to the remaining DSPs. With this configuration, multiple results from each channel are first merged using the VIM bi-FIFOs and then across all channels using the IP bi-FIFOs of the last board in the daisy-chain.

One can extrapolate this interconnection to support multiple cameras with multiple-channel readout. Figure 3.3(b) shows a simple grid-like setup using N cameras each with C-channel readout and eight dedicated DSP per channel. Note that Option 330 of the 4291 model is required for this setup (i.e., one with more than four channels total) or for any setup with better-than-linear scaling topology (such as a tree).

## 4. ALGORITHM ANALYSIS

In this section, we present the detailed performance analysis of the real-time algorithms needed in a Shack-Hartmann adaptive optics system. We show that our implementation of the algorithms is asymptotically optimal given the architectural constraints described in Section 3.1. The proof is based on the simple idea: Since each of the eight functional units uses a 32-bit instruction and one 256-bit VLIW instruction packet is fetched every clock cycle, it is beneficial to maximize the number of units executing useful operations per cycle. Hence for this reason, we unrolled

(software-pipelined) all of the compute-intensive loops. How many times to unroll a loop depends on which 'C6701 architectural constraint dominates the calculation. On one occasion, we found that the theoretical number offered by the dominating constraint was impossible to realize (i.e., schedulable with any valid sequence of instructions) due to an undocumented constraint (a CPU bug in our opinion but has yet been confirmed by TI). For an exhausted list of constraints, please see reference [2].

As an example, the wavefront reconstruction loop is unrolled 3 times to allow 4 iterations to execute in parallel – for which at least 4\*3 words must be fetched and 4 words stored. One can schedule these memory accesses using 5 instruction packets, one per cycle, where three of the instruction packets each perform two 64-bit loads and the remaining two each perform two 32-bit stores. This schedule is optimal because all of the available memory bandwidth are utilized in order to perform the minimum needed memory accesses. Had the loop been twice unrolled which would have resulted in 3 iterations executing in parallel which would in turn required at least 3\*3 words be fetched and 3 words be stored, the best schedule under this scenario would have required 4 parallel instructions - two of which would each issue two 64-bit loads while the third instruction would issue two 32-bit stores and the last instruction would have a 32-bit load in parallel with a 32-bit store. Since the 32-bit load is used instead of the 64-bit load, this schedule does not fully exploit the available on-chip memory bandwidth and therefore it is sub-optimal.

#### 4.1 Flat fielding

Eq. 4.1.1 below shows how the flat field values are calculated, given the input raw pixels, the pixel offsets (sky background) and the pixel gains, for  $n$  2x2 subapertures or  $4n$  pixels.

$$p_i^{flat} = (p_i^{raw} - p_i^{offset}) \cdot p_i^{gain} \quad \text{for } 1 \leq i \leq 4n \quad (4.1.1)$$

Eq. 4.1.2 gives the exact performance  $T_{calc}^{flat}(n)$  of our flat field implementation in processor cycles. The constant 15 is the overhead incurred by the filling and flushing of the software pipeline.

$$T_{calc}^{flat}(n) = 5n + 15 \text{ cycles} \quad (4.1.2)$$

To maximize the available on-chip memory bandwidth, we unroll the loop denoted by Eq. 4.1.1 three times to allow 4 flat field values to be calculated in parallel per loop iteration hence requiring at least 4\*3\*32 bits of input data be fetched and 4\*32 bits of output data be stored. This can be scheduled using 5 instruction packets, one per cycle, where three of the instructions each performed two 64-bit loads while the last two each performed two 32-bit stores. Since there are 4 pixels per 2x2 subaperture, this translates to an asymptotic performance of 5 cycles per subaperture.

Eq. 4.1.3 below shows the total time to load the pixel offsets and the pixel gains from external memory to on-chip memory for  $n$  subapertures.  $t_{mem}$  denotes the average number of cycles to load one 32-bit word from external memory.

For single-cycle throughput SBSRAM,  $t_{mem} = 1$ . Unlike Eq. 4.1.2 that runs repeatedly for every  $n$  subapertures, the DMA setup code that receives the raw pixels runs only once per frame. As such, it is assumed that each DSP's on-chip memory is large enough to hold all of the raw pixels coming its way. Thus, Eq. 4.1.3 included neither the raw pixels' load time nor the software pipeline overhead.

$$T_{xfer}^{flat}(n) = 8n \cdot t_{mem} \text{ cycles} \quad (4.1.3)$$

#### 4.2 Centroiding

The quadcell centroiding algorithm consists of three loops described by Eqs. 4.2.1 through 4.2.3. Their combined performance for  $n$  subapertures is given in Eq. 4.2.4.

$$\begin{aligned} u_i &= (b_i - a_i) + (d_i - c_i) \\ v_i &= (b_i + a_i) + (d_i + c_i) \\ w_i &= (b_i + a_i) + (d_i + c_i) \end{aligned} \quad \text{for } 1 \leq i \leq n \quad (4.2.1)$$

The first loop denoted by Eq. 4.2.1 calculates the subaperture flux  $w_i$  and partial x-y centroid values  $(u_i, v_i)$  given each subaperture's four pixel flat fields  $(a_i, b_i, c_i, d_i)$ . For performance reason, we push the division by the subaperture flux out of Eq. 4.2.1 (where it is normally done) and into Eq. 4.2.3. To keep the two floating point adders

busy every cycle, we unroll the loop 3 times to allow 4 subapertures to be calculated in parallel hence requiring a minimum of 4\*7 adds/subtracts per iteration which can be scheduled with 14 instruction packets each performing 2 adds/subtracts per cycle.

$$\begin{aligned} z_{i,j} &= z_{i,j-1} \cdot (2 - w_i \cdot z_{i,j-1}) \\ w_i^{-1} &= z_{i,2} \end{aligned} \quad \text{for } 1 \leq i \leq n, \text{ and } z_{i,0} = \_rcpsp(w_i) \quad (4.2.2)$$

The second loop denoted by Eq. 4.2.2 calculates nothing more than the reciprocal of a 32-bit floating point value  $w_i$ , for  $n$  values. To obtain the desired 23 bits mantissa accuracy for  $w_i^{-1}$ , the loop performs two iterations of the Newton-Rhapson algorithm on an initial estimated reciprocal value returned from the assembly instruction  $\_rcpsp$  that provides only 8 bits mantissa accuracy. The purpose of this vectorized routine is to minimize the average number of cycles required to perform a reciprocal operation through software pipelining. With appropriate loop unrolling, it is possible to deliver one reciprocal every 2 cycles (the maximum performance achievable with two S units), in contrast to 27 cycles with the non-vectorized version from Texas Instrument.

$$\begin{aligned} x_i &= u_i \cdot w_i^{-1} - x_i^{ref} \\ y_i &= v_i \cdot w_i^{-1} - y_i^{ref} \end{aligned} \quad \text{if } r_i \geq r_{\min}, \text{ else } x_i = y_i = 0, \text{ for } 1 \leq i \leq n \quad (4.2.3)$$

The third loop denoted by Eq. 4.2.3 calculates the centroid offsets  $(x_i, y_i)$  for  $n$  subapertures, given the fixed centroid references  $(x_i^{ref}, y_i^{ref})$  and the results of previous two loops  $(u_i, v_i, w_i^{-1})$ . Again, to obtain peak performance, we unroll the loop 3 times to allow 4 centroid offset pairs to be calculated per iteration resulting in a schedule with nine instruction packets, one per cycle, where five of the instructions each perform two 64-bit loads and the other two each perform two 32-bit stores. This translates to a performance of 9 cycles per 4 subapertures.

$$T_{calc}^{cent}(n) = \frac{31}{4}n + 33 \text{ cycles} \quad (4.2.4)$$

Eq. 4.2.4 above gives the combined performance for all three loops. Eq. 4.2.5 below gives the total time to load the input data (centroid references) required by the three calculations from external memory for  $n$  subapertures.

$$T_{xfer}^{cent}(n) = 2n \cdot t_{mem} \text{ cycles} \quad (4.2.5)$$

### 4.3 Wavefront Reconstruction

Eq. 4.3.1 shows the pipelined implementation of the general matrix vector multiplication that also scales efficiently to banded matrices. Its peak performance  $T_{calc}^{recon}(n)$  is given in Eq. 4.3.2.

$$r_k = r_k + R_{k,2j-1} \cdot x_j + R_{k,2j} \cdot y_j \quad \text{for } 1 \leq j \leq n, 1 \leq k \leq M, \text{ and } r_0 = 0 \quad (4.3.1)$$

The factor  $f_D$  in the equation denotes the ratio of the diagonal band to the entire  $M$ -by- $2M$  matrix  $R$  area or loosely speaking a measure of denseness or non-sparseness due to the diagonal band. By definition,  $f_D$  ranges between 0 and 1, with 1 representing the general matrix.  $M$  denotes the number of actuators (same as subapertures) in the system.

$$T_{calc}^{recon}(n) = \left(\frac{5}{4}f_D M + 12\right) \cdot n + 7 \text{ cycles} \quad (4.3.2)$$

To maximize the memory bandwidth utilization, we unroll the loop 3 times resulting in 4 products computed in parallel per iteration for which at least 4\*3\*32 bits must be fetched and 4\*32 bits be stored. One can schedule these memory accesses using 5 parallel instructions, one per cycle, where three of the instructions each perform two 64-bit loads and the remaining two each perform two 32-bit stores.

$$T_{xfer}^{recon}(n) = 2nf_D M \cdot t_{mem} \text{ cycles} \quad (4.3.3)$$

Eq. 4.3.3 shows the number of cycles required to fetch the matrix data from external memory to on-chip memory via DMA for  $n$  subapertures. Hence, the wavefront reconstruction algorithm is memory-bandwidth limited, not compute-bounded, on the C6701 architecture. It is also the application bottleneck, when compared with the rest (4.1, 4.2).

One possible way to mitigate this bottleneck is to decrease the size of data needed to be fetched for the calculation. This can be done by quantizing the data, i.e. by converting the matrix from the current 32-bit floating point

representation to groups of contiguous (memory-wise) 16-bit integer values where each group has an associated a 32-bit scale factor. A group might be a subcolumn of the original matrix, for instance. This quantization process could reduce the fetch time by ~50% (i.e., the scale factor in Eq. 4.3.3 goes from 2 to unity). But at the same time, it also increases the compute time (Eq. 4.3.1) by 20% (i.e., the factor in Eq. 4.3.1 goes from 1.25 to 1.5) -- hence making the new algorithm compute-bounded while resulting in an overall improvement of 25% over the original time.

$$T_{calc}^{sum}(M) = M + 13 \text{ cycles} \quad (4.3.4)$$

Eq. 4.3.4 shows the vector sum operation for merging the results of Eq. 4.3.1 from all DSPs. This operation is necessary to obtain the true residual vector when multiple DSPs are used to process the pixel data from each channel or when multiple channels are processed simultaneously. As Eq. 4.3.4 shows, for two vectors of  $M$  elements, our implementation provides a single-cycle throughput for each element-to-element addition, the maximum performance achievable with the available on-chip memory bandwidth.

$$T_{xfer}^{sum}(M) = M \cdot t_{comm} \text{ cycles} \quad (4.3.5)$$

Eq. 4.3.5 on the other hand gives the number of cycles required to transfer a vector of  $M$  elements between the on-chip memory areas of two DSPs, where  $t_{comm}$  denotes the average transfer time for one such 32-bit word. Thus,

$$T_{xfer}^{sum}(M) > T_{calc}^{sum}(M) \text{ since } t_{comm} > t_{mem} \quad (4.3.6)$$

Like the wavefront reconstruction, the sum operation is also bandwidth-limited (communication bandwidth).

#### 4.4 Servo Control

Eqs. 4.4.1 through 4.4.3 show the most time-consuming portions of the servo control logic.  $f_1, f_2, f_3$  and  $s_{offset}$  are all constants in these equations. The first loop denoted by Eq. 4.4.1 computes the new DM position  $d_k(t)$  from its current position  $d_k(t-1)$  using the simple proportional-integral control algorithm. This loop is clearly dominated by multiplications (and additions equally). Hence for peak performance, we must utilize the two  $M$  units every cycle.

$$s_k(t) = d_k(t-1) + f_1 \cdot r_k(t-1) + f_0 \cdot r_k(t) \quad \text{for } 1 \leq k \leq M \quad (4.4.1)$$

$$\text{Performance: } \frac{3}{2}M + 12 \text{ cycles}$$

The second loop denoted by Eq. 4.4.2 makes sure that the new DM position is valid, that is,  $d_k(t) \in [d_{min}, d_{max}]$ . This loop is on the other hand dominated by comparisons (the  $S$  units).

$$d_k(t) = \min(d_{max}, \max(d_{min}, s_k(t) + s_{offset})) \quad \text{for } 1 \leq k \leq M \quad (4.4.2)$$

$$\text{Performance: } \frac{5}{4}M + 7 \text{ cycles}$$

Finally, the third loop (Eq. 4.4.3) converts the new valid position into real DM commands  $c_k(t)$  for moving the actuators. As the equation shows, this loop is clearly dominated by memory accesses, and hence bandwidth-limited.

$$c_k(t) = f_2 \cdot d_k(t) \quad \text{for } 1 \leq k \leq M \quad (4.4.3)$$

$$\text{Performance: } \frac{3}{4}M + 12 \text{ cycles}$$

So, summing up the performance of all three equations above, we obtain Eq. 4.4.4 for the servo performance  $T^{servo}$ .

$$T^{servo} = \frac{14}{4}M + 31 \text{ cycles} \quad (4.4.4)$$

## 5. COMPUTE LATENCY FUNCTION

In this section we derive the total compute latency  $T$  of the system as a function of the DSP interconnection topology plus the following basic parameters:

$$M = \# \text{ actuators in the system, which we assumed the same as the number of subapertures in Eq. 5.2.}$$

- $C$  = # channels available for simultaneous readout of pixels from all constituent regions.  
 $P$  = # DSP's used for processing each channel.  
 $f_D$  = ratio of the band to the entire matrix area, a measure of non-sparseness due to the band,  $0 < f_D \leq 1$   
 $t_{pixel}$  = # cycles between pixels (pixel readout period) --  $C$  pixels, one per channel, are output every  $t_{pixel}$  cycles.  
 $t_{comm}$  = # cycles to send one 32-bit word of local on-chip memory to the remote DSP's on-chip memory.  
 $t_{mem}$  = # cycles to load a 32-bit word from external memory to on-chip memory.

For clarity reason, the latency function is also expressed in terms of three additional variables defined below.

- $n$  = # subapertures processed (flat fielded, centroided, and reconstructed) at a time by each DSP.  
 $m$  = # actuator values computed and prefetched at a time during the residual merging.  
 $Q$  = minimum # of pixels that must be received from each channel by all  $P$  DSP's for processing to begin.

These three variables can be derived analytically from the basic parameters. However, their values are much easier obtained after the system is built by either the direct, intrusive, "clear-box" instrumentation method or the indirect, non-intrusive, "black box" measurement described in [7]. Regardless, they should be tuned to minimize the total processing latency. We chose the experimental method because it is easier and more practical. The analytical method would present quite an on-going challenge to completely and accurately count and recount all the cycles -- associated with software pipeline filling and flushing, subroutine calls, and other in-between subroutine glue logic -- every time the affected source code changes.

$$T = T^{resid} + T^{sum} + T^{servo} \quad (5.1)$$

As Eq. 5.1 shows, the total system latency is made up of three basic sequential components (i.e., they must be processed one at a time from left to right in the order shown). The first component  $T^{resid}$  represents the longest processing time required by any processor to compute a residual vector. The middle component  $T^{sum}$  represents the time to sum up the partial results from all processors across all channels to produce one true residual vector. The last component  $T^{servo}$  represents the time to generate the command to the DM and FSM from the final summed vector.

We minimize the latencies of the first two components of Eq. 5.1 by pipelining each component's processing steps (i.e., overlapping foreground computation with background prefetch of data via DMA). This results in two different latency expressions of similar form that consists of three parts. The first part which covers all the terms before the summation term signifies the overhead incurred to fill up the pipeline. The middle part (the  $\sum$  term) signifies the total time expended given the sustainable bandwidth of the pipeline. The last term which immediately followed the summation term signifies the overhead incurred to flush the pipeline.

$$T^{resid} = P \cdot t_{comm} + Q \cdot t_{pixel} + T_{xfer}^{resid}(n) + \sum_{i=2}^{\frac{M}{nPC}} (T_{calc}^{resid}(n) \parallel T_{xfer}^{resid}(n)) + T_{calc}^{resid}(n) \quad (5.2)$$

Eq. 5.2 shows the longest total time  $T^{resid}$  taken by  $P$  processors to process a channel of data. The first two terms account for the time to distribute the first  $Q$  pixels in a frame to all processors so they can begin the calculations. The third term  $T_{xfer}^{resid}(n)$  is the total DMA time taken by each DSP to load the non-pixel data from external memory for  $n$  subapertures. These three terms represent the total overhead incurred to fill up the residual processing pipeline.

As shown in Eq. 5.3,  $T_{xfer}^{resid}(n)$  comprises of the times to load the flat field data, the centroid references, and the reconstruction matrix elements from external memory, plus the data transfer overhead.

$$T_{xfer}^{resid}(n) = T_{xfer}^{flat}(n) + T_{xfer}^{cent}(n) + T_{xfer}^{recon}(n) + T_{xfer}^{overhead}(n) \quad (5.3)$$

Note that since the data transfer occurs in parallel with the pixel arrival and forwarding,  $T_{xfer}^{overhead}(n)$  should account not only for the pixel data transfer time but also for the transfer overhead caused by the interruption of the periodic pixel arrival and the automatic pixel forwarding, which comprised of the following five delays:

1. The time used by the EMIF at the end of an external data access that is often referred to as the CE\_READ\_HOLD cycle count for reads or the CE\_WRITE\_HOLD cycle count for writes.
2. The switching time between accesses by different external memory resources (SBSRAM, SDRAM, bi-FIFO) and direction of accesses.
3. The switching time between external DMA accesses.
4. The time delay between the current DMA burst ends and a new burst begins.
5. The time delay from a DMA synchronization event to the beginning of a data access.
6. The latency between read bursts and write bursts when transferring between locations in internal memory.

We did not include any of these five delays in the calculation of the transfer overhead, as their accurate cycle counts were not available at the time of this writing. However, based on our experience, these numbers are significant and they should be captured accurately in a complete latency or bandwidth analysis.

Hence, if we assumed that the operating pixel rate is sufficiently fast to keep up with the external-to-internal memory transfer of non-pixel data, i.e.,

$$\sum_{i=1}^{\frac{M}{nPC}} (T_{xfer}^{flat}(n) + T_{xfer}^{cent}(n) + T_{xfer}^{recon}(n)) - T_{xfer}^{flat}(n) - T_{xfer}^{cent}(n) \geq \frac{4M}{C} \cdot t_{pixel} \quad \text{or equivalently,}$$

$$\left( \frac{10M + 2f_D M^2}{PC} - 10n \right) \cdot t_{mem} \geq \frac{4M}{C} \cdot t_{pixel} \quad \text{from Eqs. 5.1.3, 5.2.3, 5.3.3.} \quad (5.4)$$

then we should at the minimum approximate the total transfer time required by the primary DSP (the one with physical interface to the camera) per frame processed for each channel as follows:

$$\sum_{i=1}^{\frac{M}{nPC}} T_{xfer}^{resid}(n) \equiv \sum_{i=1}^{\frac{M}{nPC}} (T_{xfer}^{flat}(n) + T_{xfer}^{cent}(n) + T_{xfer}^{recon}(n)) + \frac{4M \cdot 2}{C}$$

$$\equiv \frac{10M + 2f_D M^2}{PC} \cdot t_{mem} + \frac{8M}{C} \quad (5.5)$$

When the exact cycle counts for the said five delays become available, Eq. 5.5 should be updated to reflect the additional cycle counts. As shown, the equation assumes zero interruption overhead. The factor of 2 in the last term accounts for the storing and the forwarding of the raw pixels. Note that Eq. 5.5 does not hold for the last DSP (the farthest from the primary) in the chain of P processors since that DSP receives only  $\frac{1}{P}$  of the pixels the primary receives from the channel.

Eq. 5.6 shows the compute analogue of the DMA transfer time. As shown, it comprised of the computing delays from three calculations, the flat field, the centroid offset, and the wavefront reconstruction.

$$T_{calc}^{resid}(n) = T_{calc}^{flat}(n) + T_{calc}^{cent}(n) + T_{calc}^{recon}(n) \quad (5.6)$$

The summation term in Eq. 5.2 gives the total time to load non-pixel data for the remaining subaperatures to internal memory *and* to perform the necessary calculations on the data. The calculation time for the last  $n$  subaperatures is given by the last term of Eq. 5.2. The binary operator  $\parallel$  is used to connote the parallelism of the background DMA transfer with the foreground CPU calculation. It returns for the function value whichever time greater of the two.

The subtraction of  $10n$  in Eq. 5.4 is to ensure that all the pixels have been received on-chip by the time the second-to-last reconstruction step finished. This additional stipulation on the operating pixel rate, when coupled with the fact 5.7, allows us to start the last flat field and centroid calculation immediately after the second-to-last reconstruction

$$T_{calc}^{flat}(n) + T_{calc}^{cent}(n) < T_{xfer}^{recon}(n) - T_{calc}^{recon}(n) \quad \text{for } M > 20, \text{ given 4.1.2-3, 4.2.4-5, 4.3.2-3.} \quad (5.7)$$

$$T_{calc}^{resid}(n) < T_{xfer}^{resid}(n)$$

completed and finish it before the last matrix data transfer finishes -- thereby reducing the last term of Eq. 5.2 to just the time to reconstruct the wavefront for the last  $n$  subapertures. In other words, Eq. 5.2 is reduceable to Eq. 5.8,

$$T^{resid} = P \cdot t_{comm} + Q \cdot t_{pixel} + \sum_{i=1}^{\frac{M}{nPC}} T_{xfer}^{resid}(n) + T_{calc}^{recon}(n) \quad (5.8)$$

which can be approximated by Eq. 5.9 using Eq. 5.5.

$$T^{resid} \cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25nf_D M + 12n \quad (5.9)$$

where  $\delta = 0$  if both flat field and centroid data fit on-chip, otherwise  $\delta = 10M$ .

Given the DSP network topology in Figures 4.3(a,b), the middle component of the total system latency can be decomposed into two terms. As shown in Eq. 5.10, the first term represents the total time required to sum up the  $P$  residual vectors produced by  $P$  processors assigned to a given channel. The second term represents the time to sum  $C$  vectors from the  $C$  channels produced by the previous term. As usual, for performance reason, we pipelined the processing required of each term. One processor was singled out to compute the sum of two vectors of  $m$  elements long while it prefetched the next  $m$  elements in the background via DMA. The rest of the processors do nothing but forward its residual vector to the next processor.

$$T^{sum} = h(P) + h(C), \text{ where} \quad (5.10)$$

$$h(x) = T_{xfer}^{sum}(m) + \sum_{i=2}^{\frac{(x-1)M}{m}} (T_{calc}^{sum}(m) \parallel T_{xfer}^{sum}(m)) + T_{calc}^{sum}(m) \text{ if } x > 1, \text{ else } h(x) = 0.$$

Applying Eqs. 4.3.6, 4.3.4, and 4.3.5 to 5.10, we obtain Eqs. 5.11 and 5.12.

$$h(x) = \sum_{i=1}^{\frac{(x-1)M}{m}} T_{xfer}^{sum}(m) + T_{calc}^{sum}(m) \quad \text{if } x > 1, \text{ else } h(x) = 0 \quad (5.11)$$

$$h(x) \cong (x-1)M \cdot t_{comm} + m \quad \text{if } x > 1, \text{ else } h(x) = 0 \quad (5.12)$$

It is clear from Eqs. 5.10-12 that as the number of processors ( $P$ ) and channels ( $C$ ) increases, any interconnection topology that scales better than linearly (Figures 3-3), such as a tree, ought to be used. Otherwise, the third latency component  $T^{sum}$  may dominate Eq. 5.1.

Substituting Eqs. 5.8, 5.10, and 4.4.4 into Eq. 5.1, we obtain Eq. 5.13 for the total system latency (5.13)

$$T \cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25nf_D M + 12n + h(P) + h(C) + 3.5M$$

And for  $m \ll M$  and  $n \ll M$ , Eq. 5.13 can be approximated by Eq. 5.14.

$$T \cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25nf_D M + (P + C - 2)M \cdot t_{comm} + 3.5M \quad (5.14)$$

where  $\delta = 0$  if both the flat field and centroid data fit on-chip, otherwise  $\delta = 10M$ . Eqs 5.13 and 5.14 are valid as long as the condition 5.4 is satisfied.

## 6. COMPUTE LATENCY EXAMPLES

In this section, we present two concrete examples in which we work out the details of the latency equation. For the first example where  $m \ll M$  and  $n \ll M$ , we apply Eq. 5.14 and remind ourself that Option 330 of the 4291 board is required for this configuration. For the second example, we use Eq. 5.13 and assume the standard 4291 board is used.

### Example #1

$$\begin{aligned}
 M &= 1600 \text{ actuators.} \\
 C &= 4, \text{ for quad-channel readout, one channel per quadrant.} \\
 P &= 4, \text{ for three DSPs per quadrant.} \\
 Q &= 44, \text{ since it takes this many pixels to start up the pipelines of P processors for each channel.} \\
 k &= 4, \text{ for } 2 \times 2 \text{ subaperature.} \\
 n &= 1, \text{ since only 1 subaperture is processed at a time per DSP.} \\
 m &= 32, \text{ since 32 actuator values are summed/sent at a time.} \\
 t_{mem} &= 1 \text{ cycle, since the single-cycle throughput SBSRAM is large enough to hold all of the data.} \\
 t_{pixel} &= \left(\frac{400}{6}\right) \text{ cycles, for } 2.5 \text{ Mpix/sec., } 6 \text{ ns./cycle.} \\
 t_{comm} &= 9 \text{ cycles, since Bi-FIFO is used for inter-DSP communication.} \\
 f_D &= .25, \text{ for a diagonally-banded reconstructor matrix of } 25\% \text{ denseness.}
 \end{aligned}$$

$$\therefore T = 178670 \text{ cycles} = 1.072 \text{ ms.} \quad \Rightarrow \quad f = \frac{1}{T} = 932 \text{ Hz.}$$

The latency of this configuration can be improved by using a tree-like interconnect topology.

### Example #2

$$\begin{aligned}
 M &= 256 \text{ actuators.} \\
 C &= 4, \text{ for quad-channel readout, one channel per quadrant.} \\
 P &= 1, \text{ for 1 DSPs per channel.} \\
 Q &= 34, \text{ since it takes this many pixels to start up the pipelines of P processors for each channel.} \\
 n &= 1, \text{ since only 1 subaperture is processed at a time per DSP.} \\
 m &= 32, \text{ since 32 actuator values are summed/sent at a time.} \\
 t_{mem} &= 1 \text{ cycle, since the single-cycle throughput SBSRAM is large enough to hold all of the data.} \\
 t_{pixel} &= \left(\frac{400}{6}\right) \text{ cycles, for } 2.5 \text{ Mpix/sec., } 6 \text{ ns./cycle.} \\
 t_{comm} &= 18 \text{ cycles, since Bi-FIFO is used for inter-DSP communication.} \\
 f_D &= 1, \text{ for a full matrix.}
 \end{aligned}$$

$$\therefore T = 50649 \text{ cycles} = 0.304 \text{ ms.} \quad \Rightarrow \quad f = \frac{1}{T} \sim 3.29 \text{ kHz.}$$

Hence, for this example, we are limited by the maximum frame rate supported by the camera which is 2.3 kHz.

## 7. CONCLUSION

In this paper, we have presented a computer architecture of a Shack-Hartman AO system that based on the present-generation TMS320C6701 processors interconnected in a simple grid-like topology. We analyzed the performance of such a system by deriving the compute latency function. We proved that this latency is dominated by the reconstruction of the wavefront and that it is limited by the available memory bandwidth of the processor. We also showed that as the number of processors and channels increase, the interprocessor communication bandwidth may become a contending limiting factor. We offered suggestions how to improve the overall latency. To mitigate the memory bandwidth bottleneck, we proposed data quantization. To reduce the communication latency, we suggested a

network topology with logarithmic scaling characteristics such as a tree. Finally, we pointed out how our analysis can be made more accurate by accounting for the DMA transfer overhead.

## 8. ACKNOWLEDGEMENT

This research was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the California Institute of Technology and the National Aeronautics and Space Administration.

## 9. REFERENCES

- [1] Dekany R., Nelson J., and Bauman B., "Design Considerations for CELT Adaptive Optics", Proceedings of SPIE, 4003, 2000.
- [2] Texas Instruments, Inc., "TMS320C6000 CPU and Instruction Set Reference Guide", Literature No. SPRU189F
- [3] Texas Instruments, Inc., "TMS320C6000 Peripherals Reference Guide", Literature No. SPRU190D.
- [4] Pentek, Inc., Upper Saddle River, New Jersey, USA, "Pentek Models 4290 and 4291 Operating Manual", Rev. C2, Document No. 800.42900.
- [5] SciMeasure Analytical Systems, Inc., "The FiberCam: An Image Acquisition Systems for Adaptive Optics Astronomy".
- [6] DuVarney, Beau C., Motter G., Shaklan S., Kuhnert A., Brack G., Palmer D., Troy M., Kieu T., Dekany R., "EEV CCD39 Wavefront Sensor Cameras for AO and Interferometry", Proceedings of SPIE, 4007, 2000.
- [7] Wang, R., Krishnamurthy, A., Martin R., Anderson T., and Culler D, "Modeling Communication Pipeline Latency", Proceedings of the SIGMETRICS '98/PERFORMANCE '98 Conference, June 1998.

\* [Tuan.N.Truong@jpl.nasa.gov](mailto:Tuan.N.Truong@jpl.nasa.gov); phone 1 818-393-6151; fax 1 818-393-9471; Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA, USA 91109.