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

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

ABSTRACT

Adaptive optics (AO) systems currently under investigation will require at least two orders of magitude increase in the number of actuators, which in turn translates to effectively a 10^4 increase in compute latency. Since the performance of an AO system invariably improves as the compute latency decreases, it is important to study how today's computer systems will scale to address this expected increase in actuator utilization. This paper answers this question by characterizing the performance of a single deformable mirror (DM) Shack-Hartmann natural guide star AO system implemented on the present-generation digital signal processor (DSP) TMS320C6701 from Texas Instruments. We derive the compute latency of such a system in terms of a few basic parameters, such as the number of DM actuators, the number of data channels used to read out the camera pixels, the number of DSPs, 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 that utilizes multiple DMs and guide stars. We demonstrate that the principal performance bottleneck of such a system is the available memory bandwidth of the processors and to lesser extent the IPC bandwidth. This paper concludes with suggestions for mitigating this bottleneck.

Keywords: Adaptive optics, wavefront processors, TMS320C6701

1. INTRODUCTION

Adaptive optics (AO) systems are now considered standard instrument for all ground based telescopes. Using AO, the effects of atmospheric turbulence can be corrected and the benefits of larger telescope apertures realized. Without the benefits of AO, the typical size of an image is limited to the observing wavelength λ divided by the atmospheric coherence length r_0, where r_0 is on the order of 75cm (at λ=1.5μm). However, using AO, diffraction limited images (λ/D) are routinely achieved at wavelengths longer than 1μm. This results in over a factor of ten increase in resolution and enables a factor of 2 (at Palomar) to 4 (at Keck) improvement over the Hubble space telescope resolution. Hence, there is a desire to implement AO systems that operate on larger ground-based telescopes at shorter wavelengths that provide wide fields of view of correction using multiple DMs and guide stars. As expected, these factors all contribute to an increase in computational requirements.

AO systems currently under investigation [1,2] 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. Most of this latency is induced by the reconstruction of the wavefront that is traditionally carried out by the manipulation of a matrix of size N by 2N, where N is the number of actuators. Since the performance of an AO system improves as the compute latency decreases, it is important to investigate how today's computer systems will scale to address the expected increase in actuator utilization. This paper answers this question by characterizing the performance of a single DM Shack-Hartmann natural guide star AO system implemented on the present-generation DSPs TMS320C6701 from Texas Instruments (TI). We derive the overall compute latency of such a system, assuming a simple 1D-mesh (linear array) interconnection topology, 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 that limit its performance. Section 5 presents the total system compute latency. Section 6 derives the latency for two sample AO systems. Section 7 presents the conclusion of this paper.

* 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.

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 one or more DSPs (only one shown in Figure 2.1). Each DSP performs the same processing steps needed in a Shack-Hartmann AO system which include sky-background subtraction, flat fielding, centroiding, and wavefront reconstruction. The results from the reconstruction step are merged (summed), in a pipelined fashion, first from all DSPs within the same channel, then across all the channels - to obtain the true DM residual vector from which the servo commands to the DM and FSM (fast steering mirror) are constructed.

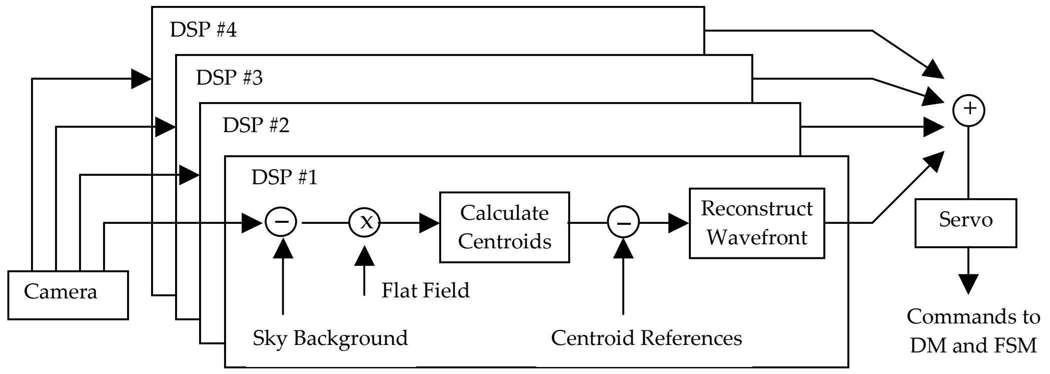


Figure 2.1: Data flow of a wavefront processor taking input from a camera with quad-channel readout. Though figure shows only one, multiple DSPs per channel are often used to further reduce the compute latency.

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 [3]. 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 load/store 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 3 floating point units per data path, the ‘C6701 is capable of up to 6 floating point operations or 2 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.

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 [4]. 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.

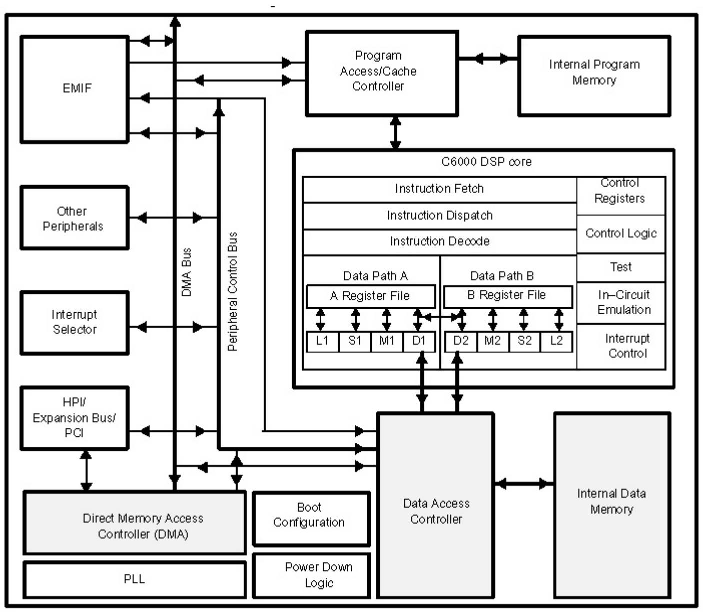


Figure 3.1: TMS320C670x Block Diagram. Courtesy of Texas Instruments,Inc. [4]

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 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 have 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.[5] As Figure 3.2 shows, each board has 4 identical DSPs each equipped with private and shared resources. The following paragraphs briefly describe only the resources used in the application.

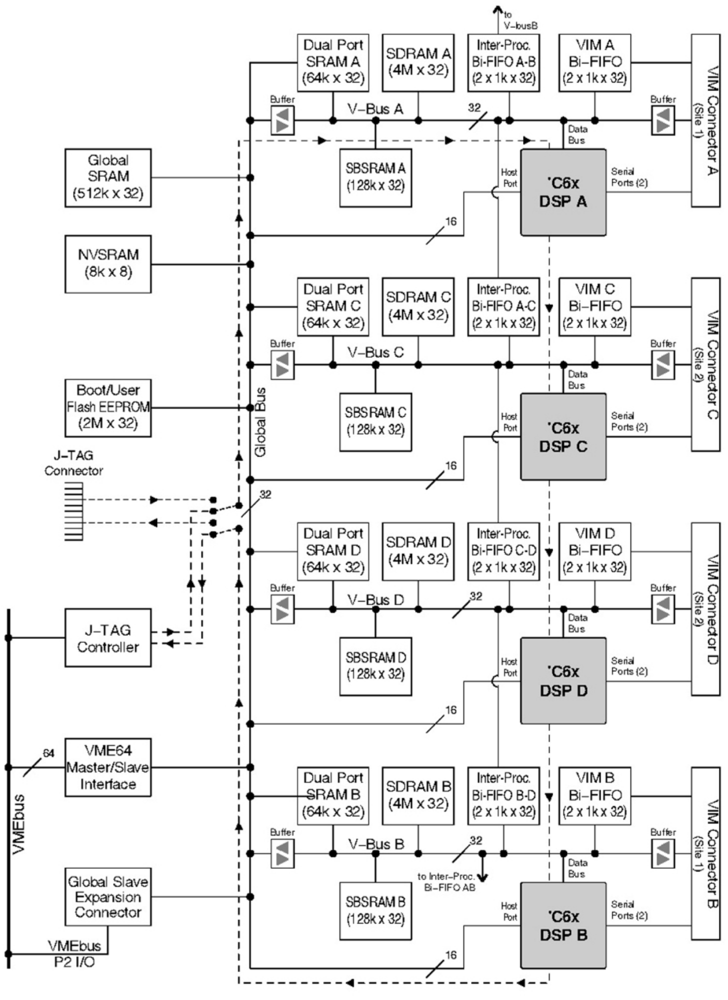

to V-busB
Dual Port SRAM A (64k x 32)
SDRAM A (4M x 32)
Inter-Proc. Bi-FIFO A-B (2 x 1k x 32)
VIM A Bi-FIFO (2 x 1k x 32)
VIM Connector A (Site 1)
V-Bus A
Buffer
Data Bus
Global SRAM (512k x 32)
SBSRAM A (128k x 32)
Host Port
'C6x DSP A
Serial Ports (2)
NVSRAM (8k x 8)
Dual Port SRAM C (64k x 32)
SDRAM C (4M x 32)
Inter-Proc. Bi-FIFO A-C (2 x 1k x 32)
VIM C Bi-FIFO (2 x 1k x 32)
VIM Connector C (Site 2)
V-Bus C
Boot/User Flash EEPROM (2M x 32)
SBSRAM C (128k x 32)
'C6x DSP C
Global Bus
J-TAG Connector
Dual Port SRAM D (64k x 32)
SDRAM D (4M x 32)
Inter-Proc. Bi-FIFO C-D (2 x 1k x 32)
VIM D Bi-FIFO (2 x 1k x 32)
VIM Connector D (Site 2)
V-Bus D
J-TAG Controller
SBSRAM D (128k x 32)
'C6x DSP D
VME64 Master/Slave Interface
Dual Port SRAM B (64k x 32)
SDRAM B (4M x 32)
Inter-Proc. Bi-FIFO B-D (2 x 1k x 32)
VIM B Bi-FIFO (2 x 1k x 32)
VIM Connector B (Site 1)
V-Bus B
to Inter-Proc. Bi-FIFO AB
VMEbus
Global Slave Expansion Connector
SBSRAM B (128k x 32)
'C6x DSP B
VMEbus P2 I/O
32
16
64

Figure 3.2: Functional Block Diagram – Model 4290/4291. Reprinted by permission of Pentek,Inc. [5]

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, and is used to hold data that exceeds the available on-chip memory, primarily the reconstruction matrix. Due to a design exception with the particular ‘C6701 silicon release used by our boards (known in the ‘C6701 silicon errata datasheet as #2.0.4), full speed SBSRAM reads are 2 clocks.

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 accessible by both the VME host computer and the DSPs. We organized this memory area 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 chose not to use the 2 MB global SRAM, which is the other shared resource available, for this function because of its slow access time (35 clocks) and of the potential bus contention created by multiple DSPs on writes.

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 and 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 features an access time of only 2 cycles for 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. [6,7]

3.3 DSP Network Topology

To simplify the analysis, we used a simple 1D-mesh (linear array) topology to interconnect the DSPs. As shown in Figure 3.3(a), we dedicated a 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 merged first using the VIM bi-FIFOs and then across all channels using the IP bi-FIFOs of the last board in the daisy chain.

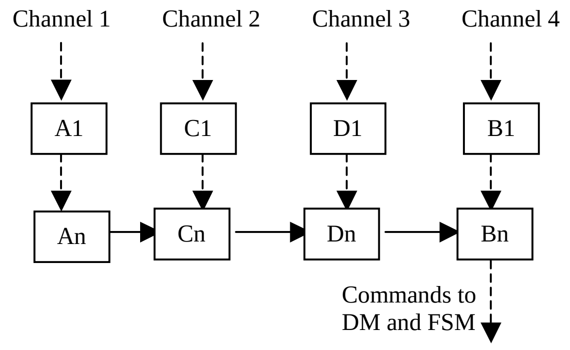
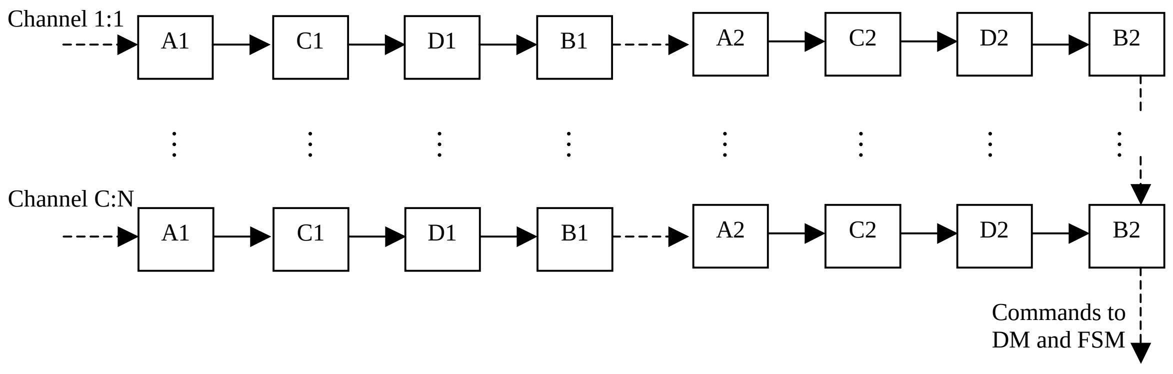


Figure 3.3(a) on the left shows a DSP network of n Pentek 4291-boards (standard model) driving one DM on data from one camera with 4-channel readout and n dedicated DSPs per channel, where n<=4. The dashed arrow utilizes the VIM bi-FIFO path while the solid utilizes the IP bi-FIFO path.

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.



One can extrapolate this interconnection to support multiple cameras with multiple channel readout. Figure 3.3(b) shows a simple 1D-mesh setup consisted of N cameras each with C-channel readout and eight dedicated DSPs per channel. Note that Option 330 of the 4291 is required for this setup (i.e., one with more than four channels total) or for any setup with good scaling characteristic such as a tree or a 3D-hypercube.

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 each algorithm is asymptotically optimal given the architectural constraints described in Section 3.1. The proof is based on the following 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. For this reason, we unrolled (i.e., software-pipelined) all of the compute-intensive loops. The optimal number of times to unroll depends on which of the 'C6701 architectural constraints dominates the loop calculation. On two occasions, we found the theoretical number offered by the dominant constraint impossible to realize (with any valid sequence of instructions) due to two documented (but unknown to us at the time) ‘C6701 silicon errata . For a list of constraints and known silicon errata, please see [3].

4.1 Centroiding

The quadcell centroiding algorithm consists of four loops described by Eqs. 4.1.1 through 4.1.4. The subscript i in these equations represents the loop index that ranges from 1 to n, where n is the number of 2x2 subapertures processed by the loop. The first loop denoted by Eq. 4.1.1 shows how each subaperture’s flat fields are calculated, given the input raw pixels, the pixel offsets (sky background) and the pixel gains. To maximize the available on-chip memory bandwidth, we unrolled the loop 3 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 packets each performed two 64-bit loads and the last two each performed two 32-bit stores. This translates to the theoretical peak performance of 5 cycles per subaperature.

$$p_{i,k} = \left(p_{i,k}^{raw} - p_{i,k}^{offset}\right) \cdot p_{i,k}^{gain} \qquad \text{for } 1 \le k \le 4 \qquad (4.1.1)$$

The second loop denoted by Eq. 4.1.2 calculates the subaperture flux w_i and partial x-y centroid values (u_i, v_i) given the above four flat field values. For performance reason, we pushed the division by the subaperture flux out of Eq. 4.1.2 (where it is normally done) and into Eq. 4.1.4. To keep the two floating point adders busy every cycle, we unrolled the loop 3 times to allow 4 subapertures to be calculated in parallel hence requiring 4*7 adds/subtracts per iteration which can be scheduled with 14 instruction packets each performing 2 adds/subtracts per cycle.

$$\begin{aligned} u_i &= \left(p_{i,2} - p_{i,1}\right) + \left(p_{i,4} - p_{i,3}\right) \\ v_i &= \left(p_{i,2} + p_{i,1}\right) + \left(p_{i,4} + p_{i,3}\right) \\ w_i &= \left(p_{i,2} + p_{i,1}\right) + \left(p_{i,4} + p_{i,3}\right) \end{aligned} \qquad (4.1.2)$$

The third loop denoted by Eq. 4.1.3 calculates nothing more than the reciprocal of a 32-bit floating point value. To obtain the desired 23 bits mantissa accuracy for the reciprocal, the loop performs two iterations of the Newton-Rhapson algorithm on an initial reciprocal estimate (of 8-bit mantissa accuracy) provided by the instruction *_rcpsp*. Then, by vectorizing (i.e., software pipelining n reciprocal operations), we obtained the theoretical peak performance of 2 cycles per reciprocal operation, which is significantly faster than the non-vectorized implementation of 27 cycles from TI.

$$w_i^{-1} = q_{i,2}, \text{ where } q_{i,j} = q_{i,j-1} \cdot \left(2 - w_i \cdot q_{i,j-1}\right) \text{ and } q_{i,0} = \_\,rcpsp\left(w_i\right) \qquad (4.1.3)$$

The fourth loop denoted by Eq. 4.1.4 calculates the x-y centroid offsets, given the fixed centroid references and previous two loops' results. By unrolling the loop 3 times, we obtained the peak performance of 9 cycles per 4 subapertures.

$$\begin{aligned} x_i &= \min(x_{\max}, \max(x_{\min}, u_i \cdot w_i^{-1} - x_i^{ref})) \\ y_i &= \min(y_{\max,} \max(y_{\min}, v_i \cdot w_i^{-1} - y_i^{ref})) \end{aligned} \quad \text{if } w_i >= w_{\min}\text{, else } x_i = y_i = 0. \tag{4.1.4}$$

Eq. 4.1.5 gives the combined performance of the four loops for *n* subapertures . The constant 48 is the total overhead incurred by the filling and flushing of all four software pipelines. Eq. 4.1.6 gives the total time to load the required input data (the pixel offsets, pixel gains, and the x-y centroid references) from external memory with t_{mem} denoting the sustained transfer time per 32-bit word. Unlike Eq. 4.1.5, Eq. 4.1.6 does not include any software pipeline overhead because the DMA setup code that loads the data executes only once per frame. It is assumed that the on-chip memory is large enough to hold all of the raw pixels coming to each DSP.

$$T_{cpu}^{cent}(n) = \tfrac{51}{4} n + 48 \text{ cycles} \tag{4.1.5}$$

$$T_{dma}^{cent}(n) = 10n \cdot t_{mem} \text{ cycles} \tag{4.1.6}$$

4.2 Wavefront Reconstruction

Eq. 4.2.1 shows the pipelined implementation of the general matrix vector multiplication. By parameterizing the lower and upper limits of the loop index k, the algorithm also supports vector multiplication with a band matrix efficiently. The factor f_D in the equation denotes the ratio of the diagonal band to the entire Mx2M matrix R area or loosely speaking a measure of sparseness. By definition, the value ranges between 0 and 1, with 1 representing the general full matrix. M denotes the number of actuators (which we assumed the same as the number of subapertures) in the system.

$$r_k = r_k + R_{k,2j-1} \cdot x_j + R_{k,2j} \cdot y_j \qquad \text{for } 1 \le j \le n,\ 1 \le k \le M \tag{4.2.1}$$

To obtain the peak performance given by Eq. 4.2.2, we unrolled 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 instruction packets, one per cycle, where three of the packets each perform two 64-bit loads and the remaining two each perform two 32-bit stores.

Eq. 4.2.3 shows the total time required to fetch the matrix data from external memory via DMA for *n* subapertures. Clearly, the performance of the wavefront reconstruction is limited by the external memory bandwidth. It is the principal performance bottleneck, when compared to the rest of the application,

$$T_{cpu}^{recon}(n) = \left(1.25 f_D M + 12\right) \cdot n + 7 \text{ cycles} \tag{4.2.2}$$

$$T_{dma}^{recon}(n) = 2n f_D M \cdot t_{mem} \text{ cycles} \quad \Rightarrow \quad T_{dma}^{recon}(n) \approx 2 t_{mem} T_{cpu}^{recon}(n) \tag{4.2.3}$$

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 matrix 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 floating point scale factor. A group might be a subcolumn of the original matrix, for instance. This quantization would reduce the fetch time by ~50% (i.e., the factor 2 in Eq. 4.2.3 goes to unity) and increase the compute time by 20% (i.e., the factor 1.25 in Eq. 4.2.2 goes to 1.5), resulting in an overall 25% improvement over the original time.

Eq. 4.2.4 shows the theoretical peak performance attained by our implementation for summing two residual vectors. Eq. 4.2.5 gives the total time to transfer an M-element vector between two DSPs' on-chip memory areas. t_{comm} denotes the sustained transfer time per 32-bit element; and since it is greater than 1, IPC bandwidth is the performance bottleneck.

$$T_{cpu}^{sum}(M) = M + 13 \text{ cycles} \tag{4.2.4}$$

$$T_{dma}^{sum}(M) = M \cdot t_{comm} \text{ cycles.} \qquad \Rightarrow \qquad T_{dma}^{sum}(M) > T_{cpu}^{sum}(M) \tag{4.2.5}$$

4.3 Servo Control

Eqs. 4.3.1-3 show all compute-intensive loops involved in servo control. The first loop denoted by Eq. 4.3.1 computes the new DM position vector $d_k(t)$ from its current position $d_k(t-1)$, for $1 \le k \le M$, given the new and current DM residuals $r_k(t), r_k(t-1)$ and the two proportional-integral constants c_0 and c_1. This loop also computes the new DM average position which, for clarity reason, appears in Eq. 4.3.2 as the summation term. Dominated by additions, this loop has nevertheless an attainable, theoretical peak performance of 3 cycles per pair of actuator values.

$$s_k(t) = d_k(t-1) + c_1 \cdot r_k(t-1) + c_0 \cdot r_k(t) \tag{4.3.1}$$

The second loop denoted by Eq. 4.3.2 ensures that the new DM position is valid, that is, $d_k(t) \in [d_{\min}, d_{\max}]$.

$$d_k(t) = \min(d_{\max}, \max(d_{\min}, s_k(t) + d_{offset} - \frac{1}{M}\sum_k s_k(t))) \tag{4.3.2}$$

The last loop (Eq. 4.3.3) converts the DM position vector into DM commands $z_k(t)$ for moving the actuators. $f(k)$

$$z_k(t) = f(k) \mid ((d_k(t) \cdot c_2) \,\&\, c_3) \tag{4.3.3}$$

represents the actuator map in this equation. Due to a design exception (known in TI silicon errata datasheet as #0.0.15) with the particular 'C6701 silicon release used by our boards, we could not realize what otherwise single-cycle throughput performance for the last two loops. Summing up all three loops, we obtain 4.3.4 for the servo performance.

$$T^{servo} = 3.625M + 50 \text{ cycles} \qquad (3.5M + 50 \text{ cycles for silicon release 1.x or later}) \tag{4.3.4}$$

5. LATENCY ANALYSIS

In this section we derive the total compute latency T of the system as a function of seven basic parameters shown below. Due to the large number of actuators employed in future AO systems and insufficient total on-chip memory, we can and will assume that the reconstructor matrix reside entirely off-chip. To further simplify the analysis, we will also assume a simple 1D-mesh (linear array) topology for interconnecting the DSPs. When possible, a more scalable topology should be used to reduce the latency.

M	=	# actuators in the system, which we assumed the to be equal to the number of subapertures.
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 sparseness due to the band, $0 < f_D \le 1$
t_{pixel}	=	# cycles between pixels (pixel readout period) -- C pixels (one per channel) are output every t_{pixel} cycles.
t_{comm}	=	# cycle to send a 32-bit word between two DSPs' on-chip memory at sustainable (pipelined) rate.

t_{mem} = # cycles to load a 32-bit word on-chip from external memory at sustainable (pipelined) rate.

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

n	= # subapertures processed (centroided, and wavefront-reconstructed) at a time by each DSP.
m	= # actuator values computed and prefetched at a time during the residual merging.
Q	= # of pixels comprised the first P subapertures from each channel for P DSPs to begin processing.

The optimal value of these last three variables can be derived analytically from the basic equations. However, their values are much easier obtained after the system is built by either the direct, instrusive, "clear-box" instrumentation method or the indirect, non-intrusive, "black box" measurement described in [8]. Regardless, they should be tuned to minimize the total latency. We chose the experimental method because it is easier and more practical. The analytical method would require a complete and accurate recount of 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.

$$T = T^{resid} + T^{sum} + T^{servo} \tag{5.1}$$

As Eq. 5.1 shows, the total system latency is made up of three sequential components (i.e, they must be processed from left to right in the order shown). The first component represents the processing time required to compute a DM residual vector. The middle component represents the time required to sum up the DM residuals from all DSPs. The last component represents the total time to generate the DM and FSM commands from the final DM residual vector. We minimize the latencies of the first two components of Eq. 5.1 by pipelining each component's processing steps (i.e., by overlapping the calculation with background data prefetch via DMA).

Eq. 5.2 shows the total processing time taken by P processors per channel. 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 is the total DMA time taken by each DSP to load the non-pixel data from external memory for n subaperatures. These three terms represent the total overhead incurred to fill up the DM residual processing pipeline. The middle part (the summation term) denotes the total time expended given the sustainable bandwidth of the pipeline. The last term denotes the overhead incurred to flush the pipeline.

$$T^{resid} = P \cdot t_{comm} + Q \cdot t_{pixel} + T_{dma}^{resid}(n) + \sum_{i=2}^{\frac{M}{nPC}} \left(Max(T_{cpu}^{resid}(n) \parallel T_{dma}^{resid}(n)) \right) + T_{cpu}^{resid}(n) \tag{5.2}$$

where $T_{cpu}^{resid}(n) = T_{cpu}^{cent}(n) + T_{cpu}^{recon}(n)$

$$T_{dma}^{resid}(n) = T_{dma}^{cent}(n) + T_{dma}^{recon}(n) + T_{dma}^{overhead}(n)$$

Eq. 5.2 assumes that the pixel rate of the camera is sufficiently fast to keep up with the external-to-internal memory transfer of non-pixel data (pixel gains, pixel offsets, x-y centroid references, and the reconstruction matrix), i.e.,

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

$$\left(\frac{10M + 2 f_D M^2}{PC} - 10n \right) \cdot t_{mem} \ \geq\ \frac{4M}{C} \cdot t_{pixel} \quad . \tag{5.3}$$

In other words, the raw pixels must arrive on-chip in time to enable the centroid calculation to complete by the time the DMA transfer of the matrix data finishes, for every n subapertures. Coupled this with $T_{cpu}^{resid}(n) < T_{dma}^{resid}(n)$ which is true for all M values (in practice) from Eqs. 4.1.5-6 and 4.2.2-3, Eq. 5.2 is now reduceable to Eq. 5.4.

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

$$\cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2 f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25 n f_D M + 12n \qquad (5.4)$$

where $\delta = 0$ if all data required for centroiding could fit on-chip, otherwise $\delta = 10M$.

$\sum T_{dma}^{overhead}(n)$ represents the total intrinsic DMA transfer overhead incurred to bring external data on-chip and to forward the raw pixels. It comprises the following delays – none was, however, included in this analysis since their exact cycle counts were not available at the time:

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,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.

Using the interconnection topology in Figures 3.3(a,b), T^{sum} can be decomposed into two terms. As shown in Eq. 5.5, the first term represents the total time required to sum up the P residual vectors produced by P processors from each channel. The second term represents the time to combine C vectors from the C channels. As usual, for performance reason, we pipelined the processing required of each term. The last processor in a daisy-chain computes the sum of two m-element sub-vectors while it prefetches the next m elements in the background via DMA. The rest of the processors do nothing but forward its residual vector to the next in a daisy-chain.

$$T^{sum} = h(P) + h(C), \quad \text{where} \qquad (5.5)$$

$$h(x) = T_{dma}^{sum}(m) + \sum_{i=2}^{\frac{(x-1)M}{m}} \left(T_{cpu}^{sum}(m) \parallel T_{dma}^{sum}(m)\right) + T_{cpu}^{sum}(m)$$

$$= \sum_{i=1}^{\frac{(x-1)M}{m}} T_{dma}^{sum}(m) + T_{cpu}^{sum}(m) \qquad \text{if } x > 1, \text{ else } h(x) = 0.$$

$$\cong (x-1)M \cdot t_{comm} + m$$

It is clear from Eqs. 5.6 that as the number of processors (P) and channels (C) increases, any interconnection topology that scales better than 1D-mesh (Figures 3-3), such as a tree, ought to be used. Otherwise, Eq. 5.1 may be dominated by the IPC bandwidth. Substituting Eqs. 5.4, 5.5 and 4.3.4 into Eq. 5.1, we obtain Eq. 5.6 for the total system latency.

$$T \cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2 f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25 n f_D M + 12n + h(P) + h(C) + 3.625M \qquad (5.6)$$

For $m << M$ and $n << M$, Eq. 5.6 can be approximated by Eq. 5.7.

(5.7)

$$T \cong P \cdot t_{comm} + Q \cdot t_{pixel} + \frac{2 f_D M^2 + \delta}{PC} \cdot t_{mem} + \frac{8M}{C} + 1.25 n f_D M + \left(P + C - 2\right) M \cdot t_{comm} + 3.625M$$

where $\delta = 0$ if all centroid input data fit on-chip, else $\delta = 10M$. Eqs 5.6 and 5.7 are valid as long as Eq. 5.3 holds.

6. LATENCY EXAMPLES

In this section, we present two concrete examples in which we work out the details of the latency function. In the first example, we take the current compute problem for Keck or Palomar AO systems and apply Eq. 5.6 assuming the standard 4291 board is used.

M	=	256 actuators.
C	=	4, for quad-channel readout, one channel per quadrant.
P	=	1, for 1 DSPs per channel.
f_D	=	1, for a full matrix.
t_{mem}	=	2 cycle, since the single-cycle throughput SBSRAM is large enough to hold all of the data.
t_{comm}	=	9 cycles, since P=1 (and thus no pixel forwarding).
t_{pixel}	=	$\left(\frac{400}{6}\right)$cycles, for 2.5Mpix/sec., 6 ns./cycle.
δ	=	0, since all input data required for centroiding fit well within the total on-chip memory of P*C DSPs.
n	=	1, since only 1 subaperture is processed at a time per DSP.
m	=	32, since 32 actuator values are summed/sent at a time.
Q	=	18, since the first P subapertures from each channel are made of this many pixels (2P+16) for our camera.

$\therefore \quad T$ = 75461 cycles ~ 0.453 ms. $\Rightarrow \quad f = \frac{1}{T}$ ~ 2.2 kHz.

Hence, the maximum frame rate for this example is 2.2 kHz, a factor of several better than the current systems, using only 4 DSPs.

In the next example, we consider a 1600 actuator AO system in which we use a total of 16 DSPs and assume a sparseness in the reconstructor of 25%. In this case, $m << M$ and $n << M$, and we apply Eq. 5.7 and remind ourselves that Option 330 of the 4291 board is required for this configuration

M	=	1600 actuators.
C	=	4, for quad-channel readout, one channel per quadrant.
P	=	4, for three DSPs per quadrant.
f_D	=	.25, for a diagonally-banded reconstructor matrix of 25% denseness.
t_{mem}	=	2 cycles (as we assumed that Option 330 and the Standard model use the same ‘C6701 silicon release).
t_{comm}	=	4 cycles, for loading and forwarding the pixels since P>1.
t_{pixel}	=	$\left(\frac{400}{6}\right)$cycles, for 2.5Mpix/sec., 6 ns./cycle.
δ	=	0, since all input data required for centroiding fit well within the total on-chip memory of P*C DSPs.
n	=	1, since only 1 subaperture is processed at a time per DSP.
m	=	32, since 32 actuator values are summed/sent at a time.
Q	=	24, since the first P subapertures from each channel consisted of this many pixels (2P+16) for our camera.

$\therefore \quad T$ = 209516 cycles ~ 1.257 ms. $\Rightarrow \quad f = \frac{1}{T}$ ~ 790 Hz. (~1.2 kHz if t_{mem} = 1)

The compute latency of this configuration can be improved by using a more scalable topology.

7. CONCLUSION

In this paper, we have presented a computer architecture of a Shack-Hartmann AO system that is based on the present-generation TMS320C6701 processors interconnected using a simple 1D-mesh 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 increases, the interprocessor communication bandwidth may become the limiting factor. We offered suggestions on how to improve the overall latency. To mitigate the memory bandwidth bottleneck, we proposed data quantization. To reduce the communication latency, we suggested a more scalable interconnection topology such as a tree. Finally, we pointed out how our analysis can be made more accurate by accounting for the intrinsic DMA transfer overhead.

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. The authors wish to thank Samuel Igwe of Pentek Inc. for the valuable support provided during our development.

REFERENCES

[1] Dekany R., Nelson J., and Bauman B., “Design Considerations for CELT Adaptive Optics”, Proceedings of SPIE, 4003, 2000.

[2] Macintosh, B., Olivier, S., Bauman, B., Brase, J., Carr, E., Carrano, C.; Gavel, D.; Max, C.; Patience, J., “Practical high-order adaptive optics systems for extrasolar planet searches”, Proc. SPIE Vol. 4494, 2002.

[3] Texas Instruments, Inc., “TMS320C6000 CPU and Instruction Set Reference Guide”, Literature No. SPRU189F

[4] Texas Instruments, Inc., “TMS320C6000 Peripherals Reference Guide”, Literature No. SPRU190D.

[5] Pentek, Inc., Upper Saddle River, New Jersey, USA, “Pentek Models 4290 and 4291 Operating Manual”, Rev. C2, Document No. 800.42900.

[6] SciMeasure Analytical Systems, Inc., “The FiberCam: An Image Acquisition Systems for Adaptive Optics Astronomy”.

[7] 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.

[8] 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.