ABSTRACT

Title of Document: DISTRIBUTED TWO-DIMENSIONAL FOURIER TRANSFORMS ON DSPs: WITH APPLICATIONS FOR PHASE RETRIEVAL

Jeffrey Scott Smith, Master of Science, 2006

Directed By: Professor Bruce Jacob
Department of Electrical Engineering

Many applications of two-dimensional Fourier Transforms require fixed timing as defined by system specifications. One example is image-based wavefront sensing. The image-based approach has many benefits, yet it is a computational intensive solution for adaptive optic correction, where optical adjustments are made in real-time to correct for external (atmospheric turbulence) and internal (stability) aberrations, which cause image degradation.

For phase retrieval, a type of image-based wavefront sensing, numerous two-dimensional Fast Fourier Transforms (FFTs) are used. To meet the required real-time specifications, a distributed system is needed, and thus, the 2-D FFT necessitates an all-to-all communication among the computational nodes. The 1-D floating point FFT is very efficient on a digital signal processor (DSP). For this study, several architectures and analysis of such are presented which address the all-to-all communication with DSPs. Emphasis of this research is on a 64-node cluster of Analog Devices TigerSharc TS-101 DSPs.

DISTRIBUTED TWO-DIMENSIONAL FOURIER TRANSFORMS ON DSPs: WITH AN APPLICATIONS FOR PHASE RETRIEVAL

By

Jeffrey Scott Smith

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2006

Advisory Committee:
Professor Bruce Jacob Chair
Professor Steven Tretter
Professor Donald Yeung

Dedication

A special thank you to Heather, and Mr. Five Stars.

Acknowledgements

The author would like to thank Professors Bruce Jacob, Shuvra Bhattacharyya, Steven Tretter, and Donald Yeung for their time and energies with serving on the Graduate committee reviewing this thesis.

The author would like to thank the NASA Goddard Space Flight Center generally and the members of the Optics Branch specifically for the support of this work and use of facilities. Thank yous and acknowledgements are given to Bruce Dean, David Aronstein, and Ron Shiri for the many useful technical insights and discussions and moral support. Specific appreciation is given to Bruce Dean for his original development of and support for the MGS algorithm. The author would like to thank Semion Kizhner and Edward Lo for their assistance in the Analog Devices 21160M system, described in Chapter 2. The author thanks Lee Feinberg, Scott Acton, Peter Maymon, Ray Bouracut, and Kim Mehalick for discussion and support of this work.

The author would like to thank his parents and family, with specific mention of Austin with his, “How’s Mars?” questions.

Forward

The author would like to point out the significance of this research. The results from this research have become a critical component for NASA, with a specific application for the James Webb Space Telescope (JWST), a space telescope under development that is scheduled to launch no earlier than June 2013. The Testbed Telescope (TBT), a 1/6th scale model of JWST, has been built to be a critical scientific study to demonstrate TRL-6 (Technology Readiness Level) for JWST.[1] The results from this research and a working prototype, as described in Chapter 4, are integrated into the TBT and are considered by the JWST project management to be essential to meeting the January 2007 deadline for this readiness level.

Table of Contents

Dedication ii
Acknowledgements iii
Forward iv
Table of Contents v
List of Tables vii
List of Figures viii
List of Illustrations x
Chapter 1 : Introduction 1
- Motivation 1
- Application 3
- Phase Retrieval Algorithm 6
- Current Solutions 11
- Digital Signal Processors 13
- Systems of Multiple DSPs 19
- Graph Theory 23

Chapter 2 : Phase Retrieval on the ADSP-21160M 31
- Algorithm Analysis 31
- General Optimizations for CPU and DSPs 35
- Methodology 38
- Results 39

Chapter 3 : Phase Retrieval on the TigerSharc 101 42
- Methodology 42
- Computational Precision and Accuracy 43
- TS-101: 1, 4, and 8-node 46
- TS-101: 16-node Architectures 48
- TS-101: 16-node Architectures Analysis 54
- TS-101: 32-node and 64-node Architecture 57

Chapter 4 : Phase Retrieval on the TigerSharc 201 60
- Hybrid Diversity Algorithm 60
- TS-201: 4-node Architecture 61

Chapter 5 : Conclusions 63
- Summary 63
- Future Work 64

Acronyms 66
References and Bibliography 67

List of Tables

Tbl. 2.1: Timing for 1-D FFT and 2-D FFT for 256 and 512 image size on a single 21160M.
Tbl. 3.1: Scalability for image size on Fig. 3.6.
Tbl. 3.2: Timing for various image sizes on three architectures, with 4 Diversity-Defocus images after convergence (seconds).

List of Figures

Fig. 1.1: Computation of the 2-D DFT using 1-D FFT building blocks.
Fig. 1.2: Various 2-D FFT Libraries: PowerPC 970, single-precision complex input/output.[2]
Fig. 1.3: Wavefront sensing and Control Block Diagram.
Fig. 1.4: Block Diagram of focus diversity phase retrieval with two defocus images.[]
Fig. 1.5: Block Diagram of MGS using Iterative Transform (ITA) phase-retrieval.
Fig. 1.6: Block Diagram of the Analog Devices Sharc 21160M. [20]
Fig. 1.7: Block Diagram Analog Devices TigerSharc-TS-101.[20]
Fig. 1.8: Block Diagram Analog Devices TigerSharc-TS-201. [20]
Fig. 1.9: Block Diagram of 4 Node cluster of ADSP-21160M.
Fig. 1.10: Block Diagram of 4 Node cluster of TS-101.
Fig. 1.11: Block Diagram of 16 Node cluster of TS-101.
Fig. 1.12: Block Diagram of link port connection between cPCI and PMC daughter card.
Fig. 1.13: Block Diagram of 64 Node system of TS-101 with host computer.
Fig. 1.14: Distributed Transpose on 4 DSPs: Columns represent local memory, and colors represent data.
Fig. 1.15: Two graphs demonstrating graph diameter: (a) 6 Node Ring, graph diameter is 3, (b) 8 Node hypercube, graph diameter is 3, distance from N_0 to N_6 is 3, (Node label $N_{id,\ d(0,id)}$).
Fig. 1.16: Steps for constructing a graph with minimum diameter: (a) 10 Node base graph, with 3 edges per node and 6 leaves (b) Peterson Graph: graph diameter is 2.
Fig. 1.17: Base architectures for several cases: (a) Base architecture for 22 node, 3 branches per node, optimal diameter of 3 (b) Base architecture for 17 nodes, 4 branches per node, optimal diameter 2 (c) Base architecture of 46 node, 3 branches per node, with optimal diameter of 3.
Fig. 1.18: Levi graph, 30 nodes and diameter 4. [27]
Fig. 2.1: Timing for the phase retrieval algorithm in MATLAB: (a) Entire algorithm, (b) details for step 2, with a single iteration.
Fig. 2.2: Measured defocus PSF data from the Wavefront Control Testbed (WCT) at Goddard.
Fig. 2.3: Phase estimate from the MGS algorithm: (a) Mesh plot, and (b) colored image plot.
Fig. 2.4: Recovered PSF data from the MGS algorithm.
Fig. 2.5: Timing for the phase retrieval algorithm in core iterative algorithm after mathematical modification.
Fig. 2.6: Timing code for 2-D FFTW with and without the second transpose.
Fig. 2.7: Optimization of 2-D FFT for padding.
Fig. 2.8: Timing for MGS algorithm in MATLAB and on the 21160M DSP system.
Fig. 3.1: Phase estimate after single iteration, MATLAB (a), DSP (b), the subtraction of the two (c), and the log stretch of the normalized subtraction of the two (d).

Fig. 3.2: Phase estimate after algorithm convergence, MATLAB (a) the DSP (b), the subtraction of the two (c), and the log stretch of the normalized subtraction of the two (d).
Fig. 3.3: Two 8 DSPs grid architectures, hypercube (a) and 1-Möbius cube (b).
Fig. 3.4: 16 DSP architecture, inefficient use of the PCI bus for all-to-all communication.
Fig. 3.5: 16 DSP architecture, minimize graph diameter between all 16 DSPs.
Fig. 3.6: 16 DSP architecture: minimize graph diameter between clusters.
Fig. 3.7: 16 DSP architecture: use PMC cards, and create uni-directional link ports.
Fig. 3.8: Four images (a) on 64 DSPs (b) with 16 DSPs per image to produce a single phase estimate (c).
Fig. 3.9: Four images (a) on 16 DSPs (b) with 16 DSPs per image to produce a single phase estimate (c).
Fig. 3.10: 32 DSP architecture.
Fig. 3.11: 64 DSP architecture.
Fig. 4.1: Block diagram for the HDA.
Fig. 4.2: TS-201 4 DSP architecture.

List of Illustrations

Ill. 1.1: Segmented primary aperture examples: (a) The Testbed Telescope (TBT) is a 1/6th scale model of the planned James Webb Space Telescope mirror with 18 segments in the primary (b) W.M.Keck 1 Observatory with 36 segments in the primary.[4][5]

Chapter 1: Introduction

Motivation

For many image-processing algorithms a two-dimensional (2-D) discrete Fourier transform (DFT) is required. In all but a few rare cases, the Fourier transform makes use of the Fast Fourier Transform (FFT). The 2-D DFT has a many-to-one mapping, and thus for each element in the output, every element of the input is required. The 1-D DFT and inverse 1-D DFT can be seen in (Eq. 1.1) and (Eq. 1.2). Similarly, the 2-D DFT and inverse 2-D DFT can be seen in (Eq. 1.3) and (Eq. 1.4) respectively.

$$F[u] = \sum_{x}^{N-1} f[x] \cdot e^{\frac{-2\cdot\pi\cdot i\cdot u\cdot x}{N}} \qquad \text{(Eq. 1.1)}$$

$$f[x] = \frac{1}{N}\sum_{u}^{N-1} F[u] \cdot e^{\frac{2\cdot\pi\cdot i\cdot u\cdot x}{N}} \qquad \text{(Eq. 1.2)}$$

$$F[u,v] = \sum_{y=0}^{N-1}\left(\sum_{x=0}^{N-1} f[x,y] e^{-\frac{2\cdot\pi\cdot i\cdot x\cdot u}{N}}\right) \cdot e^{-\frac{2\cdot\pi\cdot i\cdot y\cdot v}{N}} \qquad \text{(Eq. 1.3)}$$

$$f[x,y] = \frac{1}{N^2}\sum_{v}^{N-1}\left(\sum_{u}^{N-1} F[u,v] \cdot e^{\frac{2\cdot\pi\cdot i\cdot u\cdot x}{N}}\right) \cdot e^{\frac{2\cdot\pi\cdot i\cdot v\cdot y}{N}} \qquad \text{(Eq. 1.4)}$$

To compute the 2-D DFT, the function $F[u, v]$ can be constructed by using the building blocks of the 1-D FFT. For this, the 1-D FFT is performed on each row, i.e. compute $F[u, y]$. Then, the 1-D FFT is performed on column of the result. This is illustrated in Fig. 1.1.

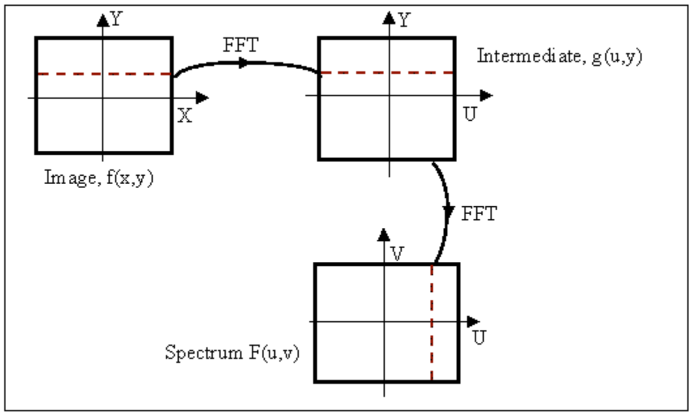


Fig. 1.1: Computation of the 2-D DFT using 1-D FFT building blocks.

To access data in column format and row format on a shared memory requires a transpose of the data. For high-level programming, the transpose of a matrix is straightforward, i.e. `f[x][y] = f[y][x]`. Yet, for a general-purpose central processing unit (CPU), an efficient transpose is non-trivial due to cache and memory constraints. Given a large enough image data size $N_{size}{\times}N_{size}$, a conventional cache will not be able to efficiently access both cases of N_{size} sequential elements and N_{size} elements that are strided by N_{size}. For this reason, the FFT for 2-D has a non-linear performance with respect to image size on a general purpose CPU, see Fig. 1.2.[2] Specifically, the 2-D FFT sees a significant drop-off with respect to performance for images of grid size 512×512 or larger. This decrease in performance is in addition to the performance decrease for the larger image size.

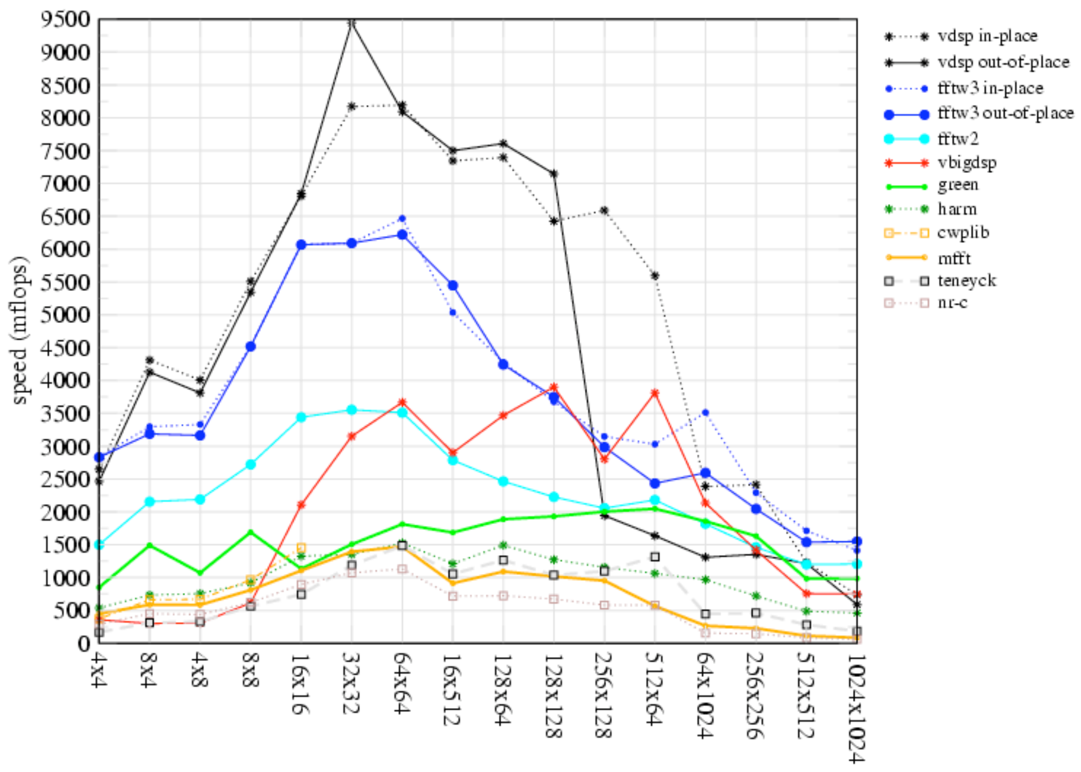


Fig. 1.2: Various 2-D FFT Libraries: PowerPC 970, single-precision complex input/output.[2]

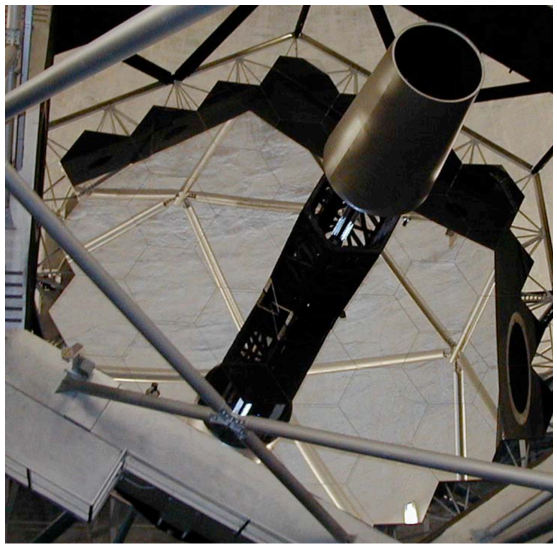
Furthermore, the FFT is optimal in performance for data that is a power of two. Yet, for the general purpose CPU, the set associative cache will miss frequently when accessing data that is strided by large powers of two.[3] For this reason, the configurable cache is desirable, such as seen in many DSPs, which is already an optimal choice for a 1-D FFT.

Application

In this section, an overview of the optical problem is presented. This justifies the motivation for this research without required a deep understanding of the details.

Many next-generation scientific optical systems have requirements that necessitate adaptive optic control. The use of adaptive optics is an engineering technique where optical components are moved in a feedback loop to optimize performance. The errors can be internal, such as optical figure error, jitter, and thermal instability or external, such as atmospheric turbulence between the object and the imaging system. These corrections can be made either with a deformable mirror, where the mirror itself is deformable such as a thin membrane, or with as a segmented primary mirror, where several rigid-bodied mirrors are pieced together to form a larger surface mirror as seen in Ill. 1.1.[4][5] For optical simplicity and efficiency, image-based wavefront sensing is preferable over a conventional interferometer-based wavefront sensor.

(a)


(b)

Ill. 1.1: Segmented primary aperture examples: (a) The Testbed Telescope (TBT) is a 1/6th scale model of the planned James Webb Space Telescope mirror with 18 segments in the primary (b) W.M.Keck 1 Observatory with 36 segments in the primary.[4][5]

For a monochromatic point object that is imaged through an optical system, the light observed in the image plane is called the point spread function (PSF) p(x,y). The PSF is the modulus squared of the coherent system function h(x,y):

$$p(x,y) = \left|h(x,y)\right|^2 \qquad \text{(Eq. 1.5)}$$

The Fourier transform of h(x,y), using (Eq. 1.3) is H(u,v), the coherent system function for the optical system function. The H(u,v) is a complex valued function represented by the pupil function A(u,v) and the phase θ(u,v).

$$H(u,v) = A(u,v) \cdot e^{i \cdot \theta(u,v)} \qquad \text{(Eq. 1.6)}$$

phase in REF _Ref150402835 \h (Eq. 1.6) is proportional to the optical wavefront. For an ideal optical system, with an optimal PSF, the wavefront is identically zero and thus θ(u,v) = 0. For real systems, the wavefront is nonzero and the phase is measured with a wavefront sensor. Then, a correction to the wavefront using the deformable mirror must be made that induces the negative of the current wavefront, which attempts to drive the system wavefront to zero.[6][7]

The fundamental problem is that the wavefront cannot be directly measured from the imaging system. This is because the detector, such as the CCD, observes the PSF. Thus, the phase is lost when the modulus squared is taken, (Eq. 1.5). A simple way to recognize the problem is to note that only intensity data is known for a Fourier conjugate pair, and it is desired to know the phase data for one or both of the pairs. One solution to the problem is adding additional optics to create an interferometer-based wavefront sensor or a Shack-Hartmann sensor. Both of these solutions have been used in practice but have known limitations.[8][9][10] Such solutions require a wavefront reference, introduce non-common path errors, and generally put strong demands on the number and quality of optical components used. For optical simplicity and efficiency, image-based wavefront sensing is preferable over the conventional optical solution. As a result, the image-based approach, while more demanding computationally, is generally less complex to implement in hardware and more attractive from payload reliability and systems engineering perspectives. Although the image-based approach is less complex in hardware than the interferometer-based systems, the increased computational demand requires significant floating-point performance along with high-speed data transfer and communication rates.

A block diagram of wavefront sensing and control is shown in Fig. 1.3. It is the goal of this research to address the specific issue of the computational demand for image-based wavefront sensing, as outlined as the red block in Fig. 1.3. As a final note, it is the science requirements of the optical system that necessitate high precision optics; and adaptive optics driven by image-based wavefront sensing is an engineering tool that provides high precision results, without the multi-million dollar cost of high precision optics.

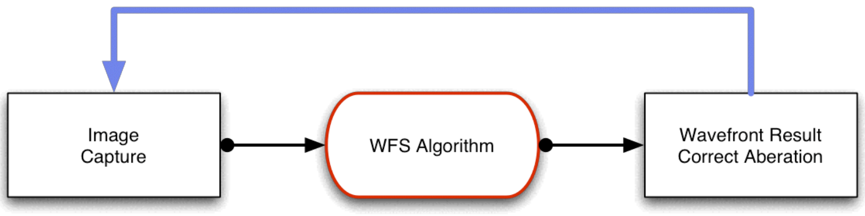


Fig. 1.3: Wavefront sensing and Control Block Diagram.

Phase Retrieval Algorithm

Phase-retrieval is an image-based wavefront sensing method that utilizes point-source images (or other known objects) to recover optical phase information. The fundamental component of phase-retrieval is an iterative transform algorithm developed by Misell, Gercherberg, and Saxton; this will algorithm is referred as the

MGS.[11][12][13] The NASA Goddard implementation of the MGS is thoroughly discussed elsewhere [14] but will be summarized in this section. The MGS utilizes the relationships between amplitude data in both Fourier domains as well as Fourier transform relationship that connects the domains.

The algorithm estimates the phase $\hat{\theta}(x,y)$ in the spatial domain that is associated with the known amplitude data $f(x,y)$ in the spatial domain.

$$f_1(x,y) = \left|f(x,y)\right| \cdot e^{i\cdot\hat{\theta}(x,y)} \qquad \text{(Eq. 1.7)}$$

The algorithm then Fourier transforms to the frequency domain to compute $F_1(u,v)$. Next, the amplitude data in the frequency domain is replaced with the known amplitude data.

$$F_2(u,v) = \left|F(u,v)\right| \cdot \left[\frac{F_1(u,v)}{\left|F_1(u,v)\right|}\right] \qquad \text{(Eq. 1.8)}$$

Thus, $F_2(u,v)$ has the correct modulus and a phase that is derived from $\left|f(x,y)\right|$. The inverse Fourier Transform is then computed.

$$f_3(x,y) = \left|f(x,y)\right| \cdot \left[\frac{f_2(x,y)}{\left|f_2(x,y)\right|}\right] \qquad \text{(Eq. 1.9)}$$

The algorithm is repeated until a convergence criteria is met or a desired number of iterations has been executed, at which point, the phase estimate $\hat{\theta}(x,y)$ is extracted from $f_n(x,y)$.

With certain data configurations and large array sizes, particularly for under-sampled data sets, this core algorithm block can take from tens of minutes to several hours to reach full convergence. This is because two 2-D DFT are required for a single iteration of the algorithm. Furthermore, the algorithm may iterate hundreds of times for multiple images.

It is the goal of this research to document and explore several high performance computing architectures that capitalize on the limitations of implementing this algorithm on a general purpose CPU. As described, the MGS is an iterative algorithm. To speed up the algorithm, there are two techniques that can be utilized. First, the number of iterations can be decreased. Second, the time taken for a single iteration can be decreased. Unfortunately, each iteration of MGS relies on the answer to the previous iteration, thus the algorithm cannot be implemented in parallel.

For the first technique, methods have been developed that increase the convergence rate of the algorithm and thus decrease the number of iterations. Thus, before the algorithm is ported to a distributed high-performance computing architecture, it is critical to implement and consider all possible ways to maximize the performance of the algorithm. One such technique is called phase diversity. The details of why this helps convergence are beyond the scope of this thesis. The diversity, $\varphi(u,v)$, is a known phase term that is added to the H(*u*,*v*), such that (Eq. 1.6) becomes:

$$H(u,v) = A(u,v) \cdot e^{i \cdot (\theta(u,v) + \varphi(u,v))} \qquad \text{(Eq. 1.10)}$$

Another technique is utilizing of multiple images, where each image has a different diversity. Then, each image will produce a phase estimate, and then the individual phases can be combined (usually with a weighted average) into a single estimate for the system. An example of this technique can be seen in Fig. 1.4. For this block diagram, two defocus images are used. The MGS algorithm will produce a wavefront. The inverse of the wavefront is then applied to the optical system to remove the estimated aberrations.

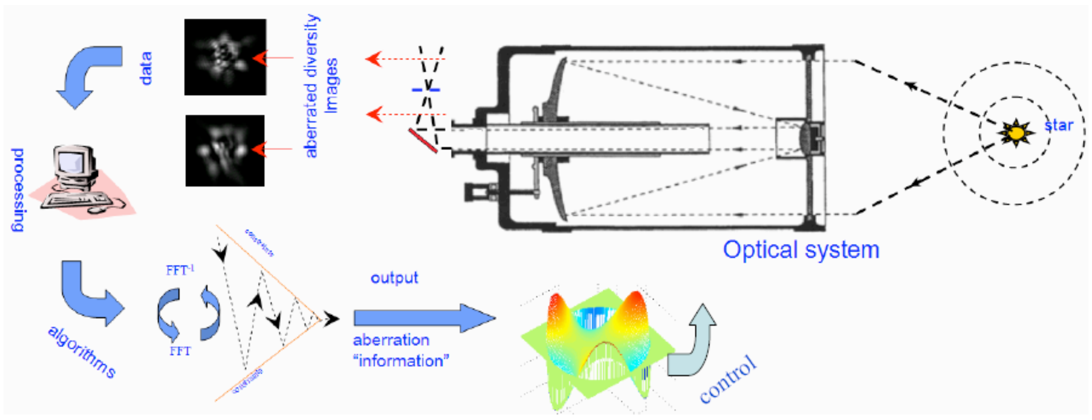


Fig. 1.4: Block Diagram of focus diversity phase retrieval with two defocus images.[15]

The basic MGS algorithm is illustrated in Fig. 1.5. The iterative transform algorithm (ITA) is the core step that is taken in (Eq. 1.7) through (Eq. 1.9). The MGS requires numerous iterations from the image plane to the pupil plane via the Fourier transform relationship. For each "inner loop" of MGS and for each image, a single iteration is completed as constraints in each Fourier domain are applied. Thus, a single iteration results in a Fourier and Inverse Fourier transform pair; hence, the number of 2-D FFTs is the product of the number of diversity-defocus images, the number of outer loops, the number of inner loops, and a factor of 2 for the Fourier and Inverse Fourier transforms of each iteration.

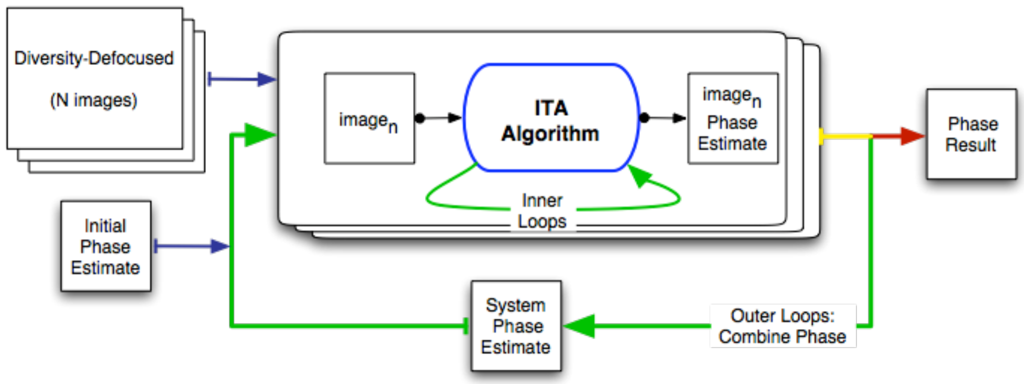


Fig. 1.5: Block Diagram of MGS using Iterative Transform (ITA) phase-retrieval.

An additional technique used for optimization is using the appropriate data size. System specifications will necessitate the number of samples for the phase. For optimal control, this is the Nyquist sampling relationship to the number of degrees of freedom in the deformable mirror or the number of mirror segments. For this application the number of samples in the wavefront should be twice the number of degrees of freedom. This aspect of the data size will be fixed by the science requirements, and thus cannot be changed to increase performance. As a system engineering measuring and testing tool, intuitively, more samples in the pupil are desired. There is an additional optimization with respect to data size. Due to rays of light converging from the pupil to the image plane, the number of samples in both planes are not the same. Given the number of samples in the pupil plane as defined by the science and optical requirements, the image plane must have 2 times that amount in both x and y directions. This is because data is collected in the image plane, and must Nyquist sample the wavefront in the pupil plane. For this reference to the Nyquist sample criteria, the number of samples in the PSF should be twice the number of samples in the wavefront in both x and y. This optical relationship allows a further optimization of the MGS because the data reduction will decrease the computational demands of the FFTs.

Current Solutions

Current desktop and server line general-purpose central processing units (CPUs) have been optimized for performing multiple tasks and use principles of data locality to increase performance, yet performance dramatically decreases when executing large 2-D FFTs.[2][3] For example, a current general-purpose CPU can take several seconds for a double precision 2-D FFT of size 2048×2048. This is mainly due to the fact that the memory architecture is not optimized for such large data sets.

To perform the numerous large 2-D FFTs efficiently, several application-specific architectures have been developed and are discussed further below. Many 1-D and 2-D FFT algorithms exist, yet for each element of the output, the algorithm requires access to every element of the input.[2][16] Thus, as one naive approach to parallelization, an image cannot be divided into sub-components that are processed completely independent. Typically, the 2-D FFT is computed as a series of 1-D FFTs. The total number of 1-D FFTs in the 2-D FFT is then the sum of the number of the image size in both rows and columns. For example with the MGS, 4 diversity-defocus images of size 512×512, with 50 outer loops and 10 inner loops result in more than 4,000,000 1-D FFTs.

The process of performing the MGS on N_{img} diversity-defocused images is highly parallel for each image. This is because the weighted average for combining the phases is significantly less computationally demanding than the 2-D FFTs. Recall that the data size for the phase is one-fourth the size of the image plane data, and as such, the communication requirements for the phase averaging routine are smaller than the communication requirements for the 2-D FFT. Furthermore, a weighted averaging routine is a n_i-to-n_i communication problem among the N_{node} clusters of nodes, where only a given node, n_i, in a cluster needs to communicate with the corresponding node in all other clusters. As such, past approaches to increase performance have used 1 to N_{img} general-purpose CPUs as a cluster.[17] Consequently, this provides a maximum factor of N_{img} improvement, while having the negative effect of increasing power requirements, footprint, and cooling requirements by N_{img}. The primary solution is to provide N_{img} application specific, highly optimized computational cores. All further discussion assumes a single computational core, and thus, the MGS is being performed on a single diversity defocus image. It should be noted that careful consideration must be made to the interface between the multiple computational cores, because of the weighted average of the phase estimates.

The computational cores used on each image can be further divided to perform a distributed 2-D FFT. To perform the distributed 2-D FFT, the computational requirements scale linearly with the number of sub-components up to the size of the diversity-defocus image. Thus, the computational requirements of the MGS on a 512×512 diversity-defocus image size scales linearly until one reaches 512 sub-components of a computational core, or 512 processing units. For implementations of distributed 1-D FFTs, and thus, the scalability for an increase in the number of sub-components larger than the diversity-defocus image size, various algorithms have been explored externally.[18],[19] All discussion in this thesis will assume the 1-D FFT is the smallest component of division.

Digital Signal Processors

For this study, three types of digital signal processors were used: Analog Devices (AD) ADSP-21160M, AD TigerSharc TS-101, and AD TigerSharc TS-201. [20] With these types of DSPs, three systems were constructed of homogenous DSP architectures. The first system consists of 8 of the ADSP-21160M DSPs, the second system consists of 64 TS-101 DSPs, and the third system consisted of 24 TS-201 DSPs. The majority of the research in architectural designs was on the 64 node TS-101 DSPs. Each of these systems are commercial-off-the-shelf (COTS) by Bittware, Inc.

It is emphasized that the science requirements necessitate adaptive optic control, and furthermore, the adaptive optics algorithm and performance requirements necessitate these types of DSPs. Currently available COTS cameras are limited to 18 bits per pixel or fewer. A naive approach is to set the wavefront sensing algorithms to this precision. Due to the iterative and feedback nature of the iterative transform algorithm, errors can quickly propagate. In combination with the desired dynamic range and resolution of the final phase estimate, it can be shown that a minimum of 32-bit floating point precision is required for all processing. An example of wavefront sensing that requires this level of precision is terrestrial planet finding, where the contrast between light directly from a star and reflected from a the planet orbiting around it is 10^{-9}.[21]

All of the DSPs in this study have 32 bit floating point precision, with many other specifications that are characteristic of a modern DSP, including: Harvard Memory Architecture, lower power consumption and a modified instruction set architecture (ISA), such as single instruction circular buffer support, single instruction multiple data (SIMD), and fused multiply adds (FMA). It is the lower power rating that allows multiple DSPs to exist on a single Peripheral Component Interconnect (PCI) card. The block diagrams of a individual 21160M, TS-101, and TS-201 processors can be seen in, Fig. 1.6, Fig. 1.7, and Fig. 1.8 respectively. Items of significant interest are dual computational cores, a link port communication channel, and the direct inter connection between the two.

The characteristics of the ISA such as the FMA and the circular buffer support, allow software developers to create a 1-D FFT with the minimal number of instructions. Furthermore, because of the SRAM scratch-pad cache, instructions can be accessed quickly, with a known timing, i.e. there are no cache misses. Additionally, the circular buffer minimizes the branch miss prediction time. The SIMD and specifically, the FMA instructions are frequently used in the 1-D FFT. Recall the 1-D FFT in (Eq. 1.1). The 1-D FFT can be divided into two steps by striding the data of size N_{fft}, and then performing the 1-D FFT on the even and odd indices of the data, (Eq. 1.11). This division is repeated until N_{fft} is 2, known as Base-2. Notice how this algorithm is a series of multiplications between the α_n and $e^{\frac{-2\cdot\pi\cdot i\cdot n\cdot k}{N_{fft}}}$, with a series of sums. If the $e^{\frac{-2\cdot\pi\cdot i\cdot n\cdot k}{N_{fft}}}$ are precompiled as W_n, then the FMA allows the computations to be performed in one instruction. Cooley and Tukey were the first to discuss this recursive algorithm, known as the FFT.[22] Alternative algorithms that use a Base-4 or Base-8 can be 20%-30% faster than the Base-2; furthermore, factorization of Base-n, for n = 2, 3, 4, 5, 7, 8, 11, 13, 16 is possible with the Winograd transform.[23] All of these approaches can utilize the FMA.

$$\sum_{n=0}^{N_{fft}-1} \alpha_n \cdot e^{\frac{-2\cdot\pi\cdot i\cdot n\cdot k}{N_{fft}}} = \sum_{n=0}^{N_{fft}/2-1} \alpha_{2\cdot n} \cdot e^{\frac{-2\cdot\pi\cdot i\cdot(2\cdot n)\cdot k}{N_{fft}}} + \sum_{n=0}^{N_{fft}/2-1} \alpha_{2\cdot n+1} \cdot e^{\frac{-2\cdot\pi\cdot i\cdot(2\cdot n+1)\cdot k}{N_{fft}}}$$
$$= \sum_{n=0}^{N_{fft}/2-1} \alpha_n^{even} \cdot e^{\frac{-2\cdot\pi\cdot i\cdot n\cdot k}{(N_{fft}/2)}} + e^{\frac{-2\cdot\pi\cdot i\cdot k}{N_{fft}}} \cdot \sum_{n=0}^{N_{fft}/2-1} \alpha_n^{odd} \cdot e^{\frac{-2\cdot\pi\cdot i\cdot n\cdot k}{(N_{fft}/2)}} \qquad \text{(Eq. 1.11)}$$

The link port is a direct communication channel from one DSP to another DSP. It connects the output pins on two chips, and runs at the same clock speed as the processor. Only the physical layer protocol is specified by the DSP, and the board developer and the software developer specify the remaining protocols for the link port. For the TS-101 DSPs, the link ports are single-channel bi-directional. The TS-201 DSPs are dual-channel uni-directional. Both DSPs have 4 link ports per DSP. As will be discussed, these details make a difference in the all-to-all communication. The board vendor, Bittware, Inc., sets these specifications. For completeness, the 21160M DSP has 6 link port channels, but were not used in this study.

The ADSP-21160M is an 80 MHz DSP. The TS-101 and the TS-201 are 250 MHz and 500 MHz DSP. Thus, a single 8-bit link port has a potential bandwidth of 250 Mbytes/sec and 500 Mbytes/sec, respectively. Therefore, the TigerSharcs have a total of 1 GByte/sec and 2 GByte/sec of IO via the link ports, respectively.

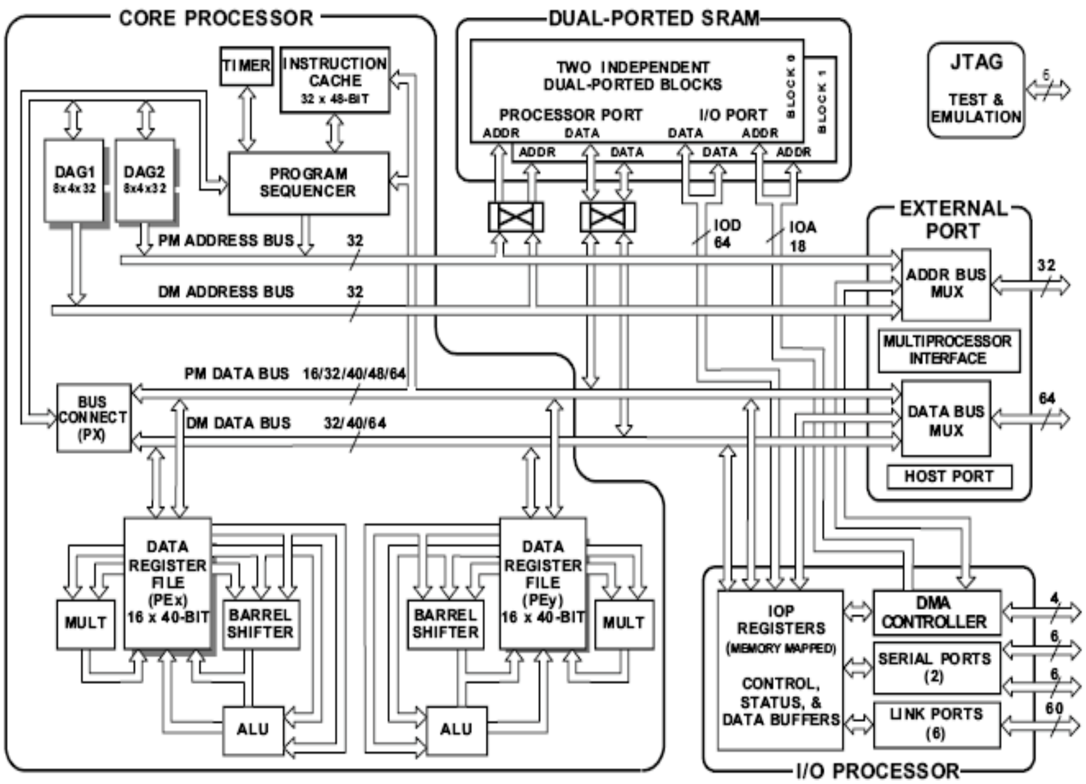


Fig. 1.6: Block Diagram of the Analog Devices Sharc 21160M. [20]

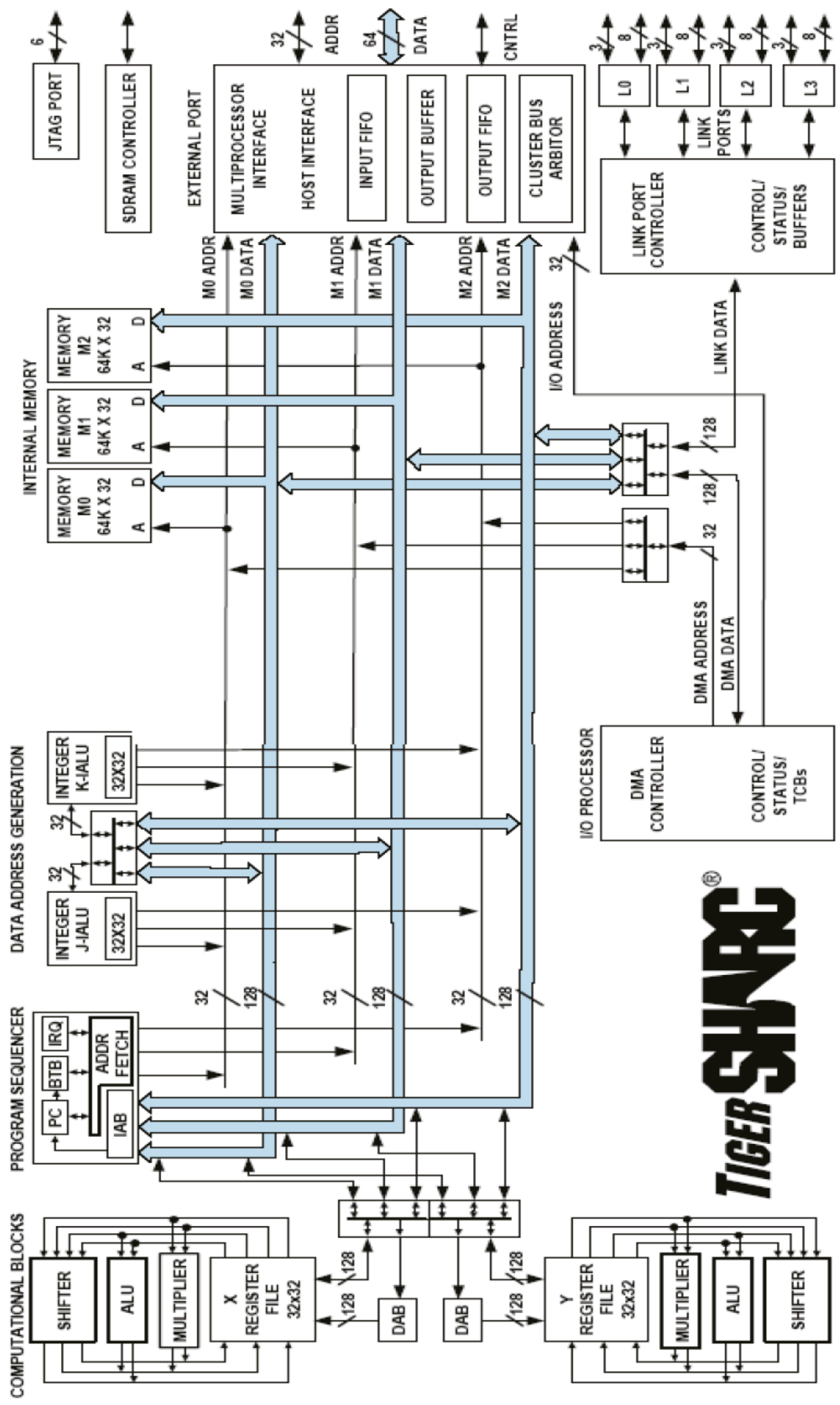


Fig. 1.7: Block Diagram Analog Devices TigerSharc-TS-101.[20]

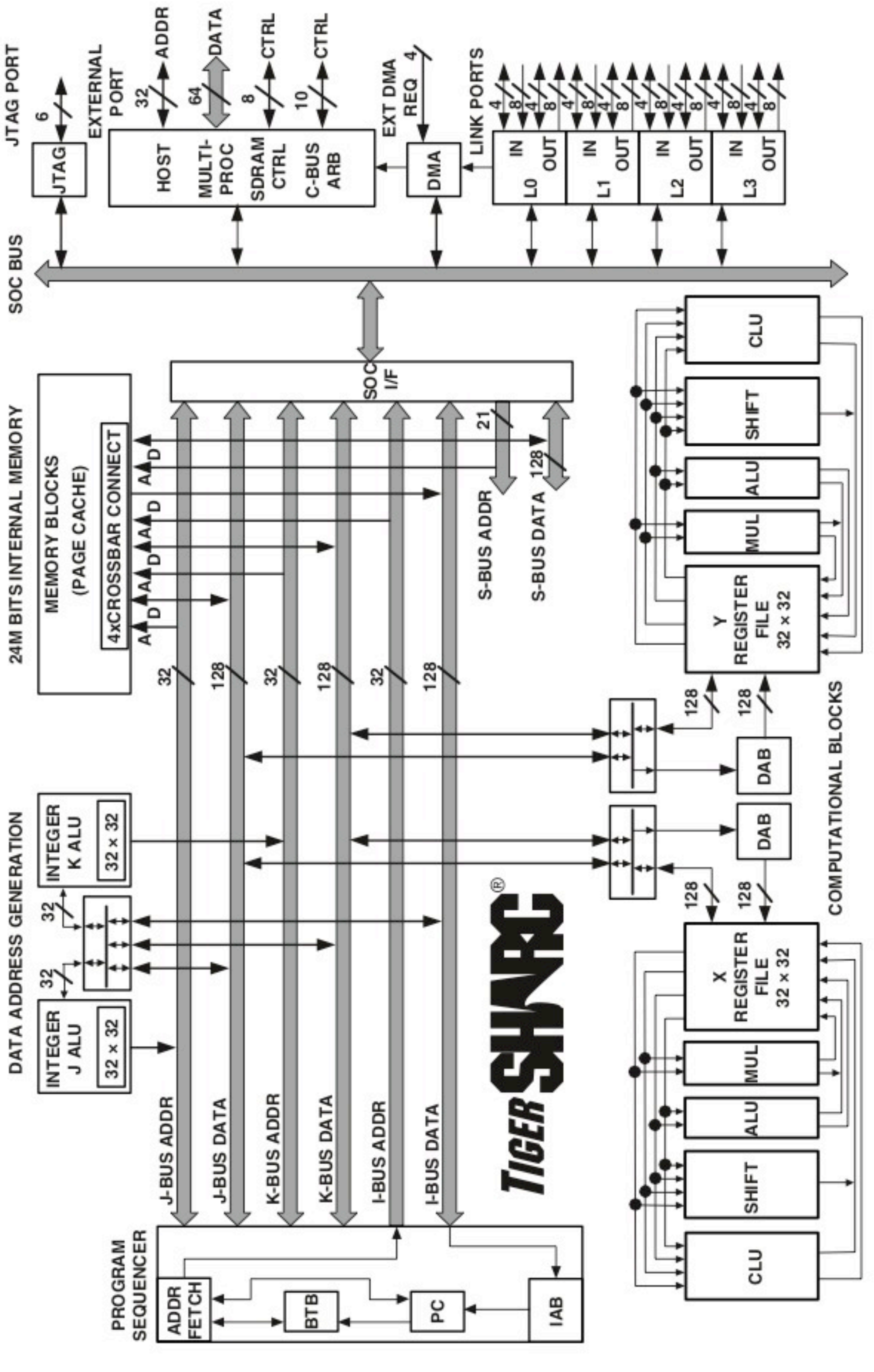


Fig. 1.8: Block Diagram Analog Devices TigerSharc-TS-201. [20]

Systems of Multiple DSPs

The three types of DSPs listed above, the 21160M, TS-101, and the TS-201 are all part of three unique clusters of homogenous DSPs. The 21160M system is a configuration of 8 DSPs. These 8 DSPs are arranged in two clusters of 4 DSPs, as shown in Fig. 1.9. Each cluster is on a single PCI board. For some tests, the two boards were in separate computers, but this is a subtlety that was used to minimize PCI bus contention when multiple images were being processed in parallel.

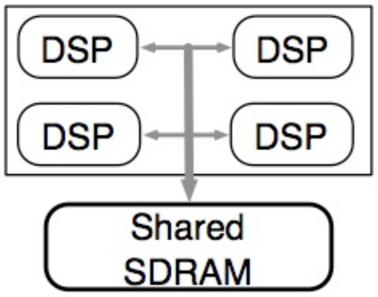


Fig. 1.9: Block Diagram of 4 Node cluster of ADSP-21160M.

The TS-101s are similarly arranged in clusters of 4 DSPs, with the addition of the utilization of the link ports, as seen in Fig. 1.10. The TS-101 system is a cluster of 64 DSPs arranged in two groups. The first group is the cPCI form factor, and the second group is a daughter card with a PMC form factor. The cPCI cards have two clusters of 4 DSP, and 2 slots for a daughter card. The PMC cards have one cluster of 4 DSP. The entire populated board has a total of 16 DSPs, and can be seen in Fig. 1.11. The total system is four of these boards, for a total of 64 DSPs, and can be seen in Fig. 1.13. The four boards are connected via the cPCI bus, and are controlled from a host computer. For software development purposes, the host computer is a general-purpose computer running Windows XP. In practice, after the development cycle is complete, alternatives do exist, such as a real-time operating system that capture images from the camera and feed them directly to the DSPs.

As previously mentioned, the DSP has 4 link ports, and Bittware specifies the layout of the link ports. For the TS-101 system, two of these link ports connect to other DSPs in the cluster, and two are external; the internal link ports are seen in Fig. 1.10 as bold lines. The two external link ports must run at half the clock speed as the processor, which creates a non-homogenous network when connecting multiple clusters. This is because the two internal link ports are on the printed circuit board, which increases the SNR for these connections to the DSPs.

Furthermore, for the TS-101, the two external link ports are routed to different types of ports. One of the ports is routed to the daughter card; this forms a cube, as seen by the red link port in Fig. 1.12. The fourth link port is routed to the additional cPCI pins that are traditionally reserved. The link ports that are routed to the cPCI pins are then the one 'free' link port, which can be used to interconnect any two of the cPCI DSPs in the entire system. The daughter card does not have a direct connection to the cPCI pins, and thus cannot connect to this link port; yet, the daughter card has a similar architecture for connecting to other daughter cards. The two link ports that connect with-in the cluster, the one link port that connects to the cPCI cluster, and a fourth link port that is another 'free' link port, that can connect to another daughter card DSP, with one exception. Only two of the four DSPs (DSP 0 and DSP 1) utilize the fourth link ports. This is because there is insufficient physical room for the additional connection.

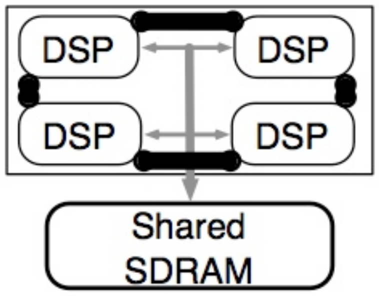


Fig. 1.10: Block Diagram of 4 Node cluster of TS-101.

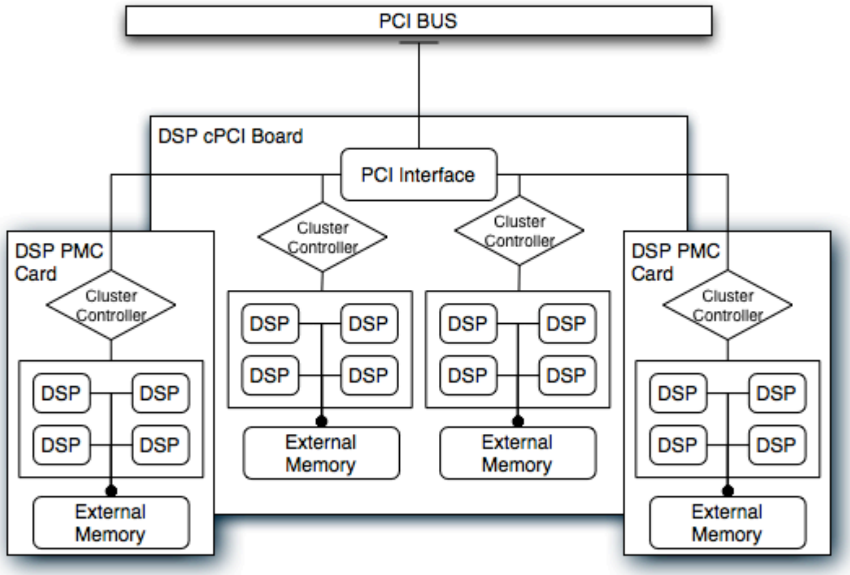


Fig. 1.11: Block Diagram of 16 Node cluster of TS-101.

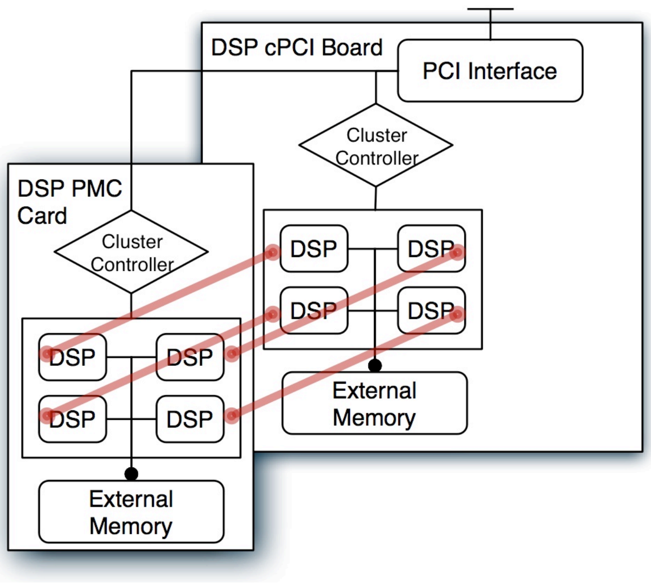


Fig. 1.12: Block Diagram of link port connection between cPCI and PMC daughter card.

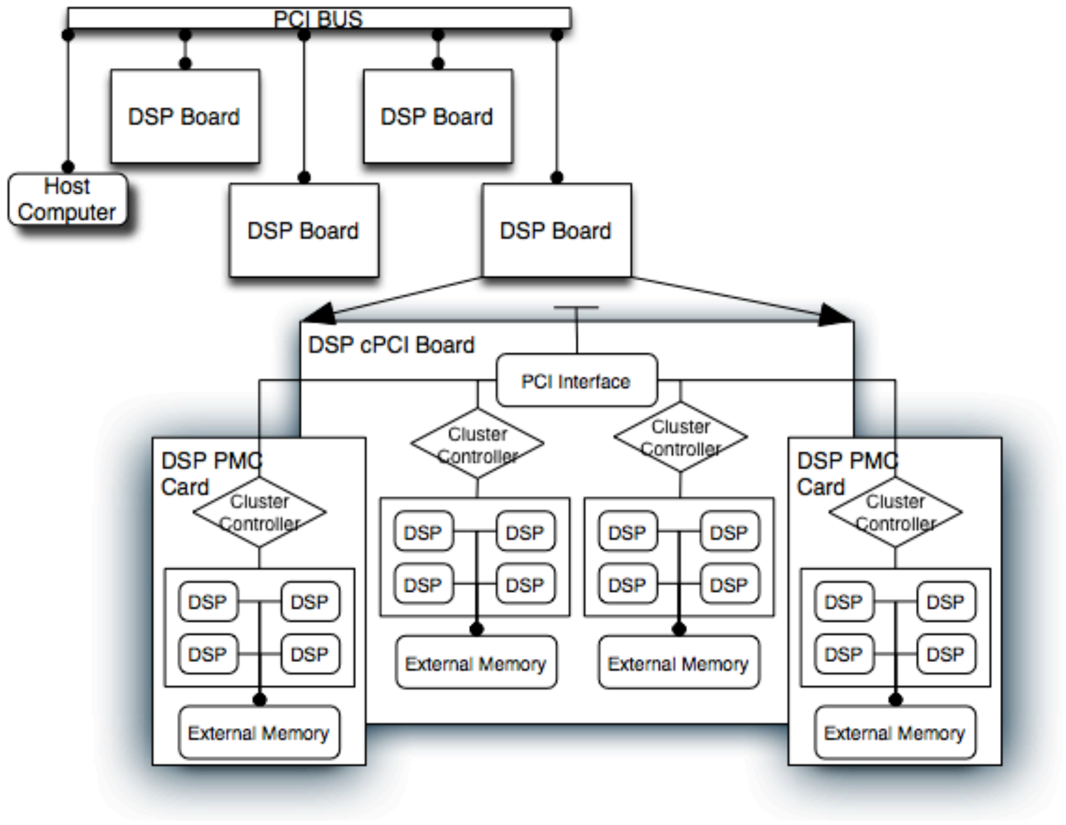


Fig. 1.13: Block Diagram of 64 Node system of TS-101 with host computer.

The TS-201 system is a cluster of 24 DSPs. These DSPs are arranged on six PCI cards of the familiar 4 DSPs per cluster. There is a significant difference with the link ports for the TS-201 compared to the TS-101. All of the TS-201 link ports are routed to an onboard Virtex II Pro FPGA. From there, the link ports can be routed back to the cluster or off the board. If they are routed back to the cluster, the behavior is very similar to the TS-101, with the only difference being the freedom to explore more graph architectures. The link ports that are routed off the board are routed to a standard Infiniband cable, which can be connected to other TS-201 boards. This allows the developer much more freedom in the development of various clusters.

The 21160M system is significantly less powerful computationally than the other systems. The 21160M system is rated at 3.84 Gflops (or .480 Gflops per DSP). On the other hand, the TS101 system is rated at theoretical maximum of 96 Gflops (or 1.5 Gflops per DSP for 64 DSPs) and the TS201 system is rated at 72 Gflops (or 3 Gflops per DSP for 24 DSPs). For funding purposes, a proof of concept was shown on the 21160M system, and after success of the MGS algorithm, the TS-101 system was acquired. Likewise, the TS-201 system was recently acquired based upon the success of the TS-101 system. Most of the research in this thesis is for the TS-101 system.

Graph Theory

A single DSP provides a very fast floating-point precision 1-D FFT. Many of the features of the chip and ISA support the necessary subtleties of the 1-D FFT, and have been optimized as such. For these systems as described above, it is now the focus of this research to maintain that level of optimization for the 2-D FFT. As a simple approach, if we are performing a 2-D FFT on a 64x64 image size,we could use 64 DSPs. This would require each DSP performing 1-D FFT on a single column, then perform the distributed transpose, and finally, each DSP would perform a 1-D FFT on a single row. This is illustrated in Fig. 1.14. The vertical columns represent the local memory of each processor, and the various colors represent the data values of the image. For the 2-D FFT, each processor is performing the 1-D FFT on its own colored column.

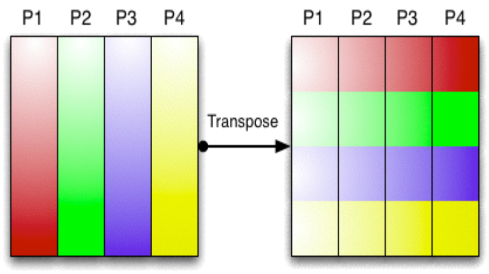


Fig. 1.14: Distributed Transpose on 4 DSPs: Columns represent local memory, and colors represent data.

There are several graph architecture and mathematical concepts that address the issue of an all-to-all communication.

To effectively solve the all-to-all communication problem without sacrificing performance, an intuitive choice for scalability is a graph. A bus architecture would not scale with the number of processors. To achieve true linear scalability in performance for an all-to-all communication, one must have an architecture that provides a direct connection of all processing units to all other processing units, such as the crossbar switch architecture[24] or the complete-graph of type $K_{N_{node}}$ (i.e. N_{node} branches for N_{node} nodes).[25] In practice, this is possible only for small N_{node}, because the number of interconnects per node grows as N_{node}-1, and thus, total number of interconnects I for the complete-graph of type $K_{N_{node}}$ grows as O($N_{node}{}^2$), as seen in (Eq. 1.12).

$$I = \sum_{i=1}^{N_{node}-1} i = \frac{N_{node}(N_{node}-1)}{2} \quad \text{(Eq. 1.12)}$$

Similarly, the crossbar switch architecture grows as O($N_{node}{}^2$) with the total number of switches.

To further explore the possible graphs while avoiding the problems with the complete-graph, some assumptions can be made:

- The graph nodes are homogenous.
- There is the constraint to fix the number of branches per node. For the TS-101 and TS-201, this is 4 branches per DSP.
- The transmit and receive data size is the same for all communication blocks.
- The edge speeds can be treated as homogenous. This is true for the TS-201 system.

The constraint for the number of branches per node is 4, but does not take into account the additional physical board constraints as outlined for the TS-101 system. Each DSP must transmit a block of equal size to every other DSP. This block size is the (Image size per DSP) / (Total number of DSPs). Furthermore, the longest time, or worst-case, to transmit and deliver a single block is the total time for the transpose. Thus, the average-case is of little interest. Furthermore, assume the graph is not distributed homogenously, and some DSPs received all of the data before others.

This would equate to a faster transpose for some DSPs, but due to the iterative nature of the phase retrieval algorithm, the next iteration would correct for this unbalanced timing. Thus, there is no free lunch from the worst-case timing of the communication from DSP_j to DSP_i.

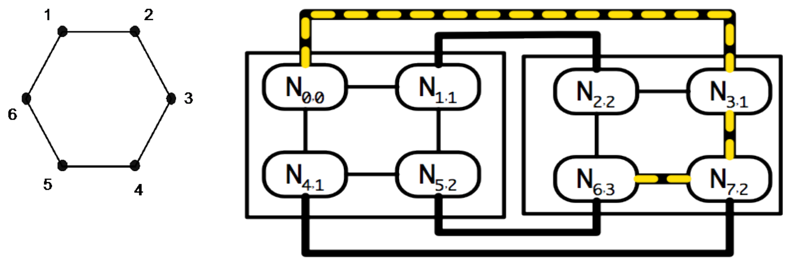
To address this problem, several solutions such as the *n*-dimensional hypercube, ring-torus, and grid architecture are used in practice. These architectures are not the optimal choice under the constraint of number of nodes and the number of branches per node. The optimal homogenous architecture for the all-to-all communication would minimize the graph diameter, (Eq. 1.13), under the constraint of degree of each node. Thus, the optimal graph would restrict the number of branches per node, while minimizing the maximum of all of the “shortest paths”.

Thus, the logical next step is to minimize the worst-case timing of transferring data between any two nodes. Hence, the goal is to minimize the longest *shortest-path* between any two graph vertices; or, to minimize the graph diameter. Two examples of the graph diameter can be seen in Fig. 1.15. For the first example, this is the worst-case of graph diameter of N_{node} nodes, a ring. Yet, for the constraint of two branches per node, it is the only solution for the all-to-all communication.

$$Graph\ \ Diameter \equiv \max_{u,v} d(u,v) \qquad \text{(Eq. 1.13)}$$

The second graph is a bit more interesting; it is an 8-node hypercube of dimension 3. The graph, represented as $N_{id,d(0,id)}$, is of interest because the diameter for this graph is 3, since the path between N_0 and N_6 is 3. The hypercube is inefficient because there are multiple paths to several nodes that are below the graph diameter. For example, there are two paths of distance 2 from N_0 to N_5, both $\{N_0, N_1,$

N_5} and {N_0, N_4, N_5}. Similarly, there are multiple paths to nodes N_2 and N_7. The fundamental flaw is that the number of paths that can be 2 away from N_0 are wasted to nodes that are already 2 away.



(a) (b)

Fig. 1.15: Two graphs demonstrating graph diameter: (a) 6 Node Ring, graph diameter is 3, (b) 8 Node hypercube, graph diameter is 3, distance from N_0 to N_6 is 3, (Node label $N_{id,\, d(0,id)}$).

An example of a solution to this problem is the Peterson Graph, as seen in Fig. 1.16 (b). Furthermore, the Moore's bound for a *p*-node undirected graph provides a methodology for placing a bound on the diameter, (Eq. 1.14), as seen in Fig. 1.17. [26]

$$p \le 1 + d\frac{(d-1)^D - 1}{d-2} \text{ or } D \ge \log_{d-1}\left[\frac{(p-1)(d-2)}{d} + 1\right] \qquad \text{(Eq. 1.14)}$$

The alternative way to view this problem is to consider how many paths exist from a starting node that can maintain the minimum graph diameter, and this can be seen in Fig. 1.16 (a). First, starting with a single node, branch outward adding the fixed number of edges, this will allow one to construct the base. At this point, the distance from the center node to the leaf edges is homogenous and known. This is the optimal-case for the base node. Now, it is desired to add edges such that the overall graph diameter is maintained at a minimum. The added edges to the base case can be seen in Fig. 1.16 (b).

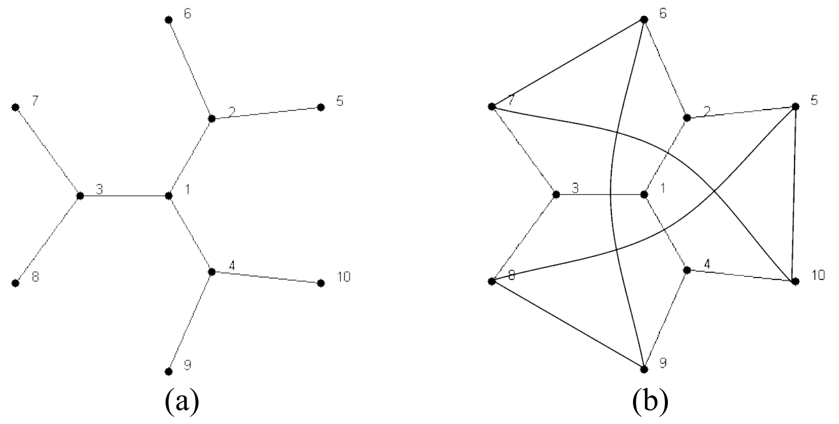


Fig. 1.16: Steps for constructing a graph with minimum diameter: (a) 10 Node base graph, with 3 edges per node and 6 leaves (b) Peterson Graph: graph diameter is 2.

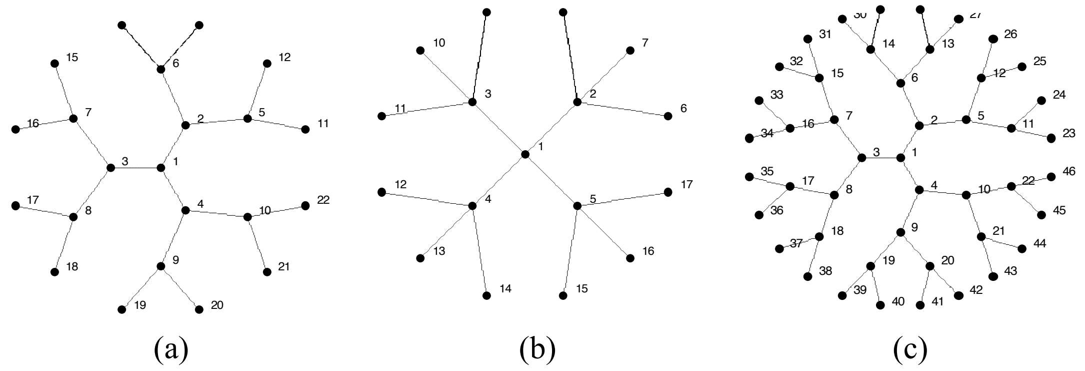


Fig. 1.17: Base architectures for several cases: (a) Base architecture for 22 node, 3 branches per node, optimal diameter of 3 (b) Base architecture for 17 nodes, 4 branches per node, optimal diameter 2 (c) Base architecture of 46 node, 3 branches per node, with optimal diameter of 3.

For the base graphs in Fig. 1.17, the leaf nodes need additional branches added to be a complete *d*-degree graph, and it is proposed that such a graph does not exist.[26] In practice, leaf nodes of the base architectures are removed, and thus, for significantly large graphs, only optimum solutions exist. For example, the Levi Graph in Fig. 1.18, with 30 nodes and diameter 4, is an optimum solution for the 46-node 3-degree diameter 4 graph in Fig. 1.17 (c). [27]

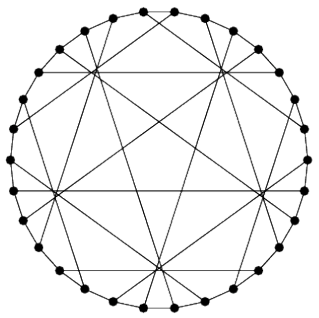


Fig. 1.18: Levi graph, 30 nodes and diameter 4. [27]

Another mathematical concept is the property of the transpose in (Eq. 1.15). For (Eq. 1.15), A, B, C, and D are all square matrices. These can be seen as 4 sub-blocks of a matrix of image. Furthermore, to follow the concept presented in Fig. 1.14, sub-blocks A and B are on one processor, and C and D are on another processor. This property allows the transpose to be performed over sub-blocks, and then to perform the transpose on sub-blocks. This iterative algorithm allows for the further development of architectures, by adaptively supporting the increase in the number of processing units.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^T = \begin{bmatrix} A^T & C^T \\ B^T & D^T \end{bmatrix} \qquad \text{(Eq. 1.15)}$$

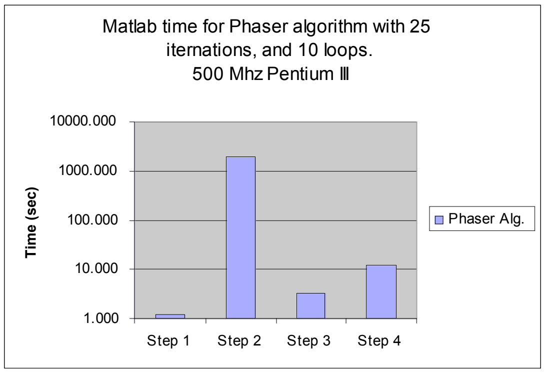
It is this mathematical concept of a recursive algorithm that allows for a balance for the optimal graph of the number of processors. The largest drawback of the optimal architectures such as the Peterson graph is the lack of scalability. To add one node to the Peterson graph requires the entire graph to be redesigned. This is not too difficult for the 10 to 11 node, but for the 32-node to 33-node graph, this is non-trivial. Furthermore, it allows the transpose to be divided into stages that can be solved iteratively with multiple graphs. For example, assume a cluseter of DSPs for the matrix A and B rather than a single DSP. Now, the cluster can be an optimal architecture for computing the A^T and B^T, then, this graph only needs to be an optimal architecture with the corresponding DSPs that are processing the C and D matrices. For example, the cluster for A and B can be a Peterson graph, and the cluster for C and D can be a Peterson graph. Then, the graphs only need to be connected such that the B and C sub-blocks can be transferred. This approach is presented in Chapter 3 on the TS-101 system. This method could then be repeated, for very large N_{node}.

Chapter 2: Phase Retrieval on the ADSP-21160M

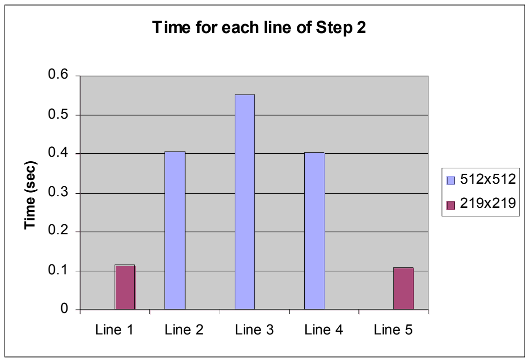
This chapter discusses the first generation of the phase retrieval algorithm on a cluster of DSPs. The hardware for this system was only accessible for 10 weeks. The focus of the research in this stage was to determine the feasibility of the algorithm for a DSP and a distributed system, and to determine bottlenecks for the next hardware generation. At this stage of the research, it was unknown that the transpose of the 2-D FFT would be a bottleneck for scalability of the algorithm.[28]

Algorithm Analysis

This section will outline some of the early decisions about which parts of the algorithm should be implemented on the DSP, and why those decisions where made. Dr. Bruce Dean provided a MATLAB script of a working phase retrieval code. The first step was to identify the bottlenecks of the algorithm, and implement those steps on the DSP. The initial timing for the MATLAB script is given in Fig. 2.1 (a), on a logarithmic scale. The algorithm was divided into 4 steps, where step 2 is the core MGS iterative algorithm outlined on page 6. Steps 1, 3, and 4 are pre-processing and post-processing. It should be noted that even though step 2 is significantly larger than the other steps, that steps 3 and 4 are 3.4 and 11.9 seconds respectively; both of these times are unacceptable for a closed loop adaptive optics control system in a frequently changing environment, as outlined in Chapter 1.



(a)



(b)

Fig. 2.1: Timing for the phase retrieval algorithm in MATLAB: (a) Entire algorithm, (b) details for step 2, with a single iteration.

The first problem to solve is the computational demands of step 2. This specific procedure can be divided into 5 distinct sub-steps:

1. Generate the estmate of the wavefront from a prior value or from the feedback of the previous iteration
2. 2-D FFT
3. Amplitude substitution in focal plane
4. Inverse 2-D FFT
5. Amplitude substitution in pupil plane.

These substeps are performed using the procedures and equations outlined in (Eq. 1.7) through (Eq. 1.9), and their timings are shown in Fig. 2.1 (b). Notice that substeps 1 and 5 are on a smaller data size. For this script and data set, the wavefront size is 219×219, and the PSF image size is 512×512, due to optical constraints of less importance to this research, the PSF image plane data has been up-sampled from 256×256. This data was collected on the Wavefront Control Testbed (WCT) at Goddard.

The PSF data used as input, the resulting estimated wavefront, and the recovered PSFs from the phase estimate are shown in Fig. 2.2, Fig. 2.3, and Fig. 2.4 respectively. In Fig. 2.3, the actuators that command the deformable mirror can be seen as small dimples in the phase.

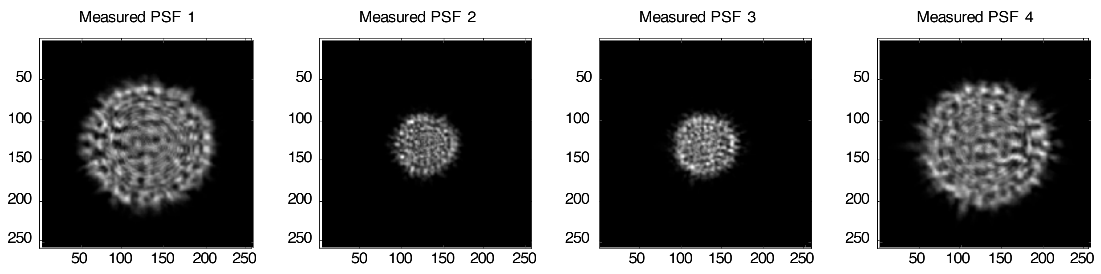
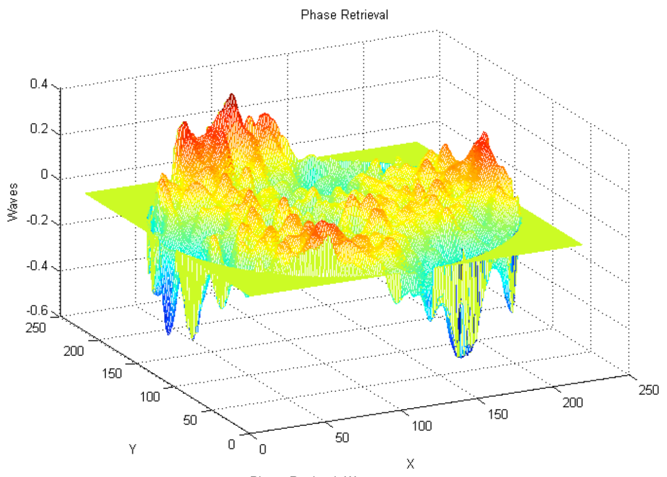
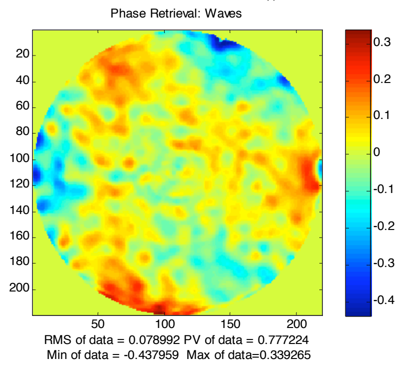


Fig. 2.2: Measured defocus PSF data from the Wavefront Control Testbed (WCT) at Goddard.



(a)



(b)

Fig. 2.3: Phase estimate from the MGS algorithm: (a) Mesh plot, and (b) colored image plot.

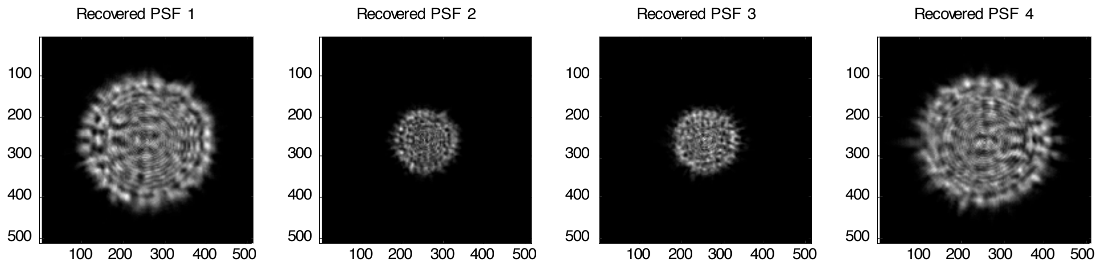


Fig. 2.4: Recovered PSF data from the MGS algorithm.

General Optimizations for CPU and DSPs

The next section outlines four performance driven optimizations that were tested on the general purpose CPU and implemented on the DSP. The techniques are algorithm implementation characteristics, and do not sacrifice computational accuracy; the only sacrifice is an optical engineer's intuition for the understanding of the algorithm. All four optimizations were compared on the general purpose CPU, but were not compared on the DSP. It is a trivial exercise to show that these optimizations are traceable to the DSP. All timings are presented using MATLAB, using a technique that is outlined in Fig. 2.6. It should be noted that MATLAB uses the "Fastest Fourier Transform in the West" (FFTW) algorithm. The FFTW is one of the fastest Fourier transform for a desktop machine, often only being out performed by proprietary chip vendor specific software packages.[2] Furthermore, the API for the FFTW in MATLAB has minimal overhead, and the timing results for a FFT in optimized C with gcc versus the FFT in MATLAB had a negligible difference.

Early on, many steps of the algorithm were identified that were ideal for an optical designer, but were not optimal for a general purpose CPU or a DSP. The first example is the amplitude substitution for substep 3. This was traditionally computed as outlined in (Eq. 2.1). For optical reasons, it provides an intuitive understanding of the underpinnings to represent the algorithm with the complex exponential. From a computer engineering view, this requires the computations of two geometric sums, for the complex exponential and the arctangent. This can be reduced, using Euler's identity and trig properties as shown (Eq. 2.2). This mathematical conversion was able to drastically reduce the computational demand of the third substep, and the new timing result can be seen in Fig. 2.5. Ironically, this is identical to (Eq. 1.9).

$$f_1(x,y) = \left|f(x,y)\right| \cdot e^{i\cdot\tan^{-1}(f_0(x,y))} \quad \text{(Eq. 2.1)}$$

$$\begin{aligned} e^{i\cdot\tan^{-1}(f_0(x,y))} &= \cos(\tan^{-1}(f_0(x,y))) + \\ &\quad + i\cdot\sin(\tan^{-1}(f_0(x,y))) \\ &= \frac{f_0(x,y)}{\left|f_0(x,y)\right|} \end{aligned} \quad \text{(Eq. 2.2)}$$

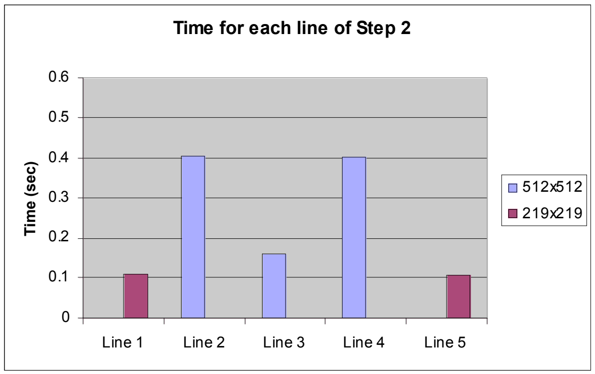


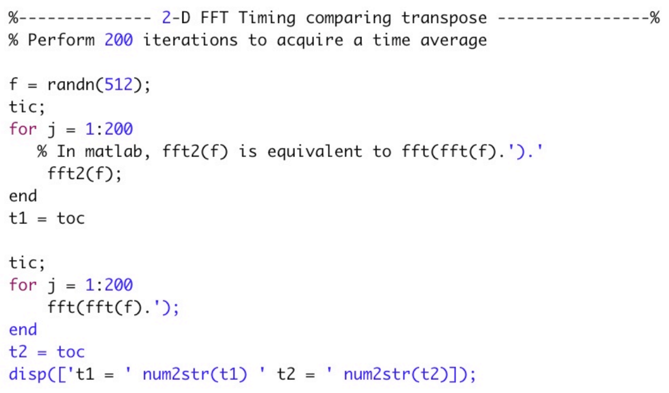
Fig. 2.5: Timing for the phase retrieval algorithm in core iterative algorithm after mathematical modification.

The second optimization of the algorithm discussed is shifting the FFT. A by-product of the 1-D and 2-D FFT is to have the DC component at the edge, $f(\vec{0})$, rather than the center. Images acquired on the CCD have the DC component at the center of the image. Thus, as an intuitive way to understand the algorithm, the computed FFTs shift the DC component to the center before replacing the estimated magnitude data with the measured magnitude data. This approach will require two shifts for the FFT and inverse FFT for each iteration of the algorithm. The solution to this problem is shifting the measured data before the first iteration, and never shift anything again. This technique resulted in an overall time saving in MATLAB of 5.2%.

The third technique in this discussion is the second transpose of the 2-D FFT. In (Eq. 2.3), the $\Im_{rows}\{f[x,y]\}$ represents the 1-D Fourier transform of $f[x,y]$ along the rows. For the 2-D FFT to produce output that is identical to the output of (Eq. 1.3), one must transpose the data twice. This is not necessary if care is given to the algorithm, specifically at the step defined in (Eq. 1.8). Again, the measured data can be pre-processed such that one does not need to transpose the data with every iteration. This can be seen in (Eq. 2.4), where the multiplication is an element-by-element multiplication. To effectively utilize this technique, one must transpose the input data. Then, the estimated PSF is not transposed the second time, and the measured PSF data will then correspond element by element to the calculated PSF. Finally, to ensure the wavefront is oriented correctly with respect to the pupil, diversity, and initial phase estimate, one does not execute the second transpose for the inverse 2-D FFT. This technique reduces the number of transposes in a single iteration from four to two. This will reduce the runtime with the 2-D FFT by 54%. The test script for this case is shown in Fig. 2.6, and similar code was generated for the other cases.

$$F[u,v] = \Im_{rows}\{\Im_{rows}\{f[x,y]\}^T\}^T \qquad \text{(Eq. 2.3)}$$

$$(A \cdot B)^T = A^T \cdot B^T;\ \cdot \equiv \text{element multiplication} \qquad \text{(Eq. 2.4)}$$



```
%-------------- 2-D FFT Timing comparing transpose ----------------%
% Perform 200 iterations to acquire a time average

f = randn(512);
tic;
for j = 1:200
   % In matlab, fft2(f) is equivalent to fft(fft(f).').'
    fft2(f);
end
t1 = toc

tic;
for j = 1:200
    fft(fft(f).');
end
t2 = toc
disp(['t1 = ' num2str(t1) ' t2 = ' num2str(t2)]);
```

Fig. 2.6: Timing code for 2-D FFTW with and without the second transpose.

The fourth optimization makes use of zero-padding that occurs between the phase (pupil plane) the PSF (image plane). Padding is simply adding zeros around an image. Padding in the spatial domain is analogous to up-sampling in the frequency domain. As shown in Fig. 2.7, one can reduce the number of 1-D FFTs by 25% just by not computing the results where the input is zero.

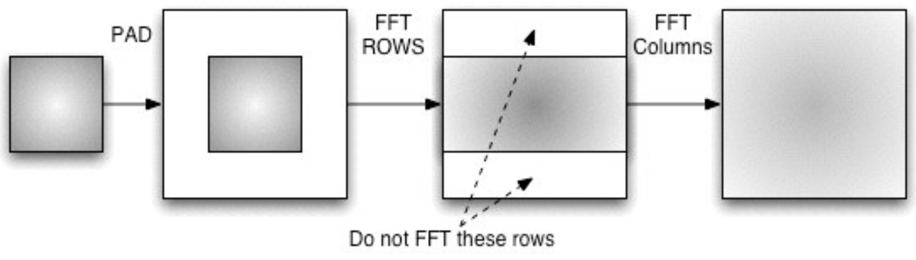


Fig. 2.7: Optimization of 2-D FFT for padding.

Methodology

The MGS algorithm was implemented and tested on the 21160M in three different ways. For this implementation, a single input data file of four diversity-defocus images were used, as seen in Fig. 2.2. The first test method was to have the MGS algorithm run on a single DSP, and utilize four DSPs on a single cluster. The largest bottleneck for this method was the access to the SDRAM for each DSP. Since each image can be run in parallel, the second method utilized four DSPs, but on two boards. Each board used two DSPs, and thus minimized the I/O to the SDRAM.

The third test was the first attempt to parallelize the algorithm at the core, which is beyond the parallelization of multiple images. In this test, two DSPs were used per image, for a total of all 8 DSPs on both boards.

Results

The MGS algorithm in MATLAB is a serial process for multiple images. Thus, the runtime for four images is four times longer than a single image; the runtime for MATLAB is the magenta line in Fig. 2.8, and the timing for MGS on a single image is 1.9 seconds.

The first method as described above is 1 DSP per image on the same board; this is represented as the blue line in Fig. 2.8. The second method as described above is 1 DSP per image on multiple boards; this is represented as the yellow line. The MGS with 1 image per DSP in both cases is the same runtime as MATLAB for 1 image, 1.9 seconds. The difference between the two cases can be seen in the second and fourth images. The 512×512 32-bit floating-point precision images are 1 MB, and thus, too large for the internal memory of a single 21160M DSP. Therefore, for a single DSP to perform the MGS on an image of this size, it must make use of the SDRAM. The cost for this is a slower memory access, which is a shared 64 bit 40 MHz bus. Two images for the first case result in bus contention for the SDRAM, and one can see from Fig. 2.8 that two images on two boards do not have the bus contention. There is a small overhead with downloading a second image to the second DSP board, as is shown in the yellow line.

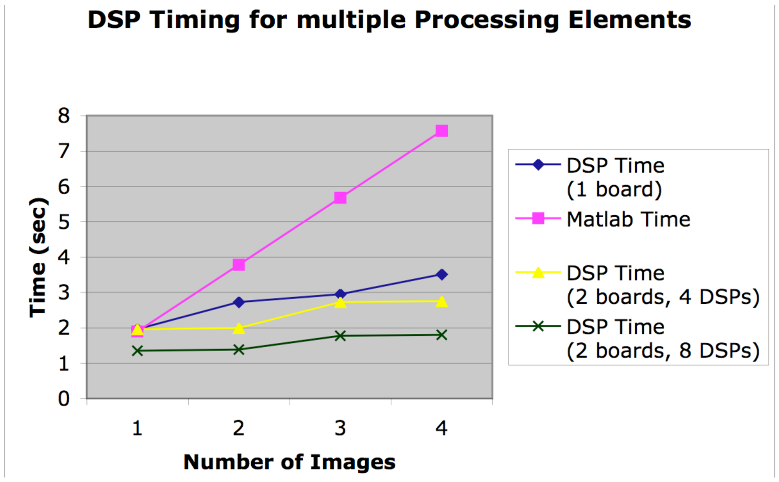


Fig. 2.8: Timing for MGS algorithm in MATLAB and on the 21160M DSP system.

	Input	Output	Time (sec)
FFT	512x1 Real	512x1 Complex	223e-6
FFT	512x1 Complex	512x1 Complex	292e-6
FFT_2D	512x512 Complex	512x512 Complex	606e-3
FFT	256x1 Real	256x1 Complex	105e-6
FFT	256x1 Complex	256x1 Complex	134e-6
FFT_2D	256x256 Complex	256x256 Complex	148e-3

Tbl. 2.1: Timing for 1-D FFT and 2-D FFT for 256 and 512 image size on a single 21160M.

In addition to timing for the MGS on the 21160M, timing for a 1-D and 2-D FFTs alone were explored and are presented in Tbl. 2.1. Analog Device provided the 1-D FFT, and the author used this as a building block to the 2-D FFT. For the 2-D FFT, a single row was moved from the shared external memory to the internal memory before the 1-D FFT was called. The interesting point is the additional time taken for the 2-D FFT in addition to performing the extra computations. The data for the 1-D FFT was in internal memory, yet, the 2-D FFT was too large for the internal memory. To compute the additional time that was spent on I/O and the transpose, one only needs to compute the number of 1-D FFTs in the 2-D FFT and calculate the runtime for these, then, subtract this from the runtime of the 2-D FFT.

For the 512×512 case, the 2-D FFT calls 1024 1-D FFTs, this is 512 for the columns and 512 for the rows. Thus, the computational runtime for the 2-D is 299e-3 seconds and the I/O runtime is 305e-3. These findings indicated that research in the transpose was necessary since the I/O and transpose of the 2-D FFT account for 50% of the runtime on a single DSP. Although desired, further tests that separated I/O versus transpose were not performed due to the short time of availability of the 21160M system. In later systems, Direct Memory Access (DMA) was used to minimize the I/O time, but this can not solve the distributed transpose problem.

Chapter 3: Phase Retrieval on the TigerSharc 101

This chapter focuses on the 64-node TigerSharc TS-101 system. The majority of the research in this thesis was performed on this system. The short but successful research of the MGS algorithm on the 21160M system allowed funding and resources to be allocated for the TS-101 system. Thus, the MGS on the TS-101 system has gone through a thorough and comprehensive study of performance optimization and other algorithm and architecture analysis.[29][30]

Methodology

Several architectures are presented, with results and analysis for most of the architectures, specifically: 1-node, 4-node, multiple types of 16-node, 32-node, and 64-node DSPs per image. As mentioned, the TS-101 DSP has 6 Mbits of internal memory, with 2 Mbits for program and 4 Mbits for data; furthermore, the 4 Mbits is divided into 2 banks of equal size, see Fig. 1.7. A 512×512 complex valued 32-bit floating-point precision image is 2 MB. This translates to 1 Mbit per DSP in a 16-node architecture, which is able to fit into one of the banks of the internal memory of the TS-101. A 512×512 complex valued image is the intermediate result after the second 1-D FFT, or (Eq. 1.8) of the MGS algorithm, and it is the largest image used in the entire algorithm for a 512×512 PSF data set. Thus, the majority of the study is for various types of 16-node architectures. Other image sizes are possible, and results for them are presented below. In every architecture, the transpose happens serially with the processing for the 1-D FFTs. This levels the playing field between architectures. It is left for future research to determine the optimal approach to performing a transpose in parallel with the 1-D FFT computations.

Computational Precision and Accuracy

An important aspect of this research is to ensure that the DSPs produce a phase estimate that is computationally precise (minimal mean of the error) and accurate (minimal standard deviation of the error) to the phase estimate that MATLAB produces. The MGS algorithm was developed in MATLAB, and the algorithm has been rigorously tested in that environment under many conditions with real and simulated data.

Understanding the computational accuracy limitations of the DSP is an important aspect; when one solely relies on the results from the DSPs to make optical decisions and corrections, the limitations must be well understood. If the DSPs produce small errors after a single iteration of the algorithm, then multiple iterations could propagate significant errors; note the feedback of the phase estimate in Fig. 1.5. In the end, computational accuracy is a particular system specification, and one cannot state that X significant figures are needed. Yet, the TS-101 DSP performs all calculations in 32-bit floating-point precision, which is between 7 and 8 significant figures in base-10. If the DSP implementation does not maintain the 7-8 significant figures of accuracy, it should be noted, and careful consideration should be given before trusting results that the DSPs produce.

In addition, any optimization should not sacrifice the accuracy of the result. There are several possible sources for computational accuracy differences between the DSP and MATLAB. As stated, the TS-101 DSP is single-precision floating point and MATLAB uses double-precision by default. Another source of error is the different FFT and mathematical implementations; it is known that different FFT implementations can produce different results.[2] Another mathematical implementation detail is the methodology for trigonometric functions. The complex exponential requires a *cos* and *sin* function, which can be computed a number of ways, such as a Talyor series using "Horner's Rules".[31] One must ensure that these errors do not propagate with the iterative algorithm.

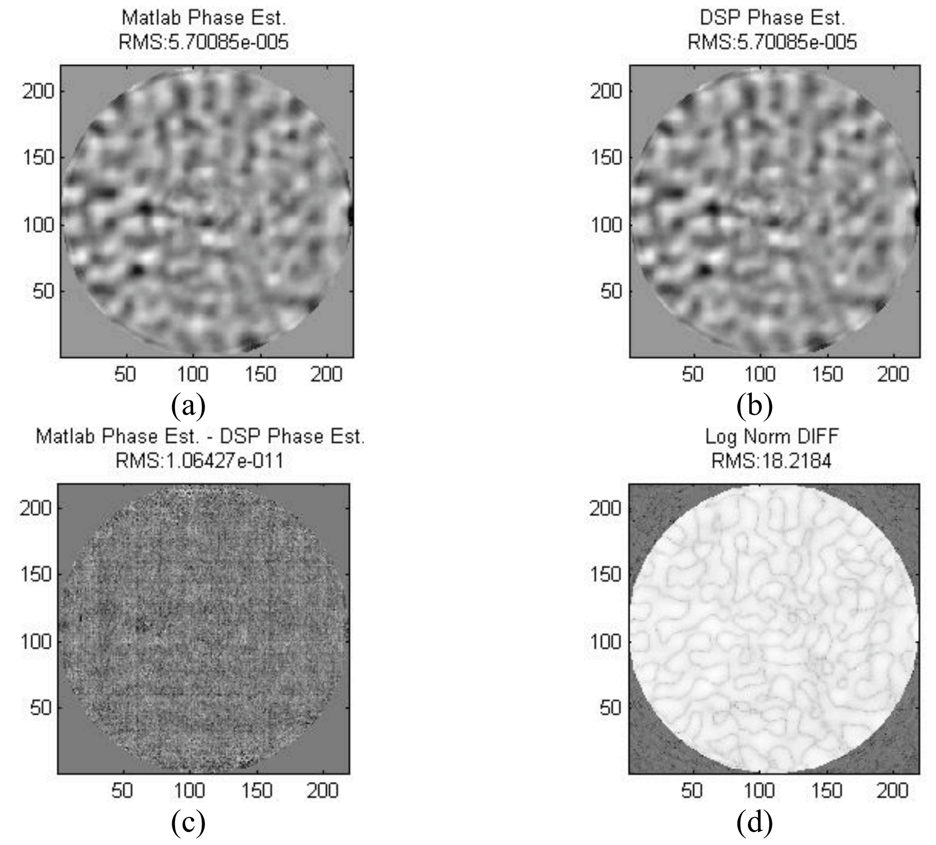


Fig. 3.1: Phase estimate after single iteration, MATLAB (a), DSP (b), the subtraction of the two (c), and the log stretch of the normalized subtraction of the two (d).

Fig. 3.1 shows the methodology that was taken to ensure computational accuracy and precision. With every optimization and technique, a comparison was made to a known phase retrieval estimate for the same input. In Fig. 3.1, the results are shown for a single iteration of the MGS algorithm. In Fig. 3.1, the phase estimate from MATLAB and the phase estimate from the DSPs are shown in (a) and (b) respectively. The subtraction of the two phase estimates and the log stretch of subtracted phases are shown in (c) and (d) respectively. The RMS value is in millimeters, or 57.0 nanometers RMS for the MATLAB phase estimate. The difference between the two phase estimates is 10 femtometers RMS after a single iteration; this is well within specification for any optical system. For the single iteration, the subtraction of the two images is Gaussian white noise.

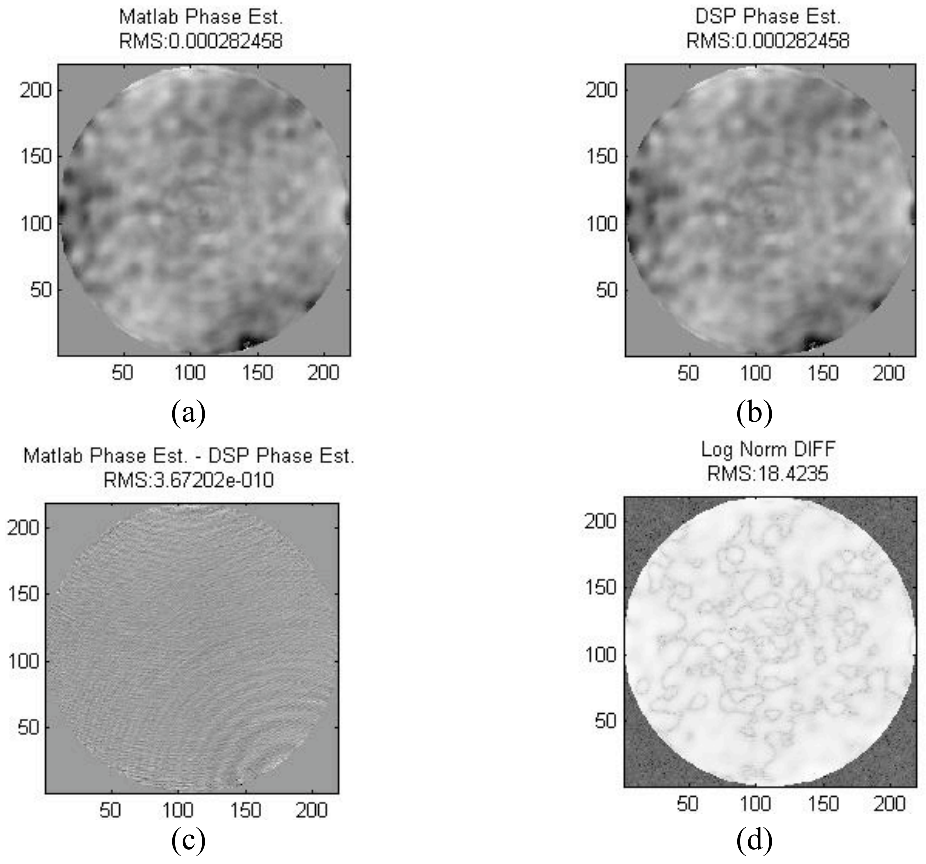


Fig. 3.2: Phase estimate after algorithm convergence, MATLAB (a) the DSP (b), the subtraction of the two (c), and the log stretch of the normalized subtraction of the two (d).

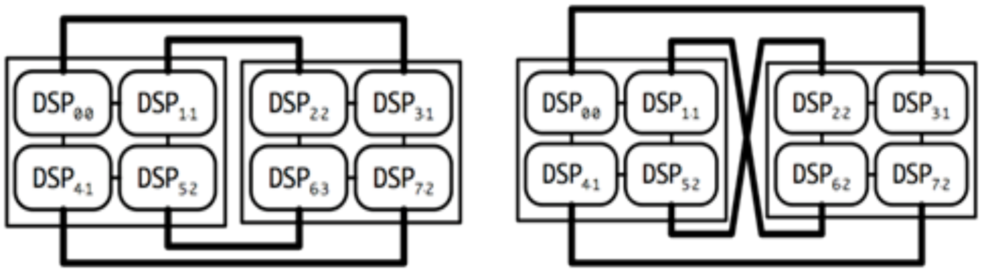
The Fig. 3.2 shows the same information with one exception --- the algorithm has iterated 100 times total to reach convergence; this includes the inner and outer iterations. First note that the difference image has increased from 10 femtometers to 360 femtometers RMS; this is still within the bounds of any optical system. Another interesting difference is the patterns that appear in the difference image. This pattern comes from the single-precision versus double-precision problem, and specifically as the Fourier transform pairs are used to propagate the values based on aliasing high frequency white noise round-off error.

TS-101: 1, 4, and 8-node

As a similar approach to the 21160M system, the first approach was to test the MGS algorithm using a single TS-101 DSP per image. For the 1-D FFT of size 512×1 and 2048×1 takes 23.3 microseconds and 106.9 microseconds respectively. This is ×2.5 and ×3.9 faster, respectively, than a Pentium 4 at 2.4 GHz. Using the similar model as before for the 21160M, the TS-101 is ×7.6 and ×10.2 faster for the 512×512 and 2048×2048 case. This is sufficient in certain applications such as small simulations or stable laboratory environments.

For other applications, specifically closed-loop control, more computational performance is needed, and thus multiple DSPs must be used. The fundamental problem to be solved is scaling the system architecture to minimize bottlenecks. The second architecture considered is the 4-node cluster shown in Fig. 1.10, which utilizes a shared bus to the external memory, SDRAM. The communication requirements of the all-to-all transmission of the 2-D FFT sub-components showed diminishing returns on performance as the number of DSPs were increased. The improvement from 1 DSP to 4 DSPs was only by a factor of ×2.1, with 4 DSPs only increasing performance by 10% over 3 DSPs. To expand beyond 4 DSPs, various grid architectures have been explored.

To utilize an 8-node system, the graph diameter was solely used to characterize optimal graph architectures. Two architectures studied are shown in Fig. 3.3. For the hypercube graph in Fig. 3.3 (a), the graph diameter is 3, as was shown in Fig. 1.15. As demonstrated earlier, the hypercube is not optimal with respect to graph diameter, because multiple paths that are less than the graph diameter are used for the same node pair. If one modifies the hypercube to construct graph (b), a 1-Möbius cube, one can reduce the graph diameter to 2.[32][33] For both of these graphs, the 2-D FFT was explored. For graph (b), the overall runtime of the 2-D FFT on a 512×512 is 10% faster than graph (a). This translates to a reduction in I/O of 21%, since the computation runtimes are the same for both architectures. For this system, the bandwidth of the graph is 125 Mbytes/sec for each link port. The data size for each DSP is 512×64, and the sub-block transmitted in the transpose is 64×64, or 32 KBytes. For each DSP, 7 sub-blocks are transmitted during the transpose.



(a) (b)

Fig. 3.3: Two 8 DSPs grid architectures, hypercube (a) and 1-Möbius cube (b).

For the example of Node-0, with the 1-Möbius cube, the sub-blocks for nodes {1, 3, 4} are delivered in the first iteration of the transpose, and sub-blocks for nodes {2, 5, 6, 7} are delivered in the second iteration of the transpose. This is analogous to the number of hops a node is from Node-0. Yet for the hypercube, the sub-block for node {6} is not delivered until the third iteration. For the total image to be transposed, a total of N_{node}×(N_{node}-1) sub-blocks must be transmitted, where N_{node} is the number of nodes, i.e. the diagonal is not transposed. Since the link ports are single channel bi-directional, the additional iteration in the hypercube requires two steps for both the send and receive among all of the nodes. Assuming in each iteration the maximum number of link ports are utilized, the hypercube requires the transmission of 3, 1, and 1 sub-blocks in each iteration. The 1-Möbius cube only requires the transmission of 3 and 1 sub-blocks in its two iterations. Thus, for both the send and receive, the total number of sub-blocks transmitted are 10 and 8 for the hypercube and the 1-Möbius cube respectively. This equates the transpose to being 20% faster than in the 1-Möbius cube than the hypercube. In practice, this was 21% faster because of the overhead of the third iteration. The 10% improvement, as stated above is realized after including the 1-D FFTs, which is fixed for both graphs.

TS-101: 16-node Architectures

Several 16-node architectures were studied in this research and are presented below. As mentioned, the 16-node architecture is of special interest because of the optimal use of the internal memory that occurs for the 512×512 PSF image size. Furthermore, the standard baseline for processing the MGS was 4 diversity-defocus PSF images, which for 64 DSPs translates to 16 DSPs per image.

The first architecture presented for implementing the MGS algorithm on a 16-node cluster is shown in Fig. 3.4. This architecture utilizes the PCI bus to transfer data between nodes. The PCI bus for the TS-101 system was the cPCI standard

64-bit at 66 MHz, and thus, operated at a speed of 538 MBytes/sec. As the theoretical maximum for the single 512×512 PSF image, this means 4 milliseconds for the transpose. As already stated, the 1-D FFT is 23 microseconds, or for the 512×512 image on 16 DSPs this is 1 millisecond including the optimal padding as previously discussed for the 21160M system. This means, as a theoretical maximum, the transpose would be 75% of the runtime for the 2-D FFT. In actuality, the PCI bus added significant overhead for small packet size, and thus the actual runtime for the 2-D FFT was over 8 milliseconds, and thus the transpose was 88% of the runtime for the 2-D FFT. Before the PCI based transpose was constructed, it was known to be inefficient, but it was developed as a baseline for comparison with other architectures.

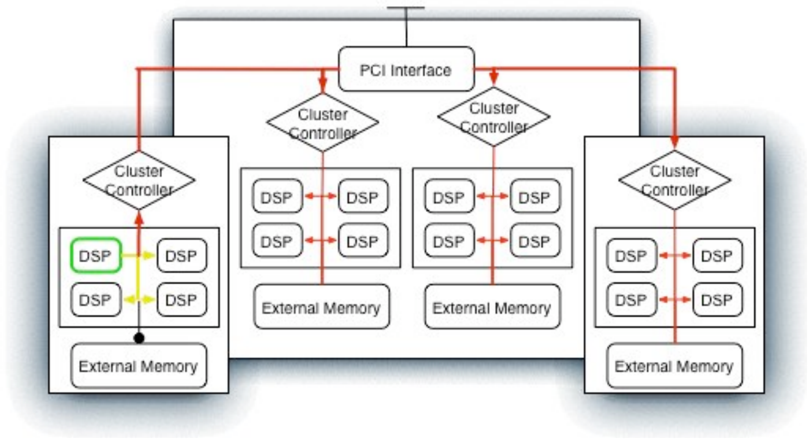


Fig. 3.4: 16 DSP architecture, inefficient use of the PCI bus for all-to-all communication.

Now that the baseline 16-node PCI bus architecture is set, we will explore the graph architecture of the distributed system. The transition from 8 to 16 DSPs per image reduced the sub-component block size for the all-to-all communication by 4, yet, still requires the same total data transfer. For example, a 512×512 image on 8 DSPs results in each DSP performing 1-D FFTs on a 512×64 block. Each block is then divided into 8 sub-blocks of 64×64, where each sub-block is transmitted to the corresponding DSP. The 64×64 sub-block has 4096 elements, but for the 16 DSP case, the sub-block is 32×32 and has 1024 elements.

The first distributed graph architecture presented is shown in Fig. 3.5. The goal of this architecture was to minimize the graph diameter treating all nodes equally. To construct this graph, the black inter-connects are fixed, based on the DSP board, with the design freedom for the red link ports. An algorithm was developed that brute force treated the red link ports as variables, and solved for the graph diameter for each case. The algorithm minimized computational complexity by trimming some repetitive graphs due to symmetry. Ideally, this graph would be similar to the Levi graph in Fig. 1.18 but have a diameter of 3. Due to the constraint of how the clusters are arranged, the graph diameter for Fig. 3.5 is actually worse, having a diameter of 4. Furthermore, this approach did not take into account that the local black link ports operate at 250 Mbytes/sec and the red link ports operate at 125 Mbytes/sec.

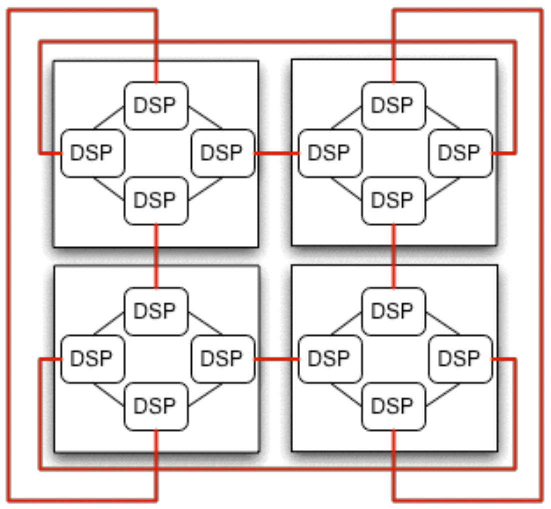


Fig. 3.5: 16 DSP architecture, minimize graph diameter between all 16 DSPs.

The architecture in Fig. 3.5 used a packet technique to route the sub-blocks to the various nodes. The sub-blocks for each DSP were treated the same, and header information was added to each sub-block. Then, each sub-block was routed around the cluster until the transpose was complete. This approach resulted in a transpose that took 55% of the total runtime, or 1.22 milliseconds for the 512×512 case. This is a significant improvement over the PCI baseline.

There are many shortcomings of the architecture in Fig. 3.5, including the necessary packet overhead for routing, the reduced sub-block size, the heterogeneous link port speeds, and the turn-around time for data flow on the physical layer of the link ports. The second architecture, shown in Fig. 3.6, addresses most of these issues and increases the performance of the transpose in practice by 24% This architecture treated the cluster as a single node in the graph, and minimized the graph-diameter between each cluster. Thus, the distance between any two clusters is 1. Furthermore, by increasing the sub-block size transmitted on the red link ports, the edge-routing

DSP reduces the overhead associated with the turn-around time between sending and receiving.

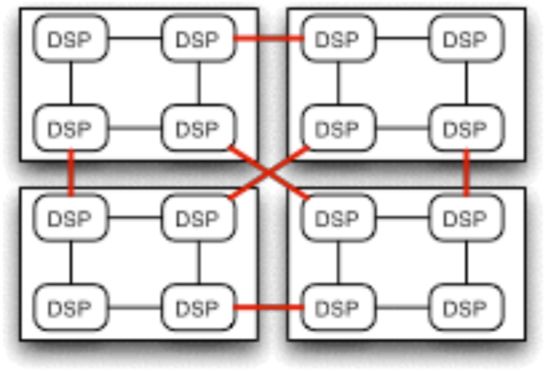


Fig. 3.6: 16 DSP architecture: minimize graph diameter between clusters.

An additional advantage, and method for understanding this performance increase, is to relate the structure of the transpose to (Eq. 1.15). This equation illustrates the recursive nature of the transpose, and thus, why the hierarchy of a low-diameter architecture is efficient. The hierarchy can be visualized if each cluster of four DSPs, rather than the individual DSP, is seen as sub-component computational core. Thus, the transpose must occur within the cluster, and then between the clusters of DSPs. This approach maximizes the bandwidth between the clusters by minimizing the administrative overhead incurred for the first 16 DSP cluster explored in Fig. 3.5.

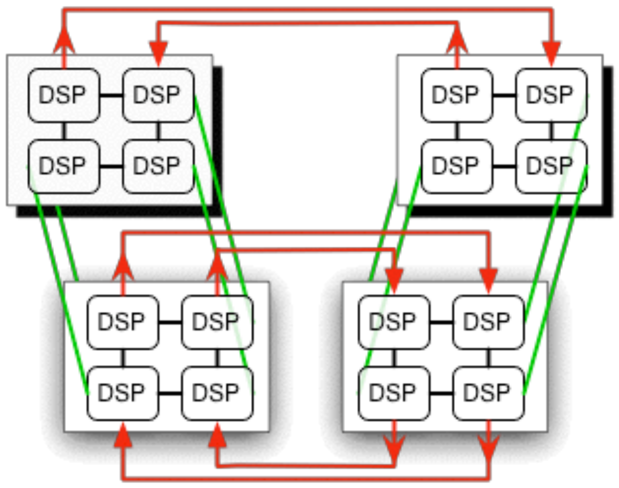


Fig. 3.7: 16 DSP architecture: use PMC cards, and create uni-directional link ports.

The final architecture presented is based on the knowledge acquired in the previous two studies. For the graph shown in Fig. 3.7, the use of PMC cards proves to be both a constraint and a distinct advantage. Although the PMC cards allow all 64 DSPs to be used for the 16 DSPs per image with four images, the PMC cards further restricted implementing the optimal architecture. That is, the PMC cards restrict the red link ports from connecting on the PMC card to the cPCI card. Furthermore, the connection between the cPCI cluster and the PMC cluster is constrained to the hypercube. In practice, the second and third implementations are very similar in performance. This is because the third implementation utilized the fourth link port, and thus increased the overall bandwidth of the graph; the fourth link port is shown in green in Fig. 3.7. Furthermore, to minimize the overhead of the turn-around time on the bi-directional link ports, the link ports were programmed at the application level to be uni-directional between the clusters. Because of the stalls necessary between transmitting and receiving, it is faster to have 1 DSP designated for sending and 1

DSP designated for receiving, and then utilize the faster 250 MBytes/sec interconnect between the two.

TS-101: 16-node Architectures Analysis

In this section, all the results are presented for the Fig. 3.7 architecture.

Timing of the MGS algorithm, : 1 Diversity-Defocus Images, 1 iterations (seconds)		
Diversity Defocus Image Size (NxN)	16 DSP timing	Timing Increase between images
32	0.000179	
64	0.000435	x 2.43
128	0.00117	x 2.68
256	0.00290	x 2.48
512	0.00819	x 2.83
1024	0.0783	x 9.56
2048	0.3265	x 4.17

Tbl. 3.1: Scalability for image size on Fig. 3.6.

The first variable explored is the image size for the MGS algorithm. These timing results are shown in Tbl. 3.1. First, note that the difference between a 512×512 image and a 1024×1024 is four times the amount of data. Thus, the 1024 case should be 4 times longer if run time scales linearly with data size. The computational aspect, the numerous 1-D FFTs, scale linearly with image size, and the distributed transpose scales near linear. Thus, the difference in performance among various image sizes for the TS-101 architecture can be seen in Tbl. 3.1 and averages 2.5 between images of size less than 512×512, where smaller numbers signify an independence between image size and performance time. With each iteration, there is a certain amount of overhead. This comes from setting up the link ports to transmit data, setting up the DMA to read in the data from external memory, or other synchronization routines. As the data size increases, these latencies contribute less to the overall runtime, and the computation and I/O bandwidth make the significant contribution to the overall runtime.

For the larger image sizes, there is a performance hit between the 512 and the 1024 case because the external SDRAM must be used more heavily. As mentioned earlier, the 512 case can fit into the internal cache of 16 DSPs. Note that as expected, the 2048 case is 4 times longer than the 1024 case.

The next case is the number of images processed on the 16 DSPs per image architecture. As mentioned earlier, combining the phase estimates after an iteration on a single image is negligible to the overall runtime. The difference in runtime for four images versus one image is only 2%; thus, the four images on 64 DSPs with 16 DSPs per image is 3.95 times faster, than four images on 16 DSPs with 16 DSPs per image. This is represented graphically in Fig. 3.8 and Fig. 3.9.

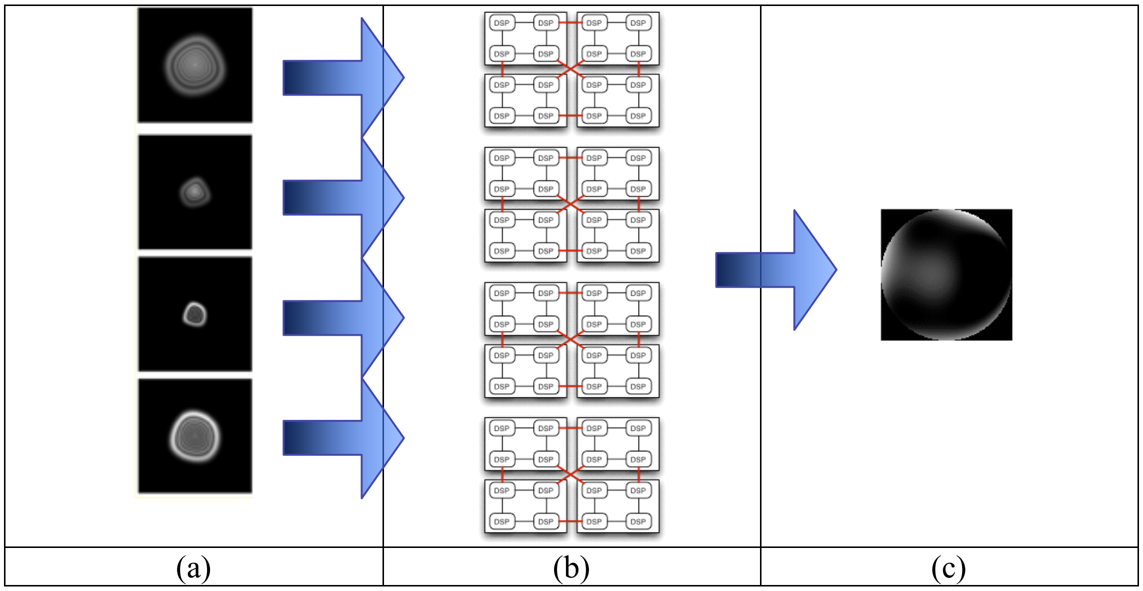


Fig. 3.8: Four images (a) on 64 DSPs (b) with 16 DSPs per image to produce a single phase estimate (c).

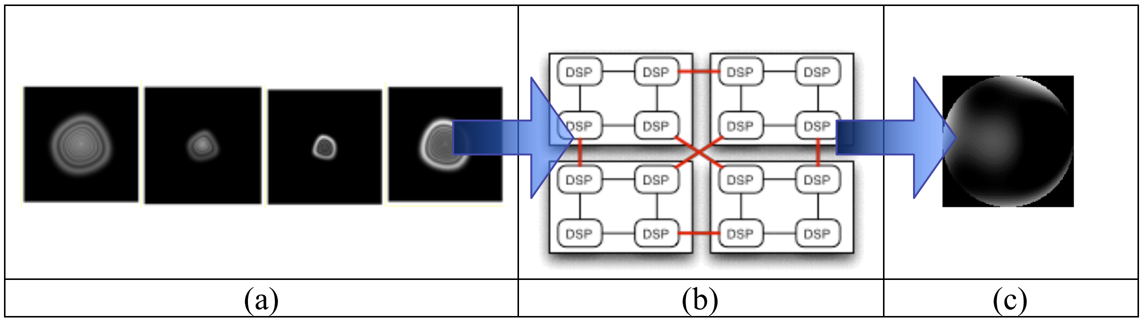


Fig. 3.9: Four images (a) on 16 DSPs (b) with 16 DSPs per image to produce a single phase estimate (c).

TS-101: 32-node and 64-node Architecture

For the 16 to 32 DSPs per image, only the sub-clustered routing architecture was explored, shown in Fig. 3.10. For computations, i.e. 1-D FFT and other mathematical routines, the improvement between 16 and 32 DSPs is nearly linear. Thus, the computational routines provided ×2 improvement. In addition to the scalability of the computational routines, one must explore the scalability of the data transfer. For a bus architecture, increasing the number of nodes will result in a decrease in the bandwidth per node. For the TS-101 systems, increasing the number of nodes also increases the total amount of network bandwidth. For example, a single TS-101 has 4 link ports at 250 MB/sec, and thus for the 16 node cluster, this network has a theoretical maximum bandwidth of 8 GB/sec (16 nodes × 250 MB/sec × 4 link ports / 2 for interconnection = 8 GB/sec). Similarly, the 32 TS101 system has an over network bandwidth of 16 GB/sec. The 32 DSP architecture presented in Fig. 3.10, was ×1.2 faster than the transpose on the 16 DSPs architecture, Fig. 3.6 and Fig. 3.7, for an overall improvement for the 2-D FFT of 1.7 for the 512×512.

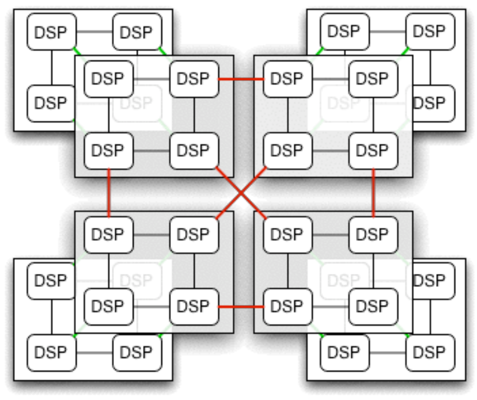


Fig. 3.10: 32 DSP architecture.

Several timings for various image sizes are presented in Tbl. 3.2.

Image size	Pentium 4 2.4 GHz	TS-101 16 DSP per image	TS-101 32 DSP per image
64×64	20.5	.0561	.0322
512×512	219.	.491	.288
2048×2048	8890	19.6	12.2

Tbl. 3.2: Timing for various image sizes on three architectures, with 4 Diversity-Defocus images after convergence (seconds).

Although for most applications of the MGS, multiple diversity-defocus images are used, some applications only make use of a single image. For the single-image case, an architecture for 64 DSPs per image was constructed, Fig. 3.11. This builds upon the proven methodology from the 16 and 32 DSPs per image. This architecture has been designed but not implemented, and results of this graph will be presented in future research. The routing algorithm would treat each cluster of 16 as a cluster in a similar method used in Fig. 3.6 and Fig. 3.10.

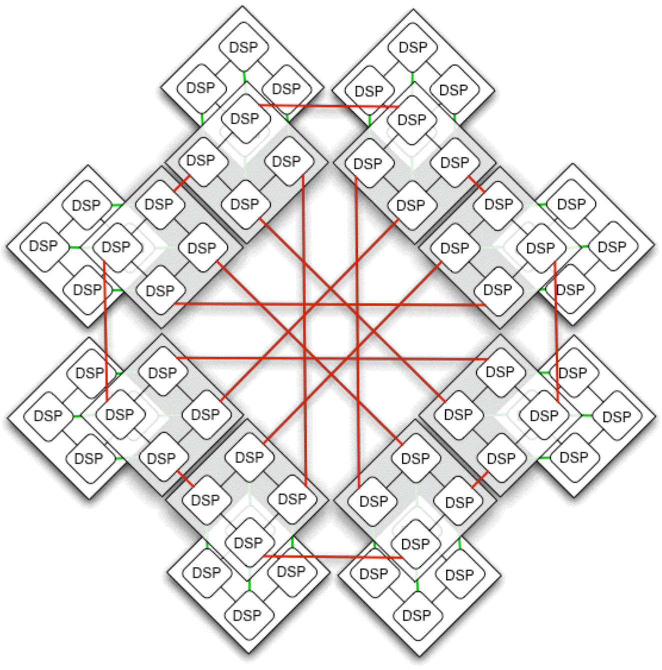


Fig. 3.11: 64 DSP architecture.

Chapter 4: Phase Retrieval on the TigerSharc 201

The final system presented in this thesis is the 24-node TS-201 system. This system was recently procured, and there has not been time to fully explore the design space of this system. As described in the Forward, this system has already been deployed in the Testbed Telescope (TBT), which is a 1/6th scale model of NASA's James Webb Space Telescope (JWST). The TBT is a segmented primary telescope as seen in, Ill. 1.1 (a). The TS-201 system is providing real-time processing on 100 images.

Hybrid Diversity Algorithm

The goal for this research was to prepare the TS-201 system for the TBT. Additionally, a more complicated phase-retrieval algorithm, adding a process called adaptation to the core MGS algorithm and now called the Hybrid Diversity Algorithm (HDA), was implemented with the DSP architecture. The HDA is outlined in Fig. 4.1.

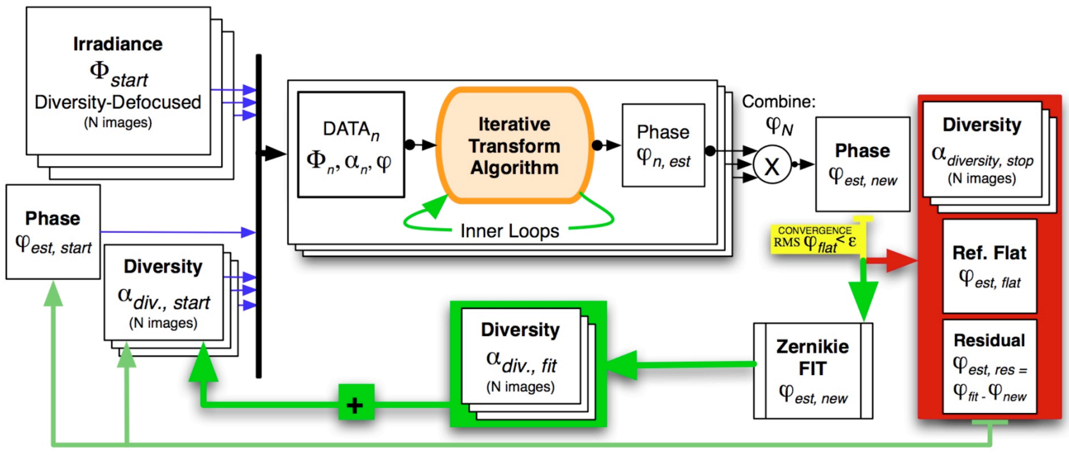


Fig. 4.1: Block diagram for the HDA.

The HDA is similar to the MGS, with the iterative transform algorithm at the core, shown in the orange block. Thus, like the MGS, the HDA must have an efficient 2-D FFT. The third feedback control loop of the HAD, shown as the faint green line, is of particular interest to performance of the algorithm implementation. This feedback is iterated between 5 and 20 times, and thus, the HDA is 5 to 20 times more computationally demanding. More details about the HDA are provided elsewhere.[14]

TS-201: 4-node Architecture

The goal of the TS-201 system was to process 100 diversity defocus images in less than 4 minutes using the new HDA algorithm on 512×512 images. The TS-201 DSP has 24 Mbits of internal memory, and thus, a 512×512 image is able to fit into the internal cache of 4 TS-201 DSPs, allowing 6 images to be processed in parallel on the 24 node cluster. If fewer than 4 DSPs per images where used, then the image data, and any temporary values would have to be stored in external memory. This would cause the same problems as were seen with the 21160M and TS-101 system, where the 4th DSP in a cluster only increased performance by only 10% when only the cluster bus alone is used for the I/O.

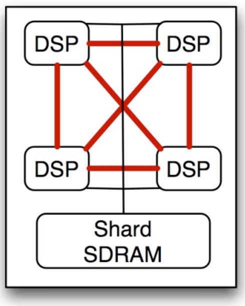


Fig. 4.2: TS-201 4 DSP architecture.

The TS-201 system used a complete-graph of type K_4. This can be seen in Fig. 4.2, where the red lines are the link ports. As mentioned, each link port is dual channel uni-directional, and thus, there is minimal overhead for this architecture when compared to a similar architecture from the TS-101 system.

The HDA algorithm required adaptation, which is a fitting routine for the phase estimate. The fit is performed to a set of basis functions that are pre-computed before the first algorithm starts. In Fig. 4.1, it is labeled as a Zernikie Fit. The number of basis functions is a system specification, and for the TBT, 285 basis functions where used. This is 15 global basis functions and 15 basis functions per segment. These two sets of basis functions where orthogonal to improve the fitting routine and reduce computational demand (cross-correlation matrix is a diagonal matrix). This fitting required 15% of the runtime of the algorithm for a single iteration.

The entire runtime for the algorithm for a single iteration on a single 512×512 image was 0.00859 seconds. This is analogous to the data in Tbl. 3.2 for the 16 DSP TS-101 system; recall that the single iteration for the 512 case is 0.00819 seconds. After removing the 15% for the adaptation, the 4 TS-201 DSP system is faster than the 16 TS-101 DSP system. This is for three reasons: (1) the TS-201 DSP is 2× clock speed and 2× the link port speed of the TS-101 DSP, (2) the complete graph K_4 allows the transpose to occur in a single iteration, where every DSP is on hop away, and (3) the larger data size per DSP reduced the latency and overhead as was seen for the TS-101 system in Tbl. 3.2.

Chapter 5: Conclusions

In summary, the HDA and MGS algorithms require the evaluation of numerous 2-D FFTs. The transpose of the 2-D FFT is the bottleneck for scalability on a distributed system. It has been shown that a distributed computational grid architecture, processing on each image, is the optimal architecture in terms of performance, with further details specified by various requirements depending on the desired footprint, power consumption, and operating environment. These details imply the specification of the sub-components.

Summary

In this research, three unique homogenous DSP systems were studied: a 16-node system of the AD 21160M, a 64-node system of the AD TS-101, and 24-node system of the AD TS-201. Several architectures for each system were explored, presenting optimal solutions to meet various optical system requirements.

Assuming a perfectly homogenous network, with entire design freedom for the interconnection, to minimize graph-diameter results in the optimal architecture for the transpose, this was shown in Fig. 3.3 (b) and Fig. 4.2. In practice, for the TS-101 system specifically, different edges have different transfer speeds and some edges are not adjustable. Thus, the optimal architecture is a hybrid graph, where clusters of nodes are treated as a single node, as seen in Fig. 3.6, Fig. 3.10, and Fig. 3.11. Then, the clusters of nodes are constructed in the optimal low graph diameter architectures.

Data size for the various architectures are optimal for the largest data size that can completely fit into the cumulative internal cache of the distributed system, which is the summed internal cache of every DSP in the architecture. The 512×512 image size fits into the architecture of 16 TS-101 DSPs, and thus is the optimal number of DSP in terms of performance. The TS-201 DSPs have a larger internal cache, and the optimal size is 4 DSPs.

Future Work

Future work will entail other wavefront sensing algorithms that require numerous 2-D FFTs. Algorithms based on the core MGS procedure constitute one class of wavefront sensing, but other classes of algorithms exist. Current work is underway to develop a phase diversity algorithm, which uses 2-D FFTs to connect data n conjugate Fourier domains, but otherwise is very different than the MGS.

The development and testing of the 64 DSP architecture in Fig. 3.11 is of interest and may be the topic of future studies. To date, it has only been of theoretical interest to study the 64 DSPs architecture, because most closed loop control systems will use multiple images in the MGS. Additionally, as a motivation, the 1024×1024 image would fit into the internal cache for this architecture.

All of this research performs the transpose in serial with the computation of the 1-D FFT. To perform these two steps in parallel is a non-trivial task, but possible. The DSP could compute the 1-D FFT on K rows, a subset of all the rows this DSP will process. Then, the DSP could transpose these K rows, while it processing the next K rows. Currently, the DSP uses DMA to move data from external memory to internal memory and to move data on the link ports. The DMA on both the link ports and shared external memory, and the computation of the 1-D FFT can be run in parallel, and a future study that determines the performance increases for such an architectures is of interest.

In addition to space optics wavefront sensing and control, additional application areas that can benefit from the application of digital signal processors include ground-based wavefront sensing, telescope image processing, laboratory optical processing, system design and tolerancing, Monte-Carlo simulations, and finite element modeling. To date, there are no radiation-hard implementations of high performance DSPs as required for autonomous space optics control. However, the FPGA (field programmable gate array) or ASIC (applications specific integrated circuit) technologies can lead to a high-performance solution in a radiation-saturated environment. Currently, NASA Goddard Space Flight Center is exploring several possibilities, including reconfigurable computing, to develop systems of FPGAs to meet the requirements of high-speed space-based image-processing as well as wavefront sensing and control. [34][35]

Acronyms

AD	Analog Devices
cPCI	Compact Peripheral Component Interconnect
DMA	Direct Memory Access
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
ITA	Iterative Transform Algorithm
JWST	James Webb Space Telescope
MGS	Misell-Gerchberg-Saxton
PCI	Peripheral Component Interconnect
SNR	Signal to Noise Ratio
TBT	Testbed Telescope
WCT	Wavefront Control Testbed
WFS	Wavefront Sensing
WFS&C	Wavefront Sensing and Control

References and Bibliography

[1] J. C. Hankins, "Technology Readiness Levels: A White Paper," NASA, Office of Space Access and Technology, Advanced Concepts Office, April 6 (1995).

[2] M. Frigo and S. G. Johnson. "FFTW: An Adaptive Software Architecture for the FFT," http://www.fftw.org.

[3] J. L Hennessy and D.A. Patterson (2003) Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publisher, San Francisco, CA.

[4] Photograph courtesy of NASA, http://www.jwst.nasa.gov/.

[5] Photograph courtesy of W.M. Keck Observatory, http://www.keckobservatory.org/.

[6] R. A. Gonsalves and P. Considine, "Phase-Retrieval from Modulus Data," J. Opt. Soc. Am., 66, 961-964 (1976).

[7] R. A. Gonsolves, "Phase Retrieval and Diversity in Adaptive Optics," Optical Engineering, Vol. 21 No. 5 page 829 (Sept./Oct. 1982).

[8] R. Q. Fugate, B. L. Ellerbroek, C. H. Higgins, M. P. Jelonek, W J. Lange, A. C. Slavin, W J. Wild, D. M. Winker,' and J. M. Wynia, "Two generations of laser-guide-star adaptive-optics experiments at the Starfire Optical Range" 0740-3232/94/010310-15$06.00, Optical Society of America (1994).

[9] F. J. Rigaut, B. L. and Ellerbroek, R. Flicker, "Principles, Limitations and Performance of Multi-Conjugate Adaptive Optics", http://citeseer.ist.psu.edu/305244.html.

[10] J. Wenhan, and L. Huagui, "Hartmann-Shack wavefront sensing and wavefront control algorithm", SPIE Adaptive Optics and Optical Structures. A91-29476 16-74, (Mar. 1990).

[11] R.W. Gerchberg and W.O. Saxton, "Phase Determination from Image and Diffraction Plane Pictures in an Electron Microscope", OPTIK, 34, 275 (1971).

[12] R.W. Gerchberg and W.O. Saxton, "A Practical Algorithm for the Determination of Phase from Image and Diffraction Plane Pictures", OPTIK, 35, 237-246 (1972).

[13] W.O. Saxton, “Computer Techniques for Image Processing in Electron Microscopy”, in Advances in Electronics and Electron Physics, Supplement 10, L Marton and C. Marton, eds. (Academic Press, New York, NY, 1978).

[14] B. Dean, D. Aronstein, S. Smith, R. Shiri, and S. Acton, “Phase-Retrieval Algorithm for JWST Flight and Testbed Telescope”, Proc. of SPIE Vol 6265 626511, 2006.

[15] Bruce Dean, “Introduction to Image-Based WFS&C”, NASA Goddard Space Flight Center, Optics Branch Technical Noon Talk (10/20/2005).

[16] R.C. Gonzalez & R.E. Woods Digital Image Processing. Prentice Hall, Upper Saddle River, N.J. (2002).

[17] W. Hayden, L. Boyce, K. Rehm, D. Redding, and C. Ohara, “Analysis of Wavefront Sensing and Control Onboard Resource Requirements – 1 & 2”, Computer Science Corporation, Technical Memorandum, Apr-2002.

[18] R.C. Agarwal, F.G. Gustavson, and M. Zubair, “A high performance parallel algorithm for 1-D FFT”, IEEE Supercomputing '94. Proceedings, pages: 34-40, Nov 1994.

[19] D. Takahashi and Y. Kanada, “High-Performance Radix-2, 3 and 5 Parallel 1-D Complex FFT Algorithms for Distributed-Memory Parallel Computers”, Springer Netherlands, pages 207-208, February 2000.

[20] Photograph(s) and specification courtesy of Analog Devices, http://www.analog.com/.

[21] P. J. Borde and W. A. Traub, “High-contrast Imaging for Space: Speckle Nulling in a Low Aberration Regime”, The Astrophysical Journal, Volume 638, Issue 1, Feb-2006.

[22] J. W. Cooley and O. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series." *Math. Comput.* 19, 297-301, 1965.

[23] W. H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Fast Fourier Transform." Ch. 12 in *Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed.* Cambridge, England: Cambridge University Press, pp. 490-529, 1992.

[24] A. S. Tanenbaum and M. van Steen Distributed Systems: Principles and Paradigms. Prentice-Hall, Upper Saddle River, N. J. (2002).

[25] E. W. Weisstein. "Moore Graph." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/MooreGraph.html.

[26] B. Parhami, Introduction to Parallel Processing: Algorithms and Architectures. Plenum Press, New York, NY. (1999).

[27] E. W. Weisstein. "Levi Graph." From MathWorld -- A Wolfram Web Resource. http://mathworld.wolfram.com/LeviGraph.html.

[28] Scott Smith and Edward Lo, "Hardware Implementation of a Phase-Retrieval Algorithm", NASA GSFC Summer Student Presentations, July 2003.

[29] B. H. Dean, J. S. Smith, J.G. Budinoff, and L. Feinberg, "Wavefront Sensing and Control Architecture for SPOT (Spherical Primary Optical Telescope)", Proc. of SPIE Vol. 6265 62654F-1, June 2006.

[30] J. S. Smith, B. Dean, and S. Haghani, "Distributed computing architectures for image-based wavefront sensing and 2-D FFTs", Proc. of SPIE Vol. 6274 627421, June 2006.

[31] W. Cheney and D. Kincaid, Numerical mathematics and Computing, Kinciad. Brooks/Cole Publishing Company.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms 2nd Ed*. MIT Press, Cambridge, MA, 2001.

[33] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, N.J., 2002.

[34] S. Kizhner, D. J. Petrick, T. P. Flatley, P Hestnes, M. J. Nilsen, and K. Blank, Pre-Hardware Optimization of Spacecraft Image Processing Software Algorithms and Hardware Implementation, 2002 IEEE Aerospace Conference Proceedings Big Sky Montana, March 9-16, 2002.

[35] S. Smith. "Radiation Tolerant Hardware Analysis", NASA Goddard Space Flight Center / Technology Transfer, Sept 2005.