A Generalized Fault-Tolerant Pipelined Task Scheduling for Decentralized Control of Large Segmented Systems

Paul THIENPHRAPA, Salvador FALLORINA, Zachariah PURNAJO, Emun PRINCE,
Helen BOUSSALIS, Charles LIU, Khosrow RAD, Jianyu DONG, and Yi ZAHO
Structures, Pointing, and Control Engineering Laboratory
Department of Electrical and Computer Engineering, California State University, Los Angeles
Los Angeles, CA 90032, USA

ABSTRACT

The control of complex, flexible structures requires substantial amounts of computational power to achieve precision performance in both space and time. This is due to the fact that such structures are inherently multiple input, multiple output systems whose complexities increase significantly with each additional parameter. The application of decentralized techniques can reduce the computational demands of these systems because multiple lower-order controllers replace a monolithic controller that would otherwise need to account for multitudes of system states in their calculations. Additionally, a decentralized control model provides a framework for the development of parallel control algorithms for both the high performance and fault tolerance of a sophisticated control system.

This paper introduces a novel approach to scheduling computational tasks on processors in a multiprocessor environment. The approach is described in detail and compared against a general straightforward scheduling mechanism. Pipelined task scheduling features increased throughput of control computations and fault tolerance, justifying its use over conventional methods. Both pipelined and straightforward task scheduling algorithms have been applied to a physical control-intensive system; the results indicate a sound design and encourage further work involving pipelined task scheduling.

Keywords: pipelined task scheduling, task mapping, parallel processing, decentralized control, control system.

1. INTRODUCTION

Background

To study the control of large segmented systems, the National Aeronautics and Space Administration (NASA) in 1994 provided funding to establish the Structures, Pointing, and Control Engineering (SPACE) Laboratory at the California State University, Los Angeles. A major goal of the project is to develop a prototype of the James Webb Space Telescope (JWST), which is scheduled for deployment by NASA in the year 2011. As the successor to the currently-active Hubble Space Telescope, a major specification of the JWST is the use of a larger optical mirror to improve upon the Hubble's image range. Because a single large mirror introduces difficulties in the telescope's transportation, the mirror of the JWST will consist of several smaller segments whose overall shape must be dynamically adjusted using an active control system. However, the quality of images collected by the telescope is a function of shaping precision in the optical mirror. Therefore the control processing system must respond to dynamic disturbances in hard real time.

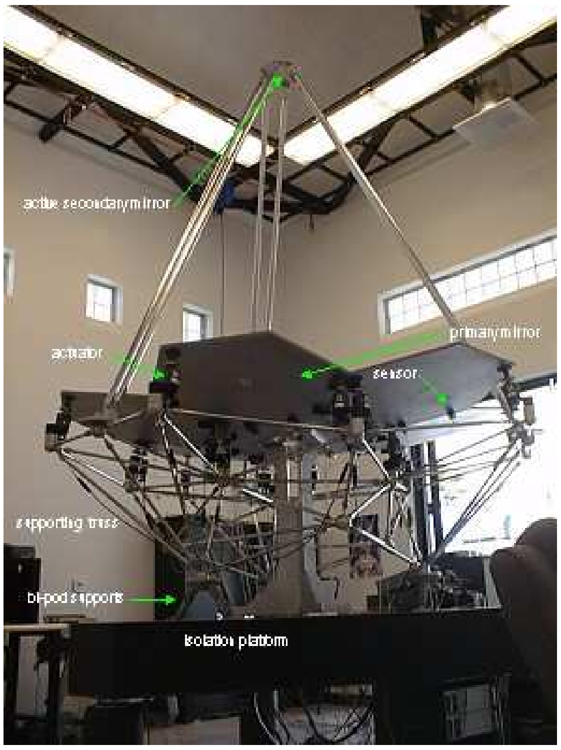

Figure 1. The SPACE testbed

Structure

The SPACE testbed, pictured in Figure 1, resembles a Cassegrain telescope with a focal length of 2.4m. Its performance is designed to emulate an actual space-borne system [1]. The optical mirror of the testbed is composed of a ring of six actively controlled hexagonal panels arranged around a fixed central panel (Figure 2). Fortytwo inductive sensors are placed along the panel edges to provide measurements of relative panel displacements and angles. The three voice-coil linear actuators mounted to the underside of each panel provide three degrees of freedom. The sensors and actuators are connected to the digital control processing system respectively via analog-to-digital and digital-to-analog converters.

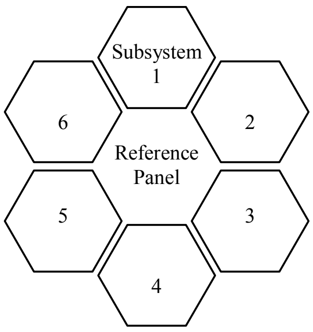

Figure 2. Top view of SPACE testbed primary mirror

Embedded Computer Architecture

The SPACE testbed control processing system utilizes a circuit board configured with four digital signal processors, each with a clock speed of 12.5 MHz. Each processor has its own local memory, in addition to global memory accessible through a common bus (see Figure 3) [9]. High-speed bidirectional communication ports provide message passing capabilities amongst processors [8]. Only one of the four processors has direct access to the sensor input channels (analog-to-digital converters) and the actuator output channels (digital-to-analog converters), giving rise to a master-slave configuration.

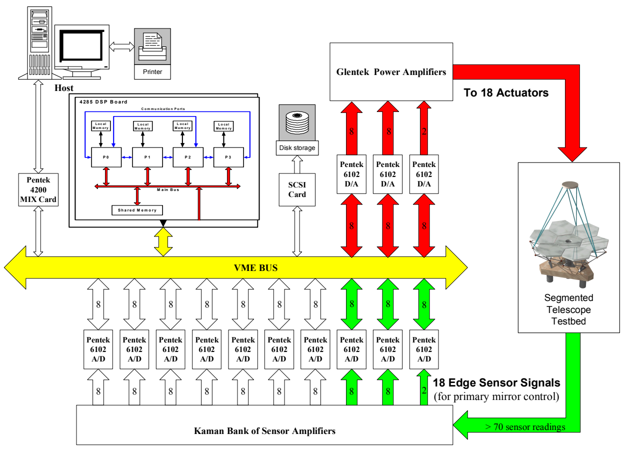

Figure 3. SPACE testbed computer architecture

Performance Requirements

For acceptable performance, the designed control algorithm must complete computations in 0.5-1.0 ms; that is, using a sampling rate of 20-40 times the system bandwidth of 50 Hz [2]. This specification, coupled with the structural complexity of the telescope, attests to the system's real-time computational requirements [10].

2. SHAPING CONTROL

Process Description

Control of the SPACE testbed proceeds as follows. Random external forces displace the mirror's segments into incorrect positions; in the JWST this event would corrupt images produced by the telescope. The sensors detect these displacements and convert them into corresponding electrical signals. Then the analog-to-digital converters sample these voltages at a frequency high enough to feed the control system. The master processor reads the digitized samples and passes them to the control system, which in turn processes them to produce resulting control outputs. The master processor then gathers these results and writes them to the digital-to-analog converters, which in turn convert the values into continuous voltages. Finally the voltages are amplified and sent to the actuators in order to reposition the displaced panels into a correct configuration. The control system performs this *control cycle* iteratively and continuously to actively maintain precise mirror shaping.

Decentralized Control Model

As described earlier, the testbed consists of a large number of structural components necessitating multiple input and output channels. In centralized mathematical models this configuration could potentially involve hundreds of states. Even after the application of classical model reduction techniques, centralized models of the testbed involve over 200 states. Consequently, the design of control laws based on conventional methods becomes exceedingly challenging. As such the consideration of a decentralized control model is particularly interesting as a viable approach in circumventing these difficulties [5]. As the testbed's optical mirror is segmented into six hexagonal panels, the system is naturally decentralized by treating each panel and its peripheral components as an isolated subsystem [3] [4]. Each subsystem, whose behavior can be governed by simpler local controllers, is identified by its set of sensor inputs and actuator outputs. Under decentralized control, a single 200th-order controller is replaced by six 12th-order local controllers, substantially reducing the computational complexity of the overall system and promoting parallel processing for high performance and fault tolerance.

Equation 1, derived in [12], is the discrete state-space representation of the decentralized control computations that must be performed for each subsystem in each control cycle. The $x(k)$ are state variables, the $e(k)$ are sensor signals, and the $u(k)$ are control results. The constant matrices Φ, Ψ, C, and D account for the geometry of the structure. Note that controlling a complex structure remains a nontrivial task even after decentralization reduction, further highlighting the importance of parallel processing.

$$
\begin{aligned}
x_{12x1}(k+1) &= \Phi_{12x12}x_{12x1}(k) + \Psi_{12x3}e_{3x1}(k) \\
u_{3x1}(k) &= C_{3x12}x_{12x1}(k) + D_{3x3}e_{3x1}(k)
\end{aligned}
\qquad (1)
$$

While decentralization helps reduce the computational complexity of control calculations, parallel processing can help increase control throughput [6]. When both concepts are applied to a sophisticated system such as the SPACE testbed, the end result is a real-time, robust system. It is highly responsive because control tasks can be distributed among several processors in parallel, decreasing the turnaround time of control cycles. It is fault tolerant because the failure of processors results in gradual performance degradation rather than complete failure. Furthermore, a decentralized control model exposes opportunities for parallel processing, so the two concepts are complementary.

3. REAL-TIME SHAPING MECHANISMS

In order to shorten the time needed to perform shaping calculations, parallel processing is used. As mentioned before, the decentralization of the control system reduces the complexity of the calculations. The decentralization of the structure into six subsystems creates six independent *control tasks*, or simply *tasks*, that must be performed continuously. This approach utilizes the full capacity of the computing system, which houses four digital signal processors.

In each control cycle we wish to distribute the number of tasks, *M*, among the number of available processors, *P*. Based on the decentralized model we make the following assumptions:

1. Each task is not further decomposed.
2. Computational complexities of all tasks are identical
3. Each task takes one control cycle to complete.
4. There are no data dependences among tasks [7].

Straightforward Task Scheduling

In using a straightforward parallel implementation of decentralized control, processors are statically mapped to subsystems such that each processor performs control computations for its respective subsystem only. Although increased performance is realized when multiple processors are used in this manner, inefficiency arises if the number of subsystems *M* is not an integer multiple of the number of processors *P*, $M > P$. Load imbalance occurs in such cases because exactly (M mod P) processors are necessarily responsible for controlling more subsystems than other processors. Optimality is sacrificed because processors with lighter loads are idle while waiting for processors with heavier loads (see Figure 4). Furthermore this mechanism does not lend itself favorably towards fault tolerance because the failure of a single processor will result in the failure of its corresponding subsystem, an unacceptable scenario.

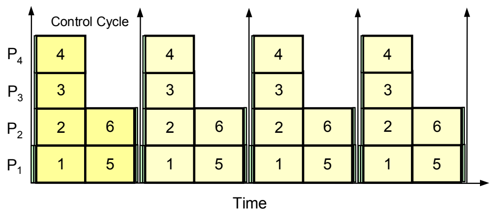

Figure 4. Load imbalance due to straightforward task scheduling

Pipelined Task Scheduling

Capitalizing on the nature of decentralized control, a more sophisticated parallel design approach can be used to provide both load balancing (i.e. improved performance) and fault tolerance. At control cycle *i*, a given task *i*, $1 \leq i \leq M$ (corresponding to subsystem *i*), is scheduled on processor P_1. In the next *P*-1 control cycles that task is scheduled on processors P_2, P_3, …, and P_p, in that order. The corresponding data streams are distributed in a cyclic fashion. This process is repeated every *P* control cycles, resulting in a task scheduling scheme that resembles a pipeline. Figure 5 illustrates this approach with *P*=4 and *M*=6.

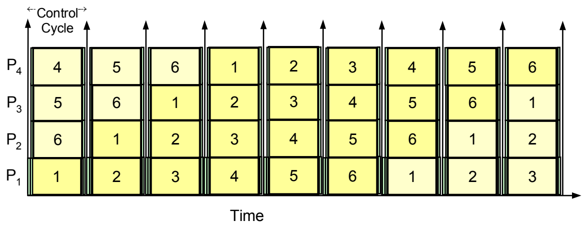

Figure 5. Pipelined task scheduling

High Performance: An advantage of implementing pipelined task scheduling is improvement in the average number of tasks completed. As mentioned before, in a straightforward parallel implementation some processors are idle during some particular control cycles whenever the number of tasks is not an integer multiple of the number of processors. On the other hand, with the pipelined parallel implementation, there are no idle processors at any given control cycle. In the straightforward parallel approach, processors may be assigned one or more tasks per control cycle; whereas in the pipelined approach processors are always assigned a single task. This apparent speed-up is essential in achieving the real-time shaping requirements.

Fault Tolerance: To accommodate the occurrences of processor failures and their recovery, task rescheduling mechanisms are used. If during a single control cycle multiple processors P_k, should happen to fail, then each remaining processor P_j must stall execution for one control cycle for *each* failed processor where $j < k$. To illustrate, let P_0, …, P_4 denote the five processors in a system. If failures are detected in processors P_1 and P_3, then processor P_2 would have to stall execution for one control cycle ($P_2 < P_3$), while processor P_0 would need to stall for two ($P_0 < P_1, P_3$). Processor P_4 would proceed as usual since $P_4 > P_1, P_3$. Recovery of previously-failed processors within a single control cycle can be handled similarly: functioning processors would stall execution for one control cycle for each recovered processor with lower-number identifiers. This stalling approach is used to preserve the pipelined task sequence and to simplify the buffering complexity – these are efforts are necessary because the input to a subsystem task depends partially on the output of that task from the previous cycle.

Caveats: Although the above pipeline scheme increases throughput and provides mechanisms for handling processor failure and recovery, it introduces new shortcomings. Because subsystems are not statically mapped to specific processors, each control cycle necessarily incurs an increased amount of communications overhead due to dynamic data distribution. Furthermore, if the number of tasks *M* is much greater than the number of processors *P*, then there will be a long delay between two successive iterations of each individual task, and precision alignment may be sacrificed. These flaws will be addressed in future work.

Pipelined processing techniques promise to tolerate failure of one or more processors because processors are no longer tied to specific subsystems. Instead, control computations are distributed amongst the processors in a manner that maintains the pipeline flow structure. Using the pipelined approach, a better linearity of throughput is observed as the number of processors increases.

4. RESULTS AND ANALYSES

Both the straightforward and pipelined task scheduling procedures have been implemented and deployed on the SPACE testbed. The experiments focus on analyzing the program throughputs as well as the control system responses to an initial static disturbance.

Execution Profile

The runtime execution time data, shown in the four tables below, were recorded from salient points in the programs. Tables 1 and 2 display the clock cycles each algorithm consumes in a single control cycle, which is described in Section 2 (Process Description). The tasks are dissected into major subtasks for comparison across various configurations (i.e. scheduling scheme and processor count). Also shown are the control cycle times in milliseconds, based on a 12.5 MHz clock speed. Figure 6 shows this data in terms of throughput. Note the better linearity of throughput provided by pipelining.

Processor Count:	1	2	3	4
Read Sensors	192	192	194	192
Assign Tasks	14	363	340	540
Compute Tasks	24724	12362	8832	8832
Collect Results	12	339	1060	1260
Write Actuators	285	285	285	285
Time (ms):	2.018	1.083	0.857	0.889

Table 1. Straightforward task scheduling execution in number of clock cycles

Processor Count:	1	2	3	4
Read Sensors	194	194	192	194
Assign Tasks	49	474	892	1324
Compute Tasks	3194	3194	3194	3197
Collect Results	42	313	679	1075
Write Actuators	282	282	283	283
Time (ms):	0.301	0.357	0.419	0.486

Table 2. Pipelined task scheduling execution in number of clock cycles

Straightforward task scheduling sees an improvement in execution time as the processor count is increased. The improvement is not linear because message passing overhead is introduced with each additional processor. When increasing from three to four processors, the control cycle time theoretically cannot improve due to load imbalance; performance actually degrades due to message passing involving the deadweight processor.

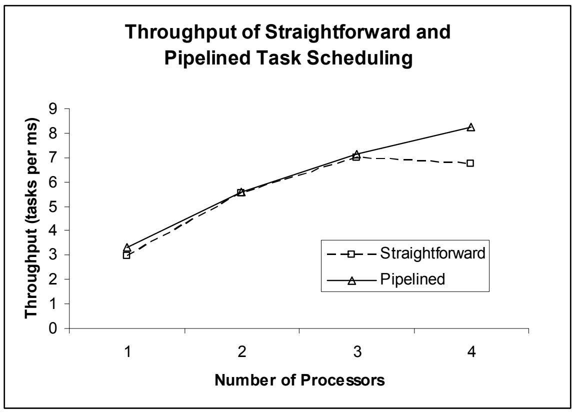

Figure 6. Throughput of control tasks

In the pipelined case, message passing time increases with processor count while computation time remains constant, resulting in slower control cycle times for increasing numbers of processors. Of course, more tasks can be processed when more processors are used since each processor is assigned one task per control cycle. As expected, the throughput for pipelined task scheduling is higher than that of the straightforward method.

As shown in Tables 3 and 4, a greater fraction of the total time is used for message passing when the number of processors is increased. At one extreme, 98% of the execution time is spent performing control computations, while at the other extreme only 53% of the work is control. However, the control cycle time of the former case is over four times slower than that of the latter.

Processor Count:	1	2	3	4
Read Sensors	0.76	1.42	1.81	1.73
Assign Tasks	0.06	2.68	3.17	4.86
Compute Tasks	98.01	91.29	82.46	79.50
Collect Results	0.05	2.50	9.90	11.34
Write Actuators	1.13	2.10	2.66	2.57

Table 3. Straightforward execution in %

Processor Count:	1	2	3	4
Read Sensors	5.16	4.35	3.66	3.19
Assign Tasks	1.30	10.63	17.02	21.80
Compute Tasks	84.92	71.66	60.95	52.64
Collect Results	1.12	7.02	12.96	17.70
Write Actuators	7.50	6.33	5.40	4.66

Table 4. Pipelined execution in %

Depending on the number of subsystems and on real-time requirements, there is an eventual tradeoff between computation and message passing, limiting the number of processors that can be effectively used. Given the four processors in the SPACE testbed, three processors is best when implementing straightforward scheduling, while four is best for pipelined task scheduling.

Control System Response

Information about the response of the control system to an applied static disturbance was gathered by recording all sensor data samples and plotting them offline. The plots, included below, demonstrate the functionality for both straightforward and pipelined task scheduling. Figure 7 shows the results of using straightforward scheduling with one, two, and three processors, while Figure 8 shows the results of pipelined scheduling with two, three, and four processors. In all cases, the initial disturbance can be seen as a large displacement at the beginning of the experiment. Over time the control system stabilizes each panel; in the plots this event appears as a horizontal asymptote. The fuzziness of the graphs represents a small amount of acceptable steady-state error. The offset in the displacement axes between the two plots is likely caused by the difficulty of reproducing disturbances to within a fraction of a millimeter.

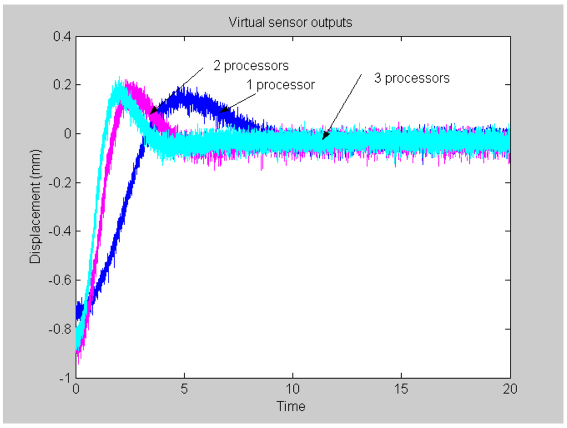

Figure 7. Control response – straightforward scheduling

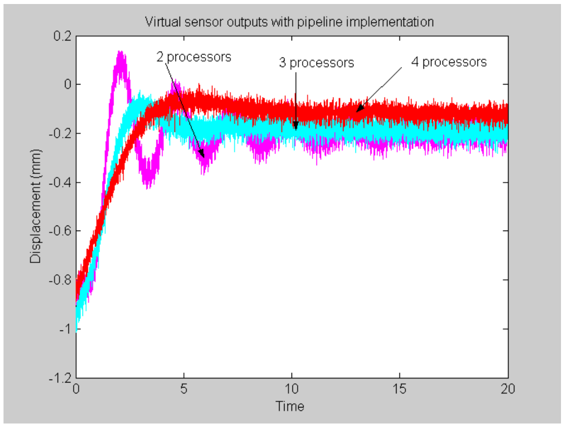

Figure 8. Control response – pipelined scheduling

For straightforward task scheduling, it is apparent that using more processors to perform calculations results in faster shaping. This same trait can also be seen in pipelined task scheduling, with significant speedup when four processors are utilized. Comparing the pipelined and straightforward approaches with three processors, the difference in the shaping time is minimal.

The shape of the stabilization plot in the straightforward case is the same for different numbers of processors because all six subsystems are processed in every control cycle. On the other hand, stabilization under pipelining exhibits different ripple delays for different numbers of processors. This is likely the result of different numbers of subsystems being processed per control cycle, depending on the number of processors present. Particularly in the two-processor pipelined scenario, the system appears to stabilize slowly due to an underdamped response, as only two subsystems are controlled per control cycle.

Panels converge to a steady state only slightly faster in the straightforward scheme than in the pipelined technique. For example, the speculation that the straightforward parallel architecture with three processors would perform shaping faster the pipelined architecture with three processors holds true. Because the shape response is satisfactory and the time data is acceptable, these experiments encourage further work in pipelined task mapping. After the application of message passing optimizations, the pipelined implementation can demonstrate performance rivaling that of straightforward scheduling.

5. CONCLUSIONS AND FUTURE WORK

Proper execution and reasonable performance are realized under pipelined task scheduling, as the results indicate. This elementary proposed scheme, however, is not without weaknesses. Some overhead is incurred when using dynamic scheduling due to the extra message passing activities. The results show that this overhead is minor but can be improved upon nonetheless. Also, a starvation problem arises with small numbers of processors because subsystems are not controlled in every control cycle. The problem manifests itself as an underdamped response. To eliminate this flaw, a more advanced pipelining scheme is being studied – group pipelining. In this method, small sets of tasks, as opposed to single tasks, are assigned to individual processors in a single control cycle. Both the communications overhead and the delay between two successive iterations of each task are thus reduced at the lesser expense of an increased control cycle time. Designing the discrete controllers to accommodate delay cycles represents another option.

Experimental data show that pipelined task scheduling yields acceptable responses while allowing for dynamic reconfiguration to tolerate failure. Continued work will evolve this method into a promising solution for the SPACE testbed and other applications.

In addition to message passing improvements, group pipelining, and general optimizations, future work on the testbed's embedded computing system includes fault detection and reconfiguration.

This work was supported by NASA under Grant URC NCC 4158. Special thanks go to all the faculty and students associated with the SPACE Laboratory for their technical and moral support.

6. REFERENCES

[1] H. Stockman, **The Next Generation Space Telescope: Visiting a Time When Galaxies Were Young**, June 1997.

[2] H. Boussalis, **Decentralization of Large Space-borne Telescopes**, Proceedings of the 1994 SPIE Symposium on Astronomical Telescopes, 1994.

[3] H. Boussalis, Mirmirani, M., Chassiakos, A., and Rad, K., **The Use of Decentralized Control in the Design of a Large Segmented Space Reflector**, Control and Structures Research Laboratory, California State University, Los Angeles Final Report, 1996.

[4] H. Boussalis, Mirmirani, M., Rad, K., Morales, M., Velazquez, E., Chassiakos, A., and Luzardo, J.A., **The Use of Decentralized Control in the Design of a Large Segmented Space Reflector**, NASA URC Technical Conference, Albuquerque, NM, 1997.

[5] D. Siljak, **Decentralized Control of Complex Systems**, New York, Academic, 1991.

[6] I. Foster, **Designing and Building Parallel Programs**, Addison-Wesley Publishing Company, Inc., 1995.

[7] J. Hennessy, and Patterson, D., **Computer Architecture: a Quantitative Approach**, Morgan publishing, San Francisco, CA, 1990.

[8] **TMS320C4x User's Guide**, Texas Instruments, Inc., 1991.

[9] **Octal TMS320C40 Processor Manual**, Pentek, 1998.

[10] J. Liu, **Real-time Systems**, Prentice-Hall, Inc., 2000.

[11] H. Boussalis, Kosmatopoulos, E.B., Mirmirani, M., and Ioannou, P.A., **Adaptive Control of Multi-variable Nonlinear Systems with Application to a Large Segmented Reflector**, ACC 1998.

[12] S. Fallorina, Boussalis, H., Liu, C., Rad, K., Dong, J., Nasser, D., and Thienphrapa, P., **A Generic Pipelined Task Scheduling Algorithm for Fault-tolerant Decentralized Control of a Segmented Telescope Testbed**, Proceedings of ASME 2004 DETC and CIE, 2004.