Proceedings of DETC'04
ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference
September 28 – October 2, 2004, Salt Lake City, Utah USA

DETC2004-57701

A GENERIC PIPELINED TASK SCHEDULING ALGORITHM FOR FAULT-TOLERANT DECENTRALIZED CONTROL OF A SEGMENTED TELESCOPE TESTBED

Salvador Fallorina **Helen Boussalis**

Charles Liu **Khosrow Rad** **Jane Dong**

Dani Nasser **Paul Thienphrapa**

Structures, Pointing, and Control Engineering Laboratory
Department of Electrical & Computer Engineering
California State University, Los Angeles
5151 State University Drive
Los Angeles, California 90032 USA

ABSTRACT

Control of complex structures requires high computational power to achieve real-time performance. Through decentralized techniques, a complex structure can be controlled by multiple lower-order local controllers, leading to reduced computational complexities. Furthermore, a decentralized approach can both simplify the development of parallel controllers and facilitate fault-tolerant designs. In our research, multiple digital signal processors are employed in a NASA-sponsored segmented telescope testbed to increase the throughput of control tasks. Although increased performance is realized when subsystems are statically mapped to specific processors for control, inefficiency arises if the number of subsystems M is not an integer multiple of the number of processors P ($M > P$) because (M mod P) processors are necessarily controlling more subsystems than others. Optimality is sacrificed because processors with lighter loads wait for processors with heavier loads. 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 subsystem.

This paper describes the design and implementation of a pipelined task mapping approach for the decentralized control of a segmented reflector telescope testbed. In our pipelined processing implementation only four of the six subsystems are processed in any given control cycle; the two unprocessed subsystems in each cycle propagate about the system in a round-robin fashion, so processors are never idle. Fault tolerance is facilitated because processors are no longer tied to specific subsystems. Instead, control computations are distributed dynamically such that the pipeline flow structure is maintained. The implementation of a watchdog technology is presented for detecting the possible processor failures. Experimental results are shown comparing the performance of the pipelined and straightforward approaches. The throughput of the system has also been estimated on a system with a larger number of processors. Such estimation shows the linearity of speedup achieved by using the pipelined approach.

Keywords: decentralized control, pipelined task mapping, pipelined task scheduling, parallel processing, fault detection, fault tolerance, watchdog

1. INTRODUCTION

As the successor to the Hubble Space Telescope (HST), the James Webb Space Telescope (JWST), formerly known as Next Generation Space Telescope (NGST), requires a larger light-gathering mirror capable of detecting faint signals [1].

Due to the manufacturing and deployment difficulties of using a monolithic piece of glass, the primary mirror of the JWST will consist of several smaller reflecting panels. However, a reflector built from segments relies on an active control system for precision alignment of the optical surface. This control system is responsible for achieving high-precision figure control and maintenance of the reflector surface to a calibrated parabolic reference figure in a dynamic disturbance environment.

To study the control of such large segmented optical 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 (CSULA). One of the major goals of this project is to design and fabricate a testbed that resembles the complex dynamic behavior of a segmented space telescope.

In the present study, a decentralized control design approach has been deployed to provide improved load balancing and fault tolerance. In this approach, each individual controller task is scheduled in a pipelined fashion among the available processors, and the sequential order of its control cycles can be observed. This method allows perfect load balancing for any combination of the numbers of nodes and tasks. Task mapping and rescheduling are used to handle the cases when one or more processors fail. A similar technique can also be employed to optimize the parallel processing of tasks when failed processors are recovered.

This paper is organized as follows: Section 2 presents the system description of the SPACE testbed. Section 3 gives an overview of the system decentralization. In Section 4, the parallel design of an embedded platform for decentralized control is described. Various task mapping and scheduling approaches are proposed and analyzed. Section 5 addresses the implementation of the proposed parallel design and presents the results. Finally conclusions are given in Section 6.

2. SYSTEM DESCRIPTION OF THE SPACE TESTBED

Figure 1 shows the major features of the SPACE testbed, which is designed to emulate a Cassegrain telescope of 2.4-meter focal length with performance comparable to an actual space-borne system [4]. The primary mirror of the SPACE testbed, supported by a lightweight truss, is composed of a ring of six actively controlled hexagonal panels arranged around a fixed central panel. The primary mirror is fitted with an ensemble of 42 inductive sensors that provide measurements of relative panel displacement and angle. Three voice-coil linear actuators are mounted on each panel to provide three degrees of freedom for each segment. The SPACE testbed's data acquisition system consists of digital signal processors and dual A/D and D/A converter packages from Pentek.

Because the quality of images collected by the JWST is a function of shaping precision in the primary mirror, the control processing system must respond to stimuli within hard real-time constraints. To achieve real-time performance, the designed control algorithm needs to be implemented using a sampling rate of 20-40 times the system bandwidth [10]. The SPACE testbed has a 50 Hz system bandwidth, so control calculations must be performed within 1.0 millisecond. While a single high-performance processor can provide sufficient computational power, this real-time requirement can also be achieved through the application of parallel processing. The use of multiple low-end processors provides an economic solution to increase throughput, thereby improving the system's responsiveness to panel disturbances. Furthermore this arrangement allows the implementation of fault tolerance schemes.

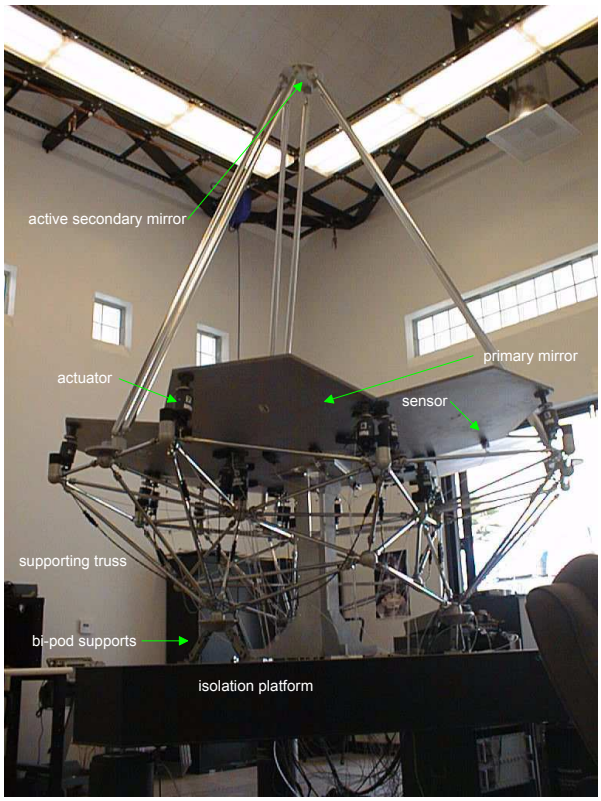

Figure 1: SPACE testbed

3. APPLICATION DESCRIPTION

3.1. SYSTEM DECENTRALIZATION

The control of large flexible structures has been an important problem in a variety of space programs. As described in Section 2, the SPACE testbed consists of a large number of structural components, as well as sensors and actuators leading to mathematical models that involve hundreds of states. Even after the application of model reduction techniques, the centralized model of the telescope has over 200 states complementing 18 edge sensors and 18 actuators. Consequently, the design of control laws based on conventional methodologies becomes exceedingly difficult. Decentralized control appears to be a viable approach in circumventing the difficulties related to the dimensionality problem.

Due to the nature of the structure, decentralized techniques are employed for the development of control laws to accomplish both fault tolerant precision pointing and reflector shape control. Decomposition techniques are implemented, resulting in physical decentralization of the structure into six lower-order subsystems.

The equations of motion of the system assume the form

$$M\ddot{\underline{\delta}} + K\underline{\delta} = B_1\underline{u} + B_2\underline{d}\,. \qquad (1)$$

M is the mass matrix, K is the stiffness matrix, $\underline{\delta}$ is a position coordinate vector, B_1 and B_2 are force amplitude matrices, $\underline{u}$ is a control-input vector and $\underline{d}$ is a disturbance vector.

For control purposes the following state-space representation of the composite system is derived from eq. (1):

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{2}$$

Decomposing the system (2) into six subsystems according to the physical structure illustrated in Figure 2 yields eq. (3).

$$\begin{aligned} \dot{x}_i &= A_{ii}x_i + \sum_{i=1}^{6} A_{ii}x_i + B_{1i}u_i + B_{2i}d \\ y_i &= C_i x_i \end{aligned} \tag{3}$$

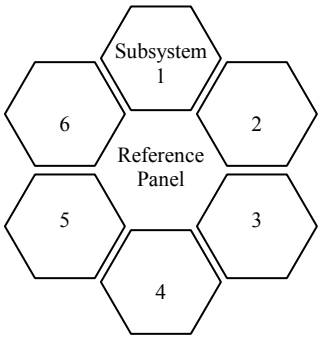

Figure 2: Top view of SPACE testbed primary mirror

The first term,

$$\dot{x}_i = A_i x_i \tag{4}$$

is the isolated component of eq. (3), and

$$x_i = \left[\delta_i^T \;\; \dot{\delta}_i^T\right]. \tag{5}$$

As illustrated in Figure 3, the system is naturally decentralized by treating each of the six peripheral segments of the primary mirror and its associated supporting structure as an isolated subsystem. Each subsystem is identified by three command inputs to the actuators and three outputs which are measured by the edge sensors. Local control algorithms are developed for each of the six isolated subsystems.

To follow the consistency of discrete control algorithms, the system is represented in the discrete state form of eq. (2).

$$\begin{aligned} x_{nx1}(k+1) &= \Phi_{nxn}x_{nx1}(k) + \Psi_{nxm}u_{mx1}(k) \\ y_{rx1}(k) &= C_{rxn}x_{nx1}(k) \end{aligned} \tag{6}$$

This discrete state equation represents an n^{th}-order system with m inputs and r outputs, where Φ is the state transition matrix, $x(k)$ is the state vector, $u(k)$ is the control signal vector, and $y(k)$ is the output vector.

As an implementation example, we refer to the replacement of a single 200th-order centralized controller that would be designed to control the primary mirror by six 12th-order local controllers running simultaneously to maintain the precision of the primary mirror shape. This method reduces the required computational difficulty and facilitates both parallel implementations and fault-tolerance. The control calculations for each of the six subsystems are represented by

$$\begin{aligned} x_{12x1}(k+1) &= \Phi_{12x12}x_{12x1}(k) + \Psi_{12x3}e_{3x1}(k) \\ u_{3x1}(k) &= C_{3x12}x_{12x1}(k) + D_{3x3}e_{3x1}(k) \end{aligned} \tag{7}$$

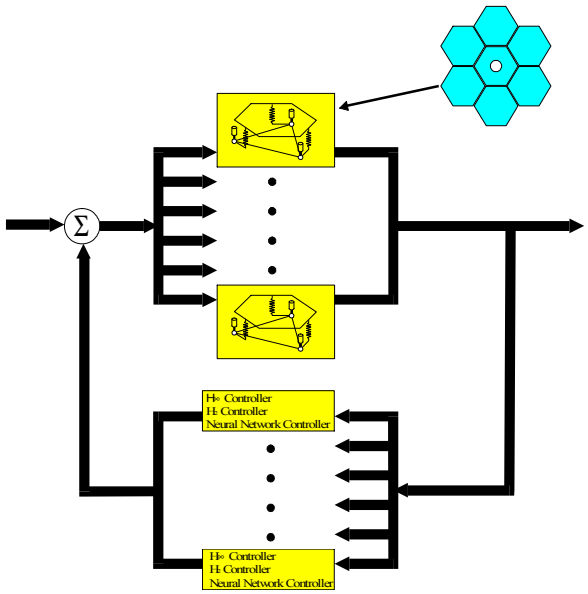

Figure 3: Decentralized control system block diagram

3.2. CONTROL PROCESS FOR THE SPACE TESTBED

The single-processor implementation of a decentralized controller for the SPACE testbed is depicted in Figure 4. After some initialization procedures, the control cycle starts by reading edge sensor signals which are geometrically transformed into virtual points, which in turn indicate the displacement and positions of the panels. The next sequence of tasks consists of calculating control commands for six subsystems. A single task involves the control calculations for a single subsystem (eq. (7)). Resultant control signals are sent to actuators to properly position the panels. These tasks must take place within a specified sampling period.

The disadvantages of sequential implementation include extended execution time and lack of fault-tolerance. However, within this cycle, the decentralized controllers present opportunities for parallel execution. Through parallelization, tasks can be conveniently assigned to available processors.

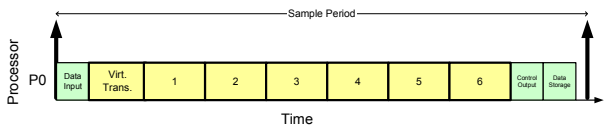

Figure 4: Sequential program of decentralized control

4. FAULT-TOLERANT PARALLEL DESIGN

Parallel processing is applied in order to achieve decentralized real-time performance. Based on our model of decentralized control, M tasks will be executed in parallel among P processors in an iterative fashion.

4.1. COMPUTATIONAL FEATURES

The computational features of the decentralized tasks are summarized as follows:

1. Each task is not further decomposed.
2. The computational complexities of all tasks are identical.
3. Each task completes a control cycle. The next control cycle cannot be scheduled until the next sample of the input signals has been received by its corresponding sensors.
4. There is no significant data dependency among the tasks. Different tasks can be updated in different control cycles in an arbitrary order.

Dependencies between the subsystems are negligible. The six panels of the primary mirror do not need to cooperate with each other to achieve precision shaping because the local controllers align the subsystems against a calibrated parabolic reference.

4.2. A STRAIGHTFORWARD TASK MAPPING AND SCHEDULING APPROACH

Based on the features described in Section 3.1, one straightforward approach is to assign the tasks to the processors as evenly as possible within the same control cycle. If M is a perfect multiple of P, then perfect load balancing is achieved by evenly distributing the tasks among the processors. Otherwise, a subset of the processors will be idle during certain control cycles due to the load imbalance. Such a scenario is illustrated in Figure 5 with P = 4, and M = 6. In this approach the length of the control cycle must extend to the amount of time required by the slowest processors, that is, the processors with the heaviest loads. Optimal capacity is not achieved in this situation because processors with lighter loads are idle while waiting for processors with heavier loads to complete their tasks.

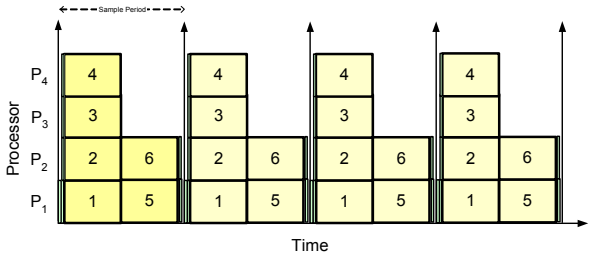

Figure 5: Straightforward parallel processing (*P*=4, *M*=6)

Furthermore this mechanism does not lend itself favorably towards fault tolerance because the failure of one processor will result in the failure of its corresponding subsystems, an unacceptable scenario.

4.3. PIPELINED TASK MAPPING AND SCHEDULING

Capitalizing on the nature of decentralized control, a more sophisticated parallel design approach has been deployed to provide load balancing and fault tolerance. Using this approach, each processor has the capability to handle all control calculations for any subsystem. Each individual task is scheduled in a pipelined fashion among the available processors, so the sequential order of the control cycles can be observed. Figure 6 illustrates the idea of the pipelined task mapping and scheduling.

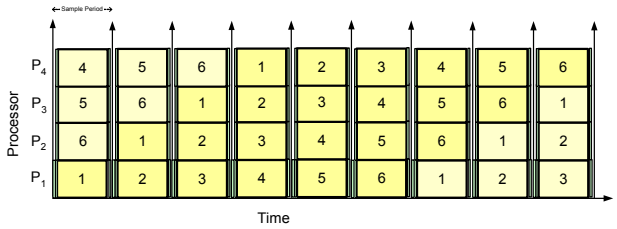

Figure 6: Pipelined parallel processing (*P*=4, *M*=6)

This approach allows perfect load balancing for any numbers of P and M. Furthermore, this technique promises to tolerate failure of one or more processors, since any one of the functioning nodes is able to control any subsystem. In this design, each processor keeps a copy of all of the constant matrices in its local memory; i.e., Φ*'s*, ψ*'s, C'*s, and *D'*s for all M controllers. While large values of M require more local memory for each processor, this approach reduces the traffic on the common bus due to the accesses of the shared memory space. On the other hand, the sensor input data, state vectors, and the calculated control output signals are passed between the master and slave processors via the high-speed interprocessor communications ports. The overhead incurred using message-passing is smaller than the latency of using shared global memory, because using global memory in this manner introduces considerable bus contention. Note that the message-passing overhead is negligible relative to control computations, making it a practical communications solution.

Note that in this real-time embedded system, the control signals of a specific decentralized controller are used to trigger the actuators to move the corresponding panel. Sensor readings of panel displacements are read and represented by an input vector. The input vector is then used for the operation of the next iteration of the control output. Thus, there is an automatic serialization between the accesses of the global vectors; no racing problem can occur.

4.4. PIPELINED TASK MAPPING AND SCHEDULING WITH FAULT TOLERANCE

Figure 7 shows the generalized tasking mapping and scheduling using the pipelined approach with M tasks and P processors, where M and P, $M > P$, can be any arbitrary positive integers. Note that task i ($1 \le i \le M$) is initially scheduled at cycle i on processor P_1. It is then scheduled for the following P-1 consecutive cycles on processors P_2 through P_p. After the first P times of the scheduling, task i is re-scheduled back to processor P_1 at cycle $M+i$ and ripples to the rest of the P-1 processors based on the pattern described in the first P times of the scheduling.

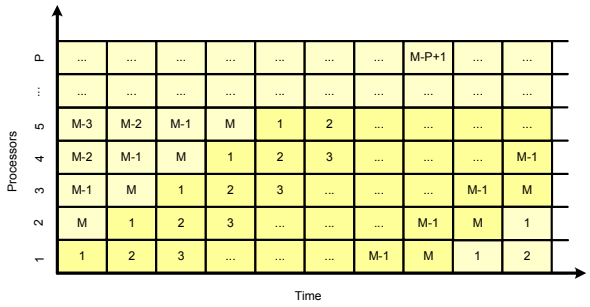

Figure 7: Generalized pipelined parallel processing

Task remapping and rescheduling are used to accommodate the occurrences of processor failures and their recovery. For instance, if P_i fails during a certain control cycle, the processors P_1 through P_{i-1} must stall for one control cycle. Then, the processor identifiers of processors P_i through P_p will be decremented by one. The stalling approach is used to simplify the buffering complexity since the input to a cycle depends on the output of its previous cycle. The stalling also preserves the original pipeline sequence, which allows the control of adjacent tasks during one cycle. Figures 8 and 9 illustrate this principle in action as P_3 fails at time period t_5. Controllers 4 and 5, which were originally meant to be handled by processors P_2 and P_1 respectively at period t_5, are shifted to the next control cycle.

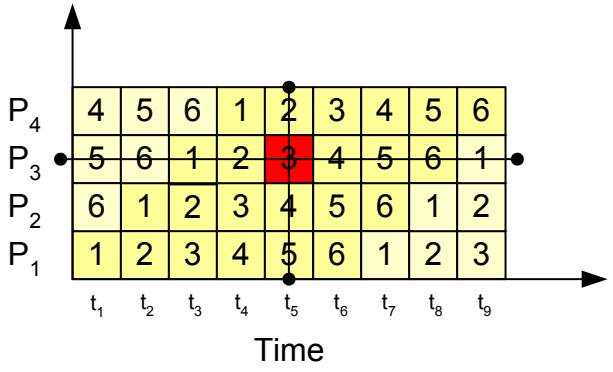

Figure 8: Failure of P_3

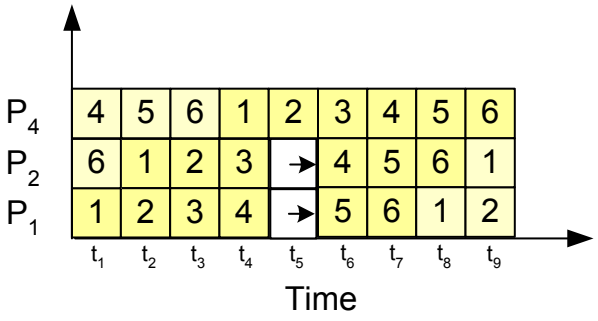

Figure 9: P_1 and P_2 stalls after failure of P_3

This stalling technique can be used in the event of a failed-processor recovery. Suppose that a processor is recovered between P_i and P_{i+1}. The newly inserted processor will be assigned an identifier, P_{i+1}. Accordingly, the old identifiers P_{i+1} to P_P will be incremented by one. Also, if P_i has performed task j in the previous cycle, the newly inserted processor P_{i+1} will perform the same task, i.e. task j, in the current cycle. Then, the processors P_{i+2} through P_p will stall for the current cycle. Using this approach the controller sequences can be, consequently, resumed. This feature is illustrated in Figure 10.

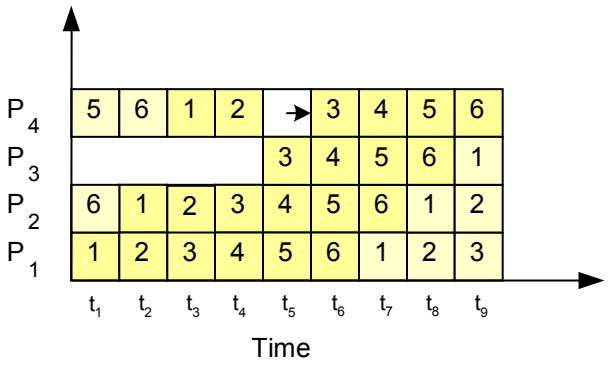

Figure 10: P_4 stalls after addition of P_3 at t_5

More generally, if multiple processors P_k should fail during a given control cycle, then each remaining processor P_j must stall execution for one control cycle for *each* failed processor where $j < k$. Let P_0, …, P_4 denote the five processors in an example system. If during a control cycle processor P_2 fails, then processors P_0 and P_1 would have to stall for one execution cycle, while processors P_3 and P_4 would continue execution as usual. Now consider the example shown in Figure 11 involving multiple processor failures. 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 $P_4 > P_1, P_3$. Such an arrangement of pipelined task mapping, in addition to minimizing processor idle time, allows the system to execute control processing correctly and sequentially despite experiencing the failure of one or more processors within the same control cycle. Recovery of multiple 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.

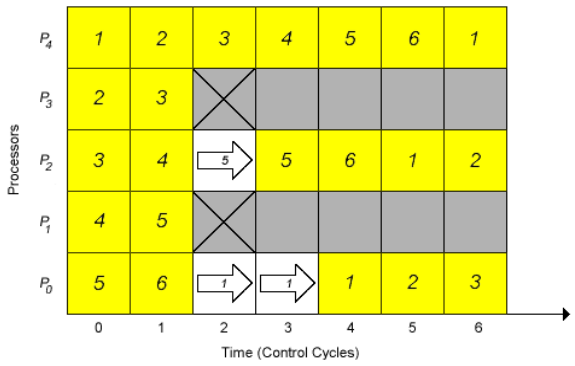

Figure 11: Handling multiple failures in one cycle

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The standard and pipelined approaches are being implemented on the SPACE testbed with $P = 4$, $M = 6$. This configuration promotes efficient use of available processors and is resilient to processor failures.

5.1. COMPUTER ARCHITECTURE

The real-time embedded system, shown in Figure 12, utilizes a Pentek 4285 board that is configured with four TMS320C40 digital signal processors [9]. Each processor has its own local memory in addition to a globally shared memory space. High-speed bidirectional communication ports allow direct message passing between processors, while the shared global random access memory (RAM) is accessed through a common bus. Here, the processors are arranged in a tree topology with one processor configured as the master since only it has access to the digital-to-analog (D/A) and analog-to-digital (A/D) converters. These signal converters, in turn, are connected to the actuator amplifiers and sensors respectively. The control of the primary mirror requires 18 sensor inputs and 18 actuator outputs; this multiple input-multiple output (MIMO) system attests to the computational requirements of this application.

The individual processing units utilized are 32-bit floating point digital signal processors that incorporate on-chip parallel processing features to obtain high individual performance as well as making them conducive to multiple processor configurations. In particular, the high-speed bidirectional communication ports provide zero glue logic, direct communication between processors. Other C40 specifications include: 40-ns instruction cycle time, 60 MFLOPS, and 6-channel direct memory access (DMA) [8].

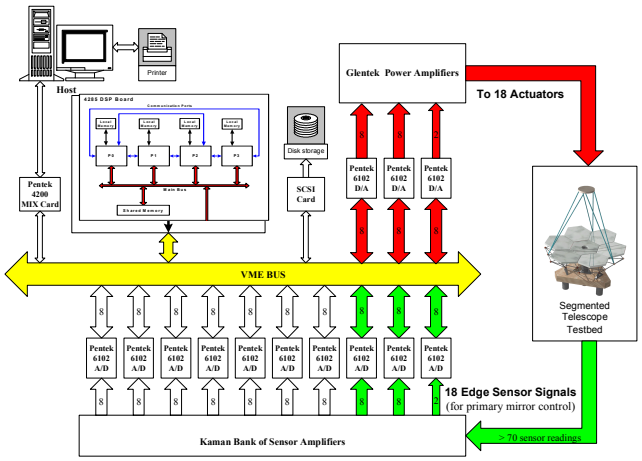

Figure 12: Computer architecture

The low processor count is appropriate primarily for this type of computationally intensive application. Message passing of input, state, and output vectors is minor in comparison to the calculation of state-space equations.

Using this architecture, the processors are organized into a tree topology as presented in Figure 13. Based on the current configuration of the embedded system, one processor is assigned the role of master which handles the sensor data input and control signal output.

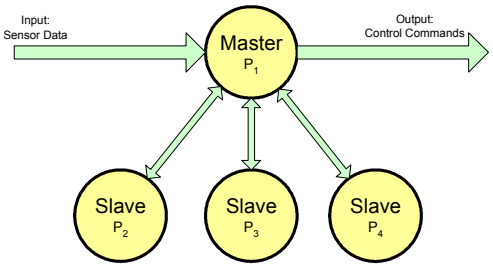

Figure 13: Decentralized control tree topology

For each control cycle, the master receives input data from the sensors and distributes them to the slave processors. Each of the nodes then performs the control signal calculations. The resulting signals are collected by the master, which then sends the output data to the actuators.

5.2. PARALLEL IMPLEMENTATION

Using a straightforward parallel approach, the six separate controllers can be conveniently assigned to available processors. Figure 14 depicts a straightforward task scheduling approach including input and output (I/O) and message passing tasks. Each processor is statically mapped to a predetermined subsystem such that it performs control computations for only its respective subsystem(s). Each of the nodes has the Φ, Ψ, C, and D matrices only for its assigned subsystem(s) in local memory. In each control cycle, the master passes digitized sensor data, as well as state vectors, to the slave processors. After the control calculations are performed, the resulting state vectors $x(k)$ and output control signals $u(k)$ are gathered together by the master, the latter used to trigger the actuators.

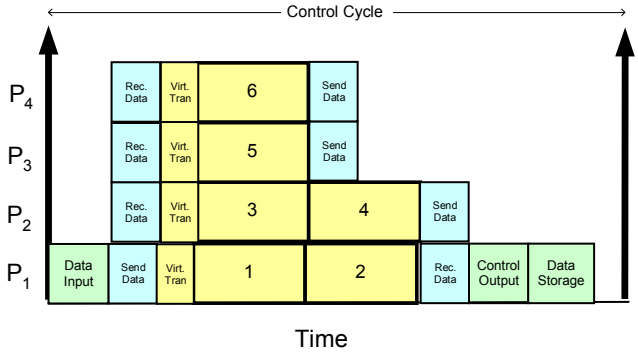

Figure 14: Straightforward task scheduling for P=4, M=6

As discussed in Section 4.3, load imbalance may occur using the straightforward task scheduling approach. Thus the pipelined approach is also implemented. The pipelined task scheduling procedure is illustrated in more detail in Figure 15. In this design, each node performs the control calculations for different subsystems, based on the pipelined task mapping described in Section 4.3. Each node has the Φ_i, Ψ_i, C_i, and D_i matrices for each of the six subsystems in its local memory (i = 1, 2, 3, …, 6). At the beginning of each control cycle, the master passes the state vectors $x_i(k)$ in addition to the digitized

sensor data to the slave processors. After the control calculations are executed, the new state vectors $x_i(k+1)$ are also gathered by the master to pass to the appropriate slaves for the next cycle.

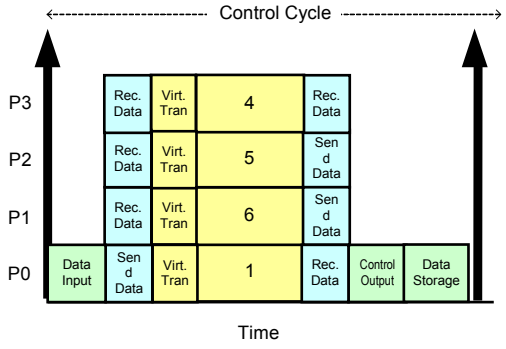

Figure 15: Pipelined task scheduling for *P*=4, *M*=6

The cycle resumes again with the nodes running a different set of controllers based on the pipelined task mapping. This approach limits the amount of time the processors are idle. There is some minute overhead due to an increase in the amount of data being passed around and the increased repetition of data input and output.

5.3. FAULT DETECTION AND RECONFIGURATION

Dynamic task mapping is achieved through establishing a protocol in the system that detects failed nodes and reconfigures the remaining processors to handle the failures. The reconfiguration of the processors when faulty nodes are detected is discussed in Section 4.4. One viable approach of detecting processor failures is through a watchdog timer. One of the processors is designated as the watchdog to the other processors and maintains a counter for each node. By the end of each control cycle, all processors must send a counter reset signal to the watchdog using an interprocessor interrupt. Failure is detected and reported by the watchdog if it does not receive the signal by that time. Upon detection of failure, the watchdog updates the status flag in shared memory for each node, indicating the working state of each processor. Flags indicate processor state and are used to reschedule tasks using the stalling technique described in Section 4.4 as necessary.

The fault detection scheme is shown in Figure 16. In the flowchart shown, processor *A* represents a master processor, which sends edge sensor data to processor *B*, a slave processor. Here the watchdog has two timers used to check the two nodes. Each node sends a hardware interrupt to the watchdog at the end of each control cycle. The interrupt restarts the timer associated with its source. A faulty node is detected if any of the two nodes fails to send the interrupt signal within the specified time limit. A flag is updated in shared memory for identification purposes each time a node sends the interrupt signal.

The critical task of data I/O and distribution is handled by a single master processor. Thus, the watchdog identifies which of the processors has failed and performs the proper reconfiguration scheme. For example, should the failed processor be the master, another functional processor from a prescribed list would take over the role of the master processor. Task rescheduling is then managed by the watchdog as specified earlier. Such a hand-over process is beyond the intended scope of this paper.

Note that the watchdog is synchronized with the functional nodes via shared memory. The scheme shown in Figure 16 can be further generalized for any arbitrarily large number of nodes with very few minor modifications. The robustness of the watchdog processor is a topic of further studies.

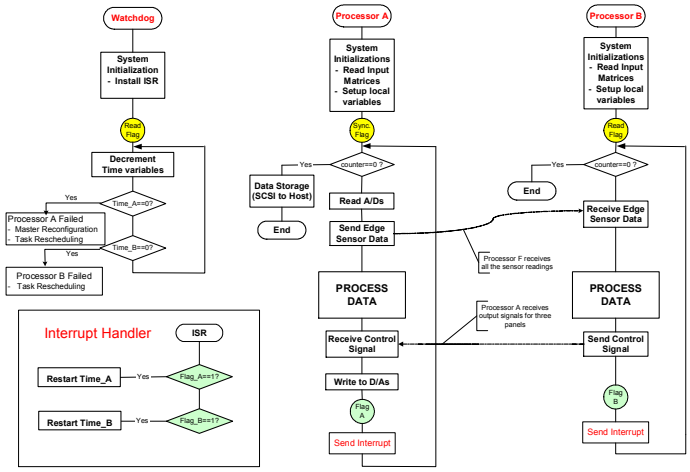

Figure 16: Fault detection scheme

5.4. EXPERIMENTAL RESULTS

The straightforward parallel implementation of decentralized controllers has been realized successfully in the SPACE testbed. These decentralized control algorithm codes were written in C and implemented using up to four DSPs running in parallel. An estimate of the standard and pipelined performance is shown in Figure 17.

The speedup curves demonstrate the effectiveness of the parallel process. The processing time is reduced as the number of processors is increased, thus allowing the attainment of real-time control objectives. However, due to the coarse grain nature of the decentralized controller tasks, there is no difference in speed for the cases $P = 3$ and $P = 4$ using the straightforward approach. In the case of four processors, P_3 and P_4 are idle during half of the control cycle as illustrated in Figure 5.

In the cases $P = 1$ to 3, the straightforward approach yields better speed-up results than pipelining due to the overhead incurred under pipelining, which is caused by message passing and the increased repetition of data input and output. However, the pipelined task mapping technique shows superiority in performance for the cases $P = 4$ and 5 due to increased throughput. The results show that the pipelined scheduling scheme, on one hand featuring fault tolerance, also resolves the problem of load unbalancing of the straightforward approach, leading to a more linear throughput as the number of processors increases. Note that the results of speed-ups of five processors are based on our estimation to show the effectiveness of the generic approach of pipelined scheduling, since there are only four physical processors in our system.

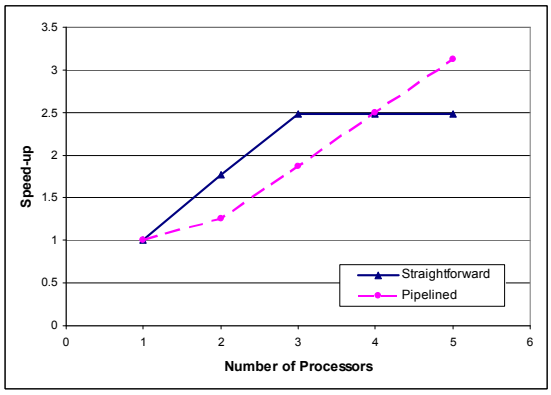

Figure 17: Speedup of straightforward and pipelined task mapping

6. CONCLUSION AND FUTURE WORK

Parallel program design and realization has been implemented successfully using decentralized control algorithms. The implemented decentralized controllers have the following features: achieve desired system performance, allow the use of small memory space, reduce computational complexity, and simplify the development of parallel programs. Real-time control performance has been achieved using a straightforward parallel program design. However, such a standard approach suffers from poor load balancing and is not resilient to processor failures. By capitalizing on the natural parallel structure of decentralized control, a fault-tolerant pipelined parallel processing design is being developed. This approach features improved load balancing for any number of processors and tasks. Pipelined task mapping seeks to improve the performance in the cases when M is not an integer multiple of P. Additionally the system allows recovery from one or more processor failures.

The potential starvation problem under pipelined task mapping is currently being investigated. In every six sample periods, each subsystem will be controlled in four periods and will go uncontrolled in the remaining two. Proper functionality remains uncompromised because the controllers view waiting subsystems as having longer sampling periods. However, questions arise concerning overall performance. Possible modifications to pipelining include assigning multiple subsystems to each processor per sample period, and shifting the pipeline window by more than one task at a time. These options will be compared tested against the original scheme. Future work also involves studying special cases of faults, such as handling master processor failure, as well as other fault detection techniques.

7. ACKNOWLEDGMENTS

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.

8. REFERENCES

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

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

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

[4] Boussalis, H., 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, February, 1997.

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

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

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

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