

**ANALYSIS OF
COMPUTATIONAL
PARALLELISM WITH
A CONCURRENT
HIERARCHICAL
ROBOT CONTROL
SYSTEM**

**John L. Michaloski
Thomas E. Wheatley
Ronald Lumia**

**U.S. DEPARTMENT OF COMMERCE
National Institute of Standards
and Technology
Robot Systems Division
Intelligent Controls Group
Bldg. 220 Rm. B124
Gaithersburg, MD 20899**

**U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
Lee Mercer, Deputy Under Secretary
for Technology
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director**

ANALYSIS OF COMPUTATIONAL PARALLELISM WITH A CONCURRENT HIERARCHICAL ROBOT CONTROL SYSTEM

**John L. Michaloski
Thomas E. Wheatley
Ronald Lumia**

**U.S. DEPARTMENT OF COMMERCE
National Institute of Standards
and Technology
Robot Systems Division
Intelligent Controls Group
Bldg. 220 Rm. B124
Gaithersburg, MD 20899**

March 1990



**U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
Lee Mercer, Deputy Under Secretary
for Technology
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director**

Analysis of Computational Parallelism with a Concurrent Hierarchical Robot Control System

John L. Michaloski, Thomas E. Wheatley and Ronald Lumia

National Institute of Standards and Technology, Gaithersburg, MD 20899, U.S.A.

November 1989

ABSTRACT

Robot control systems require concurrent architectures to overcome real-time constraints. Satisfying real-time constraints is important, but determinism, not performance, is the key element in assessing a control model. The goal of this paper is to present the concurrent hierarchical control model as an efficient, flexible, and deterministic robot control methodology and then systematically analyze the factors affecting the design and performance of a concurrent hierarchical control system. The ability for concurrent hierarchical control to be systematically modeled is derived from the generic software approach to all levels of control. The generic software control primitive of the model is the virtual control loop. A virtual control loop is a software concept analogous to a hardware duty cycle; software cyclically samples commands and status to produce outputs within a bounded response time.

Virtual control with bounded response times forms the generic software component of each level in the concurrent robot control hierarchy. Mathematical formalism can be applied to concurrent hierarchical control yielding a timing model that includes metrics of system response time and level response time. The results of the timing analysis can be applied to process management in a robot control system to yield a breakdown of control into concurrent processes: a short-term executor ensuring cyclic response and a long-term planner anticipating the future. A software control algorithm will be presented that combines concurrent executors and planners to form the generic level in the robot control hierarchy.

Key Words and Phases : communication, hierarchical control, real-time, response time, task decomposition, virtual control

1. Introduction

Multiprocessor systems, especially those built of relatively low-cost microprocessors, offer a cost-effective solution to the performance needs of an intelligent robot controller. Multiprocessor systems are beneficial for many reasons, including cost performance, modular growth, reliability through replication, and flexibility for testing alternative control strategies via different partitioning [8,18]. The effectiveness of a parallel implementation depends on the inherent parallelism of the algorithms and the cost of interprocessor communication. Algorithms that are purely sequential will not run faster on a parallel machine. Further, the cost of concurrent communication is a larger factor than it would be on a sequential machine using procedure calls. Interprocessor communication using shared memory or message passing takes longer because of the overhead of protocols and extra synchronization. The cost of communications must not negate the time savings obtained by parallel execution on the different processors. Thus, to achieve the benefits of parallelism, a robot control model must be inherently concurrent and have an efficient means of communication.

The authors current address is the National Institute of Standards and Technology, Robotics Systems Division, Bldg. 220 Rm. B127, Gaithersburg, MD. 20899 This article was prepared by United States Government employees as a part of their official duties and is therefore a work of the U.S. Government and not subject to copyright. This article references certain commercial equipment, instruments or materials and such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology nor does it imply that the materials or equipment identified are necessary the best available for the purpose.

Parallel machines cover a broad range of computer architectures but several characteristics delineate purpose. Parallel architectures vary within a fine-grain to coarse-grain domain [13, 15, 16]. General-purpose coarse-grain machines are a parallel computer architecture maximizing batch throughput. Here, the user may be unaware of any parallelism involved. Supercomputers are coarse-grained machines that specialize in fine-grained number crunching in parallel for mathematically intensive algorithms. In this case, the programmer may add some data flow enhancements to assist the number crunching. Other fine-grain computers specialize in data-intensive operations that do not handle general-computing efficiently. The variety in machines leads to the distinction between systems that maximize the throughput of many jobs, known as *throughput-oriented multiprocessors*, and systems that maximize the execution of one process, known as *speedup-oriented multiprocessors* [5]. A robot controller will be characterized as a speedup-oriented multiprocessing application since the controller is partitioned into a set of concurrent, cooperating processes.

This paper describes the applicability of the concurrent hierarchical model as a multiprocessor architecture for a robot control system. The subsequent sections of the paper are organized as follows: the second section discusses the concurrent hierarchical control model as an architecture for a multi-computer robot control system. The theory of concurrent hierarchical control model will be presented and include a discussion on the importance of task decomposition and real-time response time. A software algorithm detailing the concurrent hierarchical model that readily ports to a multi-computer system will be included. The third section presents a timing analysis of a control system based on the concurrent hierarchical model. The fourth section discusses an implementation of a concurrent, hierarchical robot control system at the National Institute of Standards and Technology (NIST, formerly the National Bureau of Standards) as applied to the NASA flight telerobotic servicer and contrasts this model to other robot control models.

2. Concurrent Hierarchical Real-Time Control

A goal of an efficient multiprocessor system is to exploit the benefits of parallelism while reducing software complexity. Hierarchical structure is a well-defined design technique often applied to concurrent robot control [1, 4, 5, 18, 21]. Hierarchical structuring of a control system offers an easy and systematic parallel approach. In itself, hierarchical structure is not sufficient to define concurrency. Task decomposition and response time further refine the concurrent hierarchical model to handle the processor management of a hierarchical system, the complexity of any level, and communication protocols.

Task decomposition is defined as the process of recursively partitioning a task into smaller, more manageable subtasks. Task decomposition is complete when the subtasks are atomic. In this paper, a subtask partition defines a level in the hierarchy. At each level in the hierarchical breakdown, an interface exists through which the adjoining levels exchange information. As applied to control processes, one should not confuse hierarchical decomposition with a deeply-nested serial process that follows a single thread of control flow. Although levels may share some data that models the world, all levels run concurrently, and each can be defined as a virtual control loop.

Virtual control, as applied to hierarchical task decomposition, is a software control strategy that consists of the following periodic actions: 1) inputting status and commands from adjoining levels, 2) producing a goal-directed output based on the status and command, and 3) outputting an action to the adjoining lower level and a status to the adjoining upper level. One iteration through the virtual control loop defines a *cycle*. When executing, each virtual control layer in the hierarchy can be considered part of a long chain defining the hierarchical state, yet each layer independently formulates its control flow. This leads to the definition of *concurrent hierarchical control* as communicating layers of virtual control loops. Figure 1 illustrates the information flow in the concurrent hierarchical

model.

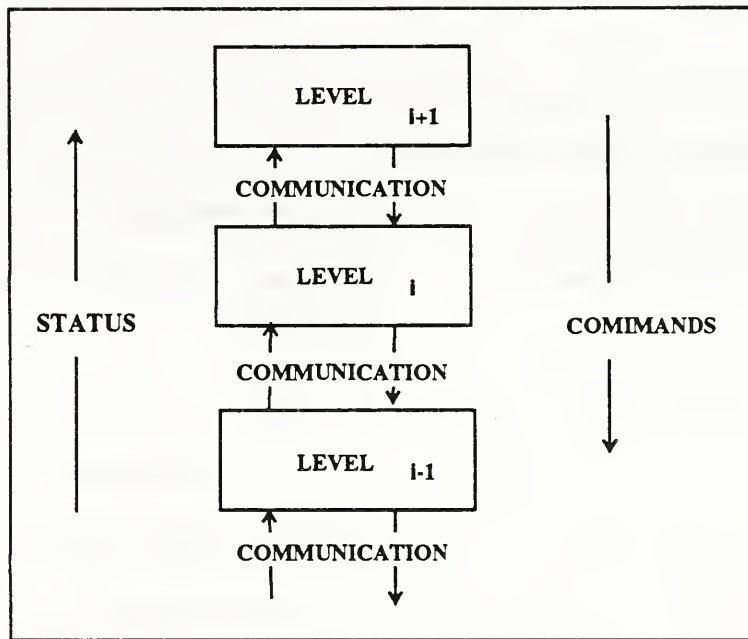


Figure 1. Information Flow in a Concurrent Hierarchical Control System

A concurrent task decomposition model does not complete a definition of the control system. Mere functional correctness is not sufficient. Of paramount importance is measuring the system performance by the time delay required to respond (i.e., calculate a solution.) A real-time system must satisfy physical time constraints. Adding timing constraints changes the mechanism of task decomposition. Now, the number of operations performed depends heavily on the time required to make a decision and has a direct effect on decomposition. Responding to an event too late nullifies the control no matter how intelligent the subsequent action. This timing restriction leads to the standard definition of *response time* as the maximum allowable duration between an event and an action resulting from the occurrence of that event.

Concurrency between levels does not guarantee a system response time. Concurrency within a level is necessary to ensure predictable system behavior. To achieve predictable real-time control, each level in the hierarchy must periodically sample inputs and produce outputs. Each level cannot run independently of adjoining levels. Should one level be unable to complete the virtual control loop within one cycle, system performance is no longer predictable. For example, a level awaiting a reading from a failed sensor could loop forever. Concurrency within a level can be achieved by dividing a control level to account for present and future actions. The *planner* is responsible for generating a plan consisting of a series of actions. The planner selects the best plan from a candidate list of alternative plans that achieve the commanded goal, given the current state of the environment. An *executor* enables state transitions and so is responsible for stepping through a generated plan. The executor must run periodically to guarantee the cyclic behavior of the virtual control loop. The planner and executor can run concurrently within a level, thus insuring a predictable level response which implies a predictable system response.

The following algorithm sketches an overview of planning and executing within a level of the hierarchical concurrency model. The algorithm is written in pseudo code using Brinch Hansen primitives *cobegin* and *coend* for concurrent execution [10]. It is important to note that the concurrent sections of code could be either tightly- or loosely-coupled depending on timing constraints. (Processes that execute in parallel on separate processors are defined as

loosely-coupled. Processes that execute in a multi-tasked operating system on a single processor are defined as *tightly-coupled*.) In this algorithm, the familiar file operations lock and unlock are primitives that guarantee mutual exclusion. Comments in the following algorithm will be delimited by double quotes.

```

procedure level()

cobegin
  repeat "executor section, runs at a higher priority"
    wait_until( next-cycle);           "timing synchronization primitive "
    update(next-cycle);               "update next cycle count"
    lock; read_command; unlock;      "read command "
    lock; read_status; unlock ;      " read status "
    read plan;                        "read current plan"
    process;                           "execute level "
    lock; write_command; unlock;     "write command "
    lock; write_status; unlock;     "write status "
  until forever;

  repeat "planner"
    plan;                               "do level planning"
    lock; write_plan; unlock;       "update executor plan"
  until forever;

coend

```

The concurrency of the planner and executor allows this software algorithm to be easily ported onto a parallel pipelined computer architecture. Each level in the hierarchy runs as a concurrent process. Parallelism is not a direct result of concurrent hierarchical control because a lower level cannot process without a command from its adjoining higher level. However, once a task is underway, this creates a pipeline in which each level is executing in parallel. For example, it would take 5 control cycles for information from the top level to filter down to the bottom level in a five layer hierarchy. However, this cyclic protocol will guarantee a deterministic system response time.

3. Timing Analysis

Systems that meet response time obligations are defined as real-time systems. The concept of response time or hard real-time must be contrasted to soft time used as a sequencer of events. For example, a sequence of robot commands *GOTO A*, *GOTO B*, does not contain any timing information. Within the decomposition of this command, only the lower levels are concerned about timing so that constant updates to the robot are guaranteed. However, the command *GOTO A BY t_1* *GOTO B BY t_{i+1}* where t_1 and t_{i+1} are some explicit time values, demands real time handling. Now, a solution must be supplied within an explicit time. This embodies the distinction between time as a dynamic real-world parameter versus time as a sequencing tool.

For a hierarchical control system to be real time, it must meet the demands of 1) the response time of the system, plus 2) the response time of each level in the hierarchy. Response time will be assumed to be an input of the specification and therefore, a constraint of, rather than the output of, some timing analysis.

3.1 System Response Time

System response time can be considered in two ways. One perspective is to measure the elapsed time a system takes to respond to a specific discrete event. Another view is the time necessary to provide a complete solution. In a

computer program, consider the distinction between the response to a keyboard terminating interrupt as opposed to the time elapsed until the program has completed. Programs that fail to respond to a program interruption are flawed. In this paper, system response will be defined as the amount of time to achieve an immediate response.

System response time can be calculated as the sum of the worst case time responses as commands and status filter up or down the concurrent hierarchy. The movement of command and status can be characterized as follows. Let R_i be the response time for level i . If n denotes the number of levels employed in the hierarchical system and the maximum response time of a control cycle of any of the levels is R_{max} , then if the i th level spends t_c^i seconds communicating/waiting for commands and status, and t_{proc}^i processing, this leads to the following minimum timing constraint for each cycle:

$$((t_{proc}^i + t_c^i) = R_i) \leq R_{max} : \text{for each } i=1 \dots n. \quad (3.1)$$

An upper bound to the system response time R_s can be calculated as follows:

$$R_s \leq n R_{max} . \quad (3.2)$$

Thus R_s/n defines R_{max} the maximum response time of any level R_{max} . No level can exceed this time limit and still guarantee proper system communication flow. This implies a synchronization signal in each level to periodically communicate (sample command and status) so that updated information can be transferred between levels. This periodic communication prevents any level from processing in isolation and skewing the system response time.

The rate at which each level samples new commands and statuses is dependent on its position within the hierarchy. Assume the machine update rate of the lowest level tied to the robot is 1 millisecond per cycle. Imposing this rate as R_{max} for each level may be too constraining. A 1 millisecond cycle precludes the use of context switching. All higher levels may want to sample using some other less-restrictive, yet bounded, time period. This leads to a definition of R_s , system response time, as the sum of R_i response times tied to the machine update rate, and the sum of R_j response times arbitrarily set to meet the system response time goal.

$$R_s = \sum R_i + \sum R_j \quad \text{for } i=1 \dots m \text{ and } j= m+1, n \quad (3.3)$$

$$\text{where } \forall (R_k) \leq R_{max} \quad \text{for } k=1, \dots, n .$$

As a design metric, the response time of a system dictates the amount of system processing power. A system that requires a response in 10 minutes has much different requirements than a system limited by 1 second response times. However, system response time does not directly correlate to performance at lower levels. An overall 10 minute system response time may still require millisecond updates at the lowest level to achieve real-time control. In effect, system response time gives a general metric to evaluate timing.

3.2 Low-Level Timing Requirements

The required system response time restricts the maximum response time imposed on each level. Meeting this timing restriction may not allow some levels enough processing time for sophisticated control. This leads to the motivation to divide task decomposition into planners and executors. Now, each level can maintain real-time control,

yet concurrently evaluate alternative future actions. An initial assumption is that t_{exec}^i , the time the executor runs each cycle, is less than the level response time limit maximum; otherwise the system response time must be increased or a finer resolution of task decomposition must be attempted.

$$t_{\text{exec}}^i \leq R_{\text{max}} \quad (3.4)$$

Response time for any level i , previously defined as R_i , depends on the timing constraints of the level in the hierarchy. R_i is composed of t_{exec} , the time each level spends running the executor, t_{comm} , the time spent communicating/waiting for data, and t_{residual} , the time remaining before the next cycle. This leads to the following definition for R_i response time for the i th level.

$$R_i = t_{\text{exec}}^i + t_{\text{comm}}^i + t_{\text{residual}}^i \quad (3.5)$$

Equation 3.5 leads to some observations. If the response time of the level is very small, say 1 millisecond, this leaves very little time to do both processing and communicating. Finally, t_{residual}^i the amount of residual time per level dictates whether the executor, the planner and other processes can run concurrently as tightly-coupled processes on one processor or must run loosely-coupled on different processors. Each of these observations will be explored further.

At the lower levels of the system, performance must be fast and predictable. An example is the servo loop at the motor level. These operations must operate with short response times and guaranteed update rates. For example, in order for a robot to exhibit smooth motion, control updates to the arm must be supplied within a set time linked to the physical hardware capabilities. A control cycle executing sufficiently fast will provide motion control that will appear continuous and thus real-time. Smooth motion results from an efficient input-compute-output cycle that maintains a small standard deviation each cycle with an upper bound σ on variance. Let us define the residual time t_{residual}^i as the time remaining in a cycle after the executor is finished. This implies that not only must t_{residual}^i be greater than zero, but must be bounded by a sum of the worst case times for both processing and communication. This leads to the following constraint.

$$0 < t_{\text{residual}}^i + \sigma \leq R_i - t_{\text{exec}}^{i,\text{max}} - t_{\text{comm}}^{i,\text{max}} \quad (3.6)$$

$$\text{where } t_{\text{exec}}^{i,\text{max}} = \max(t_{\text{exec}}^{i,j}) \quad j=1, \dots, \infty \text{ cycles}$$

$$t_{\text{comm}}^{i,\text{max}} = \max(t_{\text{comm}}^{i,j}) \quad j=1, \dots, \infty \text{ cycles}$$

This relation sets a limit on the t_{residual}^i time available to other background processes each cycle. This further implies that adding further capabilities with a small t_{residual}^i residual time will require additional processors, which will adversely affect t_{comm}^i , the time for communications.

In practice, response times at the lowest levels are very fast with minimal time devoted to communication and planning. There are numerous examples that illustrate this principle with efficient, multiprocessor designs at low

levels of the hierarchy [8, 14, 19, 20, 22].

3.3 Higher-Level Timing Requirements

Moving up in the hierarchy, timing constraints are relaxed and leads to an increase in R_i , level response time. The increase in R_i means that the communications and executor components of a virtual control cycle are a smaller percentage of the levels processing activity. Increased time at higher levels allows slower options to execute in the background, such as secondary data storage, multi-tasking, and character input and output for a user interface. In fact, loosely-coupled machines could conceivably link together in a distributed control system at the very highest levels for systems with a moderating response time.

In the middle of the hierarchy the concern for processor power remains. A planner and executor tightly-coupled on the same processor is desirable for its simplicity and the ease on communication requirements. The decision to partition a control level (i.e. executor and planning processes) as tightly-coupled on one processor or loosely-coupled on parallel processors depends on the constraints of timing and available computing resources.

In order to analyze the relationship between the planner and executor in an tightly-coupled environment, several assumptions need to be established. First, levels execute every cycle and these cycles are of fixed time c . This leads to the following equality of time cycles throughout the system.

$$t_{i+1} = t_i + c \quad i=0, \dots \infty \text{ cycles} \quad (3.7)$$

Second, once the executor has completed one cycle, it must wait for the next clock cycle before processing again. At higher levels, sufficient residual time allows for the executor to block and wait until the next time cycle before processing. For lower levels, the executor could busy/wait and poll on a system clock awaiting the next clock cycle in cases where the time for a context switch between tasks may be too large a percentage of cycle operation.

Third, the concept of blocked, running, waiting processes implies the use of multitasking. In order to achieve real-time performance, the multi-tasking scheduling need not be fair, as in a general-purpose operating system. Instead, the multitasking must allow assignment of priorities to tasks. The requirement for priorities leads to the assumption that the executor is of higher priority than the planner. The higher priority of the executor preserves the system response requirement.

Fourth, communication between the planner and the executor depends on non-interruptible critical sections. This is important since some multi-tasking systems lack a non-preemptive feature.

Given these assumptions for tightly-coupled execution, further timing constraints include context switching overhead, critical section support, and general planner throughput requirements. Whether tight-coupled execution is advisable remains to be determined. First, the planning phase of each cycle provided only a small $t_{residual}^i$, residual time processing and would preclude a planner from formulating a plan in a reasonable amount of time. The requirement that the planner must finish any plan within d cycles should be imposed on a level. This assumption leads to the constraint that a plan take d^i cycles to finish for the i th level,

$$t_{plan}^i / d^i \leq t_{residual}^i - 2 t_{cs} \quad (3.8)$$

where each cycle using $t_{residual}$ minus two times the t_{cs} a context switching amount of processing time.

If this condition cannot be held, then a multiprocessor planner and executor level should be used. Assuming the condition to finish a plan within d^i cycles does hold, more detailed constraints need resolving. First, response time needs reevaluation. Therefore, including the planner on the same processor adds two extra context switches to a cycle, represented in time t_{cs} , plus increases the amount of communication time to include both interlevel communication $t_{il-comm}^i$ and communication between the planner and executor $t_{e.p-comm}^i$. This has the corresponding effect on response time:

$$R_i = t_{exec}^i + t_{comm}^i + t_{plan}^i + t_{residual}^i + 2 t_{cs} \quad (3.9)$$

$$\text{where } t_{comm}^i = t_{il.comm}^i + t_{e.p-comm}^i$$

Since the planner and executor modules share data, whenever the planner must update the plan graph, it enters a critical section whereby it cannot be preempted. This update may occur every cycle but it must be accounted for in the worst case scenario. A problem may arise for plan updating of a large amount of data because the amount of time allotted for the critical section must only be a fractional amount of the time used by the executor each cycle. This leads to the constraint that the planner to executor communication is small.

$$t_{e.p-comm}^i \ll R_i \quad (3.10)$$

If the planners cannot meet this constraint, a parallel planner and executor should be used and interprocessor communication between the processes should use a scheme that incorporates time slices or double-buffers for plan update. Satisfying this plan updating constraint heeds one cautionary note. Should a critical section in one cycle overlap into the next cycle, the executor would wait until the planner completes updating a portion of the plan graph before continuing processing. Thus, mutual exclusion requires a non-preemptable multitasking scheduler to prevent preemption of the lower priority planner in its critical section. Summarizing, the following features must be available for an tightly-coupled planner/executor.

- high priority processes (i.e., executors) run every cycle at a fixed interval responding to current control requirements, and
- lower priority processes (i.e., planners) run in the background anticipating future control requirements.
- low priority processes allow critical sections to run to completion (i.e., non-preemptive) but must be only a fractional time portion of higher-priority processes operating cycle

3.4 Algorithm Timing Analysis

The timing constraints of the control level algorithm outlined in the previous section can be studied using a Gantt chart notation. For the sake of accounting, the algorithm will be divided into functions. These functions will be assigned arbitrary timing constraints for the purposes of analysis. R will represent the function for an interprocessor read of a command and status and will take 1 ms. W will represent the function for an interprocessor write of a command and status, and will take 1 ms. X will represent the current level executor processing and will range from 1 to 3 ms computation intervals. P will represent the planner function and will require 1 to 6 ms before updating a plan. Plans must be updated within 16 ms or at worst an updated plan should be available every third cycle. U will represent

the critical section where the planner updates the current plan graph. For simplicity, operating system computation costs such as context switches will be ignored. Initially, a response time of 8 ms will be considered. For this case, all the functions can run tightly-coupled on one processor in one cycle, with sufficient residual processing time as shown by the Gantt chart in figure 2.

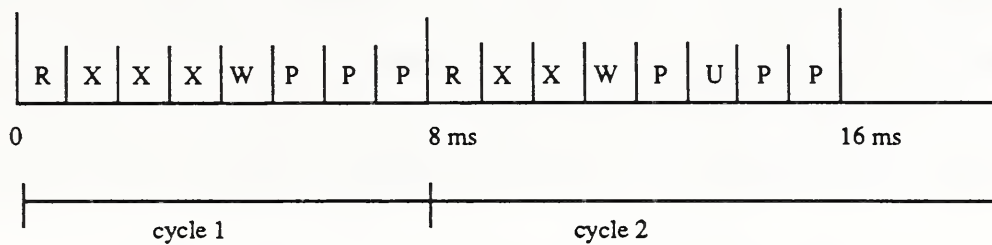


Figure 2. Control Level Computation of Single Processor Concurrent Execution

Notice that the algorithm repeats ad infinitum the basic "RXWPU" computational pattern. The planner runs at least 3 ms. each cycle which implies that after two cycles 6 ms. will have been allotted to the planner with 1 ms. for plan updating. More stringent timing constraints may degrade system performance beyond an acceptable level. For example, a 6 ms. response time allows insufficient time for planning, so that within 3 cycles no updated plan exists. Figure 3 shows the Gantt chart for this situation.

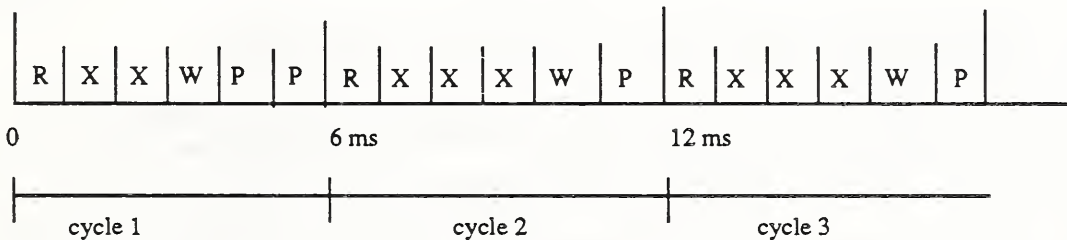


Figure 3. Control Level Failing to Meet Planning Time Constraints

The lack of computational resources for planning suggests the partitioning of the problem across processors. Now, processor one (*P1*) will have exclusive execution, but will have the additional timing constraint of interprocessor communication with the planner. The function *C* will represent the interprocessor communication overhead. *C* requires 1 ms. With excess computational power, the processor must now idle between cycles represented by a dash (i.e., -). The executor repeats ad infinitum a "CRXW-" computational pattern. Processor two (*P2*) will execute planning exclusively. Computational resources easily guarantee 6 ms. planning to generate updated plans with 12 ms. As a side effect of excess computational power, the planner may have to wait for fresh information from *R*, an executor status read. (Assume for the discussion that shared *R* status information is in a critical section and read-only for the planner.) The Gantt chart in figure 4 shows the execution scheme in parallel processors.

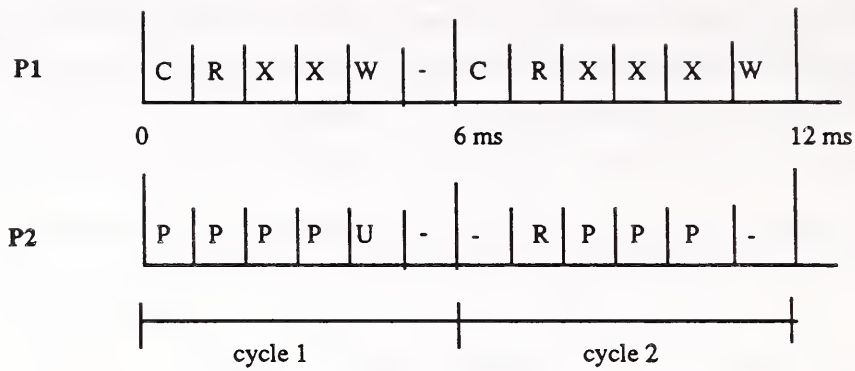


Figure 4. Control Level Computational of Dual-Processor Execution

In the parallel design, the control level effectively meets all timing constraints with some computational waste. However, this solution can be sensitive to increases in complexity of the executor (X) portion of the control cycle. For example, if the function X is replaced by a newer version which requires 4 ms. a cycle, then the above functional allocation across processors will not work. For in the worst case, there is no time for interprocessor communication with the planner (C). In this case, faster hardware or a more refined task decomposition strategy must be used.

4. Application of Concurrent Hierarchical Control and Comparisons to Other Models

The concurrent hierarchical model of a robot control system has been previously implemented with a purely executor style of task decomposition for a robot control system at the National Institute of Standards and Technology [3]. The flow of control featured state-table transitions. Where planning was appropriate, static plan definitions were used. The system offered several benefits. First, the system was sensory-interactive and adapted to perturbations in the environment in real-time even though the world model was limited to basic feature recognition. Second, interface standards were proposed as a consequence of the well-defined interfaces of hierarchical decomposition [6]. Third, the communicating sequential levels aided information-hiding and insulated software on one level from changes on another level. Fourth, high-frequency feedback from the periodic sampling of commands and status ensured predictability necessary for a reliable and safe robot controller.

The NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) is the basis for the current robot control system under development [2]. The concurrent hierarchy decomposes into these levels: task, elemental move, primitive, and servo. NASREM extends the concurrent hierarchical model to examine such research issues as task planning, extensive world modeling including maps, object definitions and feature recognition. To maintain cost, flexibility and portability, multiprocessors communicating across a shared backplane have been chosen for the hardware. Concurrency is supported as both loosely-coupled on multiprocessors and tightly-coupled on a single processor. Software development is based on a real-time, cross-target ADA environment extended to meet real-time scheduling priorities.

The real-time system consists of a series of Motorola 68020 processor boards linked with the VME bus. Each processor board of the target system contains a small, fast, multitasking, real-time executive that guarantees predictable execution times. For the real-time executive, task-switching is approximately 100 microseconds. A system clock synchronizes software execution on all processor boards. Layered on top of this real-time kernel is a shared memory reader/writer scheme used for inter-level concurrent communication. The host-target operator interface

communication is linked using a VME bus to VME bus adapter. The hosts provide non-real-time services, such as programming development, user-interface, or graphic diagnostic display.

The differing response times between levels affects granularity. At the lower primitive level [2], the planner is responsible for generating a series of trajectory points in the background. In the foreground, the executor runs at 50Hz and supplies new joint positions to the servo level at each clock cycle. At the servo level, the robot receives torque commands at updates of 400 Hz. Wheatley offers a more detailed examination of the timing constraints at the lower levels [24]. At higher levels, completion time for plans is less critical. Multitasking on a single processor supplies enough processing power for both the planner and executor. Figure 5 provides a summary of the implementation design for planning and execution within the concurrent hierarchical model based on timing constraints at NIST.

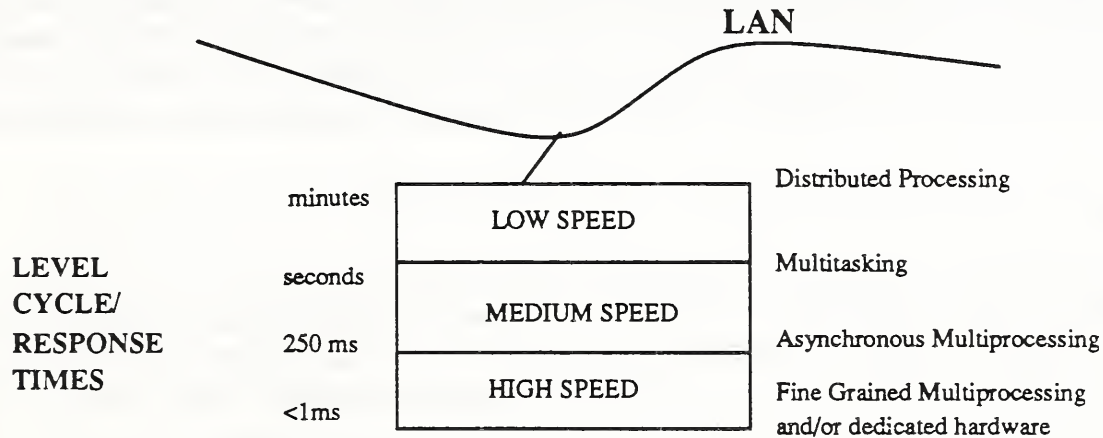


Figure 5. Concurrent Resource Requirements Based on Cycle Response Time

4.1 Comparison of Concurrent Hierarchical Control to Other Models

A survey of existing robot control systems provides numerous concurrent models. The requirement to satisfy real-time constraints has led to numerous implementations using robot control architectures with several processor boards on a common bus [8, 12, 13, 19, 20, 22]. These architectures share common characteristics with the concurrent, hierarchical control model. Although few of these control systems are directly labeled as a hierarchical, most systems have at least a two-level hierarchy, with a low-level, real-time subsystem handling real-time motion control, and a slower high-level subsystem responsible for planning. For example, the following research models use the two-level hierarchy. The CMU CHIMERA model partitions concurrency as a two layer model, low-level real-time motion control connected to a non-real-time AML/X command environment [2]. The MIT CONDOR project has a real-time system consisting of a series of Motorola 68020 based boards connected by VME bus to VME bus adapter to a SUN-3 workstation controlling the Utah/MIT hand [19]. The University of Pennsylvania has a robot coordinator issue force and motion commands to a real-time robot force and motion server [20]. Brown University uses a series of high-speed networks connecting a real-time servo systems and general computing resources [12]. IBM partitions concurrency into real-time and non-real-time by connecting the real-time system to the programming system by a real-time bridge [13]. IBM extends the exchange of commands and status to include varying frequencies; such as every cycle, every nth invocation or asynchronously.

The RCA-McGill RCCL model uses a 3 level concurrency hierarchy that supports a planning level, a trajectory level, and a joint-control level. The joint control level operates at 1 kHz and assigns 1 processor per joint. The fine motion trajectory level runs a 50 Hz and runs as high-priority cyclic task. Concurrently, a rough motion planner asynchronously produces a queue of motion requests for the trajectory level as a background task. In RCCL, the

trajectory level acts like an executor.

The HIC model is an operating system based on a hierarchy of servo loops for controlling the Utah/MIT hand [4]. This model emphasizes layers of servo loops with deterministic scheduling as the main control process and uses periodic data buffers for communication. The periodic data sampling of the model supports the deterministic flavor of concurrent hierarchical processing model.

Implicitly, these real-time models may have additional levels within the hierarchy, but further layering does not directly correspond to a virtual control loop implementation. This is due in part to the concept of one return status per procedure call of serial execution. For example, a Cartesian trajectory plan in a sequential robot language such as AML/X must be decomposed into a series of kinematic poses for the robot. This process is implicitly hierarchical. These implicit levels could be partitioned into separate concurrent processes that communicate through established interfaces rather than serial routines that communicate via subroutine calls. Further, with concurrent operation, the concept of sampling the return status, not just receiving a final status, embodies the feedback nature of a virtual control loop.

5. Conclusion

Concurrent hierarchical structuring is a model that not only satisfies robot control timing constraints, but provides predictable system performance. Hierarchical structuring uses task decomposition to create the level of granularity necessary to meet the strict real-world timing requirement at the lowest level while offering a convenient software development tool. A hierarchical control system defined with task decomposition offers information hiding between levels that can lead to well-defined, and eventually standardized interfaces. The basis of concurrency within the hierarchical model is the virtual control loop. The regulated and predictable behavior of the virtual control loop leads to timing metrics guaranteeing performance. Predictable response time, especially important as complexity increases, is a key for more reliable and safe robot controls systems.

References

- [1] J.S. Albus, A.J. Barbera and R.N. Nagel, "Theory and Practice of Hierarchical Control," *Twenty-third IEEE Computer Society International Conference*, pp. 18-39 1981.
- [2] J.S. Albus, H.G. McCain, and R. Lumia, *NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM)*, NBS Technical Note 1235, National Bureau of Standards, Gaithersburg, Md., July 1987.
- [3] A.J. Barbera, M.L. Fitzgerald, and J.S. Albus, "Concepts for a Real-Time Sensory-Interactive Control System Architecture," *Proceedings of the 14th Southeastern Symposium on System Theory*, April 1982.
- [4] D. Clark, "HIC: An Operating System for Hierarchies of Servo Loops," In *Proceedings of the IEEE International Conference on Robotics and Automation* (Scottsdale, Arizona.), IEEE, New York, pp. 1004-1009, May 1989
- [5] V. Dupourque, H. Guiot, and O. Ishacian, "Towards Multi-processor and Multi-robot Controllers," In *Proceedings of the IEEE International Conference on Robotics and Automation* (San Francisco, Cal.), IEEE, New York, pp. 864-868, April 1986.

- [6] M. Dubois, C. Scheurich, and F.A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *IEEE Computer*, pp. 9-21, February 1988.
- [7] M.L. Fitzgerald, A.J. Barbera, and J.S. Albus, "Real-Time Control Systems for Robots," *SPI National Plastics Exposition Conference*, 1985.
- [8] D. Gauthier, P. Freedman, G. Carayannis, and A.S. Malowany, "Interprocess Communication for Distributed Robotics." *IEEE Journal of Robotics and Automation*, Vol. RA03, No. 6, pp. 493-504, Dec. 1987.
- [9] R.D. Gaglianella, and H.P. Katseff, "A Distributed Computing Environment for Robotics," In *Proceedings of the IEEE International Conference on Robotics and Automation* (San Francisco, Cal.), IEEE, New York, pp. 1890-1895, April 1986.
- [10] P.B. Hansen, *Operating System Principles*, Prentice-Hall, Englewood Cliffs, N.J., 1973.
- [11] C.A.R. Hoare, "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21, No. 8, pp. 666-677, August 1978.
- [12] P. Kazanides, H. Wasti, and W.A. Wolovich, "A Multiprocessor System for Real-Time Robotic Control: Design and Applications," In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, New York, 1987, pp. 1903-1908
- [13] J.U. Korein, G.E. Maier, R.H. Taylor, and L.F. Durfee, "A Configurable System for Automation Programming and Control," In *Proceedings of the IEEE International Conference on Robotics and Automation* (San Francisco, Cal.), IEEE, New York, pp. 1871-1877, April 1986.
- [14] C.P. Kruskal and C.H. Smith, "On the Notion of Granularity", National Bureau of Standards Report, July 1987.
- [15] J. Lloyd, M. Parker, and R. McClain, "Extending the RCCL Programming Environment to Multiple Robots and Processors," In *Proceedings of the IEEE International Conference on Robotics and Automation* (Philadelphia, Pa.), IEEE, New York, pp. 465-469, April 1988.
- [16] G.E. Lyon, "On Parallel Processing Benchmarks", National Bureau of Standards Report , NBSIR 87-3580, pp. 1-23, June 1987.
- [17] G.E. Lyon, "Programming The Parallel Processor", In *Second Symposium on the Role of Language in Problem Solving*, sponsored by the Applied Physics Laboratory of Johns Hopkins University, April 2-4, 1986.
- [18] R.B. McGhee, D.E. Orin, D.R. Pugh, and M.R. Patterson, "A hierarchically-structured system for computer control of a hexapod walking machine," In *Proceedings of the 5th IFTOMM Symposium on Robots and Manipulator Systems*, (Udine, Italy), June 1984). IFTOMM, 1984.
- [19] S. Narasimhan, D.M. Siegel, and J.M. Hollerbach, "Condor: A Revised Architecture for Controlling the Utah-MIT hand," In *Proceedings of the IEEE International Conference on Robotics and Automation* (Philadelphia,

Pa.), IEEE, New York, pp. 446-449, April 1988.

- [20] R.P. Paul, and H. Zhang, "Design of a Robot Force/Motion Server," In *Proceedings of the IEEE International Conference on Robotics and Automation* (San Francisco, Cal.), IEEE, New York, pp. 1878-1883, April 1986.
- [21] D.L. Parnas, "On a 'Buzzword': Hierarchical Structure," In *Proceedings of the IFIP Congress*, North-Holland, Amsterdam, 1974.
- [22] K. Schwan, T. Biharit, B. Weide, and G. Taulbee, "High-Performance Operating System Primitives for Robotics and Real-Time Control Systems," *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 189-231, August 1987.
- [23] D. Schmitz, R. Hoffman, P. Khosla, "CHIMERA: A Real-time Programming Environment," In *Proceedings of the IEEE International Conference on Robotics and Automation* (Scottsdale, Arizona.), IEEE, New York, pp. 846-852, May 1989.
- [24] T. Wheatley, "Mapping Processes to Processors for Space Based Robot Systems," *IEEE International Conference on Systems Engineering* (Dayton, Ohio), August 24-26, 1989.

List of Figures:

Figure 1. Information Flow in a Hierarchical Control System

Figure 2. Gantt Chart Illustrating a Control Level Computational Pattern of Tightly-Coupled Concurrent Execution on One Processor

Figure 3. Gantt Chart Illustrating a Control Level Failing to Meet Timing Constraints

Figure 4. Gantt Chart Illustrating a Control Level Computational Pattern from Loosely-Coupled Parallel Execution on Two Processors

Figure 5. Summary of Concurrent Level Resource Requirements Based on Cycle Response Time

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NISTIR 90-4251	2. Performing Organ. Report No.	3. Publication Date MARCH 1990
4. TITLE AND SUBTITLE Analysis of Computational Parallelism with a Concurrent Hierarchical Robot Control System			
5. AUTHOR(S) John Michaloski, Thomas Wheatley, and Ronald Lumia			
6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions) NATIONAL BUREAU OF STANDARDS U.S. DEPARTMENT OF COMMERCE GAITHERSBURG, MD 20899		7. Contract/Grant No.	8. Type of Report & Period Covered
9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)			
10. SUPPLEMENTARY NOTES <input type="checkbox"/> Document describes a computer program; SF-185, FIPS Software Summary, is attached.			
11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here) Robot control systems typically use parallel architectures to overcome real-time constraints. Satisfying real-time constraints is important, but predictability, not performance, is the key element in assessing a control model. Concurrent hierarchical control provides a fast, predictable robot control model. Further, concurrent hierarchical control reduces software complexity; incorporating a systematic approach applicable to all levels of control. The basic control primitive of the model is the virtual control loop. A virtual control loop is a software concept analogous to a hardware duty cycle. Levels in the hierarchy run concurrently and communicate at given intervals. Predictable level response makes necessary the breakdown of control into concurrent processes; a short-term executor ensuring cyclic response and a long-term planner anticipating the future. A software control algorithm will be presented that combines concurrent executors and planners with interprocessor communication applicable to all levels in the concurrent hierarchy. Mathematical formalism can be applied to concurrent hierarchical control yielding a timing model that includes metrics of system response time and level response time. The results of the timing analysis can be applied to multiprocessor management in a robot control system.			
12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons) communication, hierarchical control, real-time, response time, task decomposition			
13. AVAILABILITY <input checked="" type="checkbox"/> Unlimited <input type="checkbox"/> For Official Distribution. Do Not Release to NTIS <input type="checkbox"/> Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. <input checked="" type="checkbox"/> Order From National Technical Information Service (NTIS), Springfield, VA. 22161		14. NO. OF PRINTED PAGES 18	15. Price A02

