Embedded Real Time Linux Kernel Design using Game Theory and Control Logic

Dipnarayan Guha and Jun Kyun Choi
Broadband Network Laboratory, Information and Communications University
103-6, Munji-Dong, Yuseong-Gu, Daejeon 305-714, Republic of Korea
{dip,jkchoi}@icu.ac.kr

Mashfique Hassan
Electronic Systems, Northrop Grumman Corporation
P.O. Box 1897, MS-1400, Baltimore, MD 21203, United States of America
mh228@cornell.edu

Abstract

This work-in-progress report describes a distributed hybrid architecture that imparts autonomous intelligence capabilities of "perception-decision-action" to an ALV (Autonomous Landing Vehicle) control unit. The system is shown to degenerate to a remote thread execution mechanism of a real-time operating Linux system where the kernel instantiation is control strategy dependent.

1 Introduction

1.1 Problem Description and Project Discussion

The problem on co-operative control of distributed Autonomous Vehicles in adversarial environments was first addressed in the consortium 2001 MURI on May 14, 2001, which included UCLA, Caltech, Cornell and MIT. Three major conceptual thrusts in the design were taken up by this study group, namely:

I. The concept of Robust Hybrid Automaton (RHA)

II. Model-based Programming of autonomous explorers

III. Game theoretic concepts

The problems for autonomous vehicle motion control were described in the light of these concepts and a continuous type tracking solution was proposed.

1.2 Problem Formulation

The basic problems for Autonomous Landing Vehicles or automated robots have always been the following:

I. Generate and execute a (sub)optimal motion plan,satisfying given boundary conditions, flight envelope and obstacle avoidance constraints,in a dynamic and uncertain environment.

II. Concept of Nonlinear Control - steering of underactuated, non-holonomic systems, Stabilization/tracking for nonlinear systems, Flight envelope protection

A number of solutions based on Robotics and Artificial Intelligence were suggested. Path planning, i.e. primarily obstacle avoidance, for non-holonomic dynamic systems was the most challenging design to be made. Questions about the automated response of the control logic unit became a very important issue. The major issues were the sheer number of the sensor data and control tracking real-time during the entire flight trajectory. The latter was an important hard constraint in the domain of Software Engineering.

Hierarchical decomposition:

There was thus a need to introduce a hierarchical structure to achieve computational tractability,

e.g. (Stengel, 93 [1]). Three logical abstractions can be thought of in the control system architecture, namely:

I. Strategic layer: Task scheduling, goal planning. This corresponds to the highest level in our architecture, which is the application layer.

II. Tactical layer: Guidance, navigation. This corresponds to the middleware level in our architecture.

III. Reflexive layer: Tracking, control navigation. This corresponds to the dynamic partitioning of the kernel and the middleware in our architecture.

The control system architecture that needs to be logically functional as general hierarchical systems and is derived from arbitrary decompositions can be extremely hard to analyze and verify. One of our goals in this design project was to examine this hierarchical structure when it comes to determining stability and signal processing. It was a robust analytic problem which required precise verification of the performance guarantees by construction. The hierarchical system thus designed is always consistent.

1.3 System Quantization

This section deals with some of the quantization aspects of the control system architecture. Here are some of the important ones listed. There can be an arbitrary number of system quantization vectors.

I. Quantization of feasible trajectories into trajectory primitives: a) Formalization of the concepts of maneuver and b) Consistent abstraction of the system dynamics

II. Hierarchical decomposition of the control tasks: a) Maneuver sequencing (guidance, trajectory planning) and b) Maneuver execution (control, trajectory tracking)

III. Control Synthesis: a) Building a maneuver library (with feedback control), b) Behavioral Programming - solve a mixed integer program on a "small" space and c) Hybrid control system with performance and safety guarantees by design

We can model the ALV's trajectory using logical Maneuver Automatons, which have the following properties:

1. Two classes of trajectory primitives (trim trajectories + maneuvers) Construct a Maneuver library, with a finite number of primitives.

2. Generate trajectories by sequencing such primitives.

3. All generated trajectories are solutions of the system's differential equations.

4. All generated trajectories satisfy the flight envelope constraints

1.4 Model-based Autonomous Design

Based on the description of the Maneuver automaton, the following questions were investigated in the prior research work:

1. How do we program explorers that reason quickly and extensively from commonsense models?

2. How do we coordinate heterogeneous teams of agents to perform complex exploration?

3. How do we couple reasoning, adaptivity and learning to create robust agents?

4. How do we incorporate model-based autonomy into computing devices?

Programmers generate breadth of functions from common sense models in the light of mission goals. This is called Models based Reactive Programming, where Programmer guides state evolution at strategic levels.

There was another way of approach through Commonsense Modeling, where the Programmer specifies commonsense, compositional models of ALV behavior.

The prior research work came up with a model-based execution kernel that reasoned through system interactions on the fly and performed significant search and deduction within the reactive control loop. This led to model-based co-operative programming, the salient points of whose programs were:

1. Specification of team behaviors as concurrent programs.

2. Specification of options using decision theoretic choice.

3. Specification of the timing constraints between activities

Here is a pseudo-code of model-based co-operative programming:

```
c,
if c next A
unless C next A
A, B
Always A
Choose reward
A in time[t, t']
```

Naturally, a model-based execution always achieves correctness and economy. It pre-plans threads of execution that are optimal and temporally consistent, responds at reactive timescales and performs planning as graph search. This approach is described in the prior work [2]

2 Research Challenges and the Evolution of the Adapted Solution Path

A very important aspect in our design was to make the control unit independent of physical parameters and constraints. We wanted two things simultaneously for the ALV: controlling via a base station externally, and an autopilot unit-on-chip for robust control. We wanted to have one single design for both the functionalities. Hence, component reuse and modularity was one of the key implementation goals in the system. There was also an ease in signal processing and involved less cost in the design without degradation of performance parameters. Also,the design provides for a continuous chain in adaptive signal processing for a huge number of data points from the sensors. The system is guaranteed to provide a stable and controlled flight path of the ALV for every instant of time and has no possibility of drifting away from the optimal control values.

2.1 Challenges of the Research

The challenges to the design were the following:

1. Eliminate parameter dependency on control.

2. Maintain constant optimality of control objective function by playing combinatorial games with the environment.

3. Design a single, modular, generic, reusable architecture for base-station controlled and autopilot controlled logic unit. This is done to reduce costs in the design and help eliminate multiple component distribution for multiple functionalities. There is a provision for integrating the control logic unit as a SoC (Systems-on-a-Chip). Verification of system design by automated test vectors for hardware-software co-design that cuts cost-to-market cycles.

4. Designing a real-time signal processing unit that could process an arbitrary number of input data points with minimum latency and overhead delays at a very high rate of parallelism. The challenge was to essentially reduce system calls to the Linux kernel for continuously varying input data

2.2 Salient features and accomplishments

1. We designed a robust control logic unit where the control objective function maintains a constant optimality by playing algebraic combinatorial games with the environment.

2. We designed a real-time control logic architecture that processes a large number of input data with minimum computational overhead.

3. We designed a scheme that allows real-time linking of computing hardware logic based on the control strategy chosen for stabilization.

4. The design provides minimal system calls to the Linux kernel due to the software architecture.

5. The embedded control logic units are described in terms of a novel mathematical structure called the ASMD (Abstract State Machine Dataflow),which fits in very nicely to the requirements of the design.

6. Introduction of object-oriented generative programming to model the embedded control blocks, development of ECO(Embedded Control Object)instantiates, description of fundamental embedded control logic using ECO states in the form of abstract state machines. This is mapped to the combinatorial game that the controller plays.

7. The Linux kernel structure and the invoked hardware logic are automated during real-time performance. Thus, an entire chain of functions is linked from the topmost high-level application layer to the kernel layer and back. A game (mathematics) is used to stabilize the flight (control), optimize the control objective function at all times (adaptive signal processing), and is mapped onto the working of a thread of the Real-Time Linux system (model

the ECO objects as ASMs, define the individual software architecture layers, and implement the kernel structure)

8. Dynamically partition the Linux kernel cooperatively for robust control and implementing the game function for control strategy to make the flight stable.

9. Hardware-Software co-design using the dynamic partitions of the kernel and the associated hardware logic.

10. Thoroughly automated test vector implementation for the verification of the Hardware-Software co-design.

11. Comparison with a commercial Real-Time kernel and showing that our designed architecture is superior with respect to performance metrics.

3 Games and Real-Time Embedded Linux Kernel Design

Games are modeled as mathematical objects. The control process is modeled as an algebraic combinatorial game. Decision theoretic considerations show that rational agents are strongly constrained in their behavior that they assign the probabilities to the occurrences of such outcomes with deterministic mathematical bounds. One of the main goals of this complexity theory is to present lower bounds on various resources needed to solve the computational problem. From a control system viewpoint, the most demanding problem is to prove nonlinear lower bounds on the complexity of designing such a non-cooperative game system. The task is to prove a statement without yielding anything beyond its validity (zero knowledge proofs)

It is possible to implement the control methodology knowing the details of the system up to a discrete time ahead. A description of the future system with gradually increasing uncertainty in time, linking of the present controller with a mathematical model for such uncertainty produces a more realistic control system. This leads to the distributed intelligent control system architecture that ports the mathematical transform logic via agent modeling to the control strategy based Linux kernel instantiation.

A control law is constructed for a linear time varying system by solving a two player zero sum differential game on a moving horizon, the game being that which is used to construct a Hinf controller on a finite horizon. Conditions are given under which this controller results in a stable system and satisfies an infinite horizon Hinf norm bound. A risk sensitive formulation is used to provide a state estimator in the observation feedback case. A receding horizon controller is formulated with each finite horizon optimization based upon a Hinf optimization. The control model is so chosen that the practical advantages of receding horizon control is combined with the robustness advantages of Hinf control. This controller is stable and satisfies an infinite horizon norm bound. This is ultimately mapped real-time using the concept of ASMs and designed on the embedded Linux kernel.

4 Approach to modeling embedded timed Linux systems using Abstract State Machine concepts

Modeling formalisms for real-world phenomena typically have some artifacts, i.e. some models have properties which do not correspond to real-world phenomena. For real-time system models, the most discussed artifact is the possibility of Zeno behavior. Another is unbounded activity. These are related to the games that our control logic plays with the environment. A Zeno behavior of a quantitatively timed or hybrid system model is a behavior in which an infinite number of discrete steps take place in a finite interval of time. The problem with Zeno behaviors is that they are considered to be too unrealistic or difficult to implement. Thus, while the abstraction that discrete events take no time and that between two discrete events no time is spent is admitted as a sensible simplification of the physical process, the assumption that an infinite number of such discrete events can take place in a finite amount of memory is not admitted as sensible. For our case of processing infinitely large data vectors from the ALV sensors, the processing block uses piecewise linear and sinusoidal functions to translate the strategy game to the computing unit. Hence, we must be careful about the Zeno behavior while modeling real-time control logic systems using Abstract State Machines (ASMs). The most important conditions are:

1. abstraction that discrete steps take no time is sufficient,

2. the assumption that time is linear and dense is sufficient, and

3. the flow of time alone cannot introduce nondeterminism is both necessary and sufficient.

Using these assumptions, it is plausible to describe the discrete steps of an algorithm with ASM rules,in terms of ASMD (Abstract State Machine Dataflow) making use of some condition on the current time for timing the activation of the rule. In this case we used the approach without problems for modeling a real time embedded Linux kernel which is instantiated with respect to control strategies. This concept leads to the definition of the control logic software architecture and the use of Embedded Control Objects (ECOs) that form the core of the processing system.

Since real-time algorithms are finally implemented in discretized time, it is possible to find weakly or even strongly well-behaved artifact rules describing the system dynamics. At a higher abstraction level, one might not want to have to deal with problems resulting from discretization, hence the success of this model in implementation. The concept makes controlling data intensive procedures fit in nicely to an embedded Linux kernel and the invoked hardware logic.

The Linux kernel design structure is expressed in terms of ASMD.

5 Software Engineering aspects of the embedded Linux kernel design

The research into the design and testing of a Linux system kernel for real-time control applications shows the concept of dynamically partitioning the kernel based on the strategy objective function for control and stability. The designed kernel is reconfigurable, multithreaded and preemptive and involves a data-driven scheduling strategy for Abstract State Machine Dataflow (ASMD) configurations. It uses the underlying hardware for high-speed context switching between the kernel and the applications. The features of the kernel can be configured according to performance requirements without a change in the applications. We performed a performance metric analysis with respect to the Texas Instruments DSP/BIOS architecture, a commercial real-time kernel software that is added on to any DSP chipset for real-time applications. Some of the results are given at the end of this paper.

The protocol design from the high-level application layer to the kernel layer makes the control engine integrated and modular in structure. The ASMD architecture supports component oriented reusability and parallelism in software design. ASMD applications consist of concurrent processes that interact with each other only by asynchronous messages sent through the data channels. Computation in such a design is data-driven, separate threads opportunistically process incoming data as it becomes available. This is a novel of the design is that the Linux kernel is called only when there is a significant deviation of the objective function from its stable value. The design of the circular buffer in the sub-interface layer has this unique functionality of making a comparison with the stored in objective function instantiate with incoming data, and only when there is a significant deviation does it call the processing kernel.

To recall the basic definition of an ASM, it is a set of states that can be reached on transition at different input vectors. The alignment of each state with a specific transition state determines the objective function value at a given time. The ASMD system is composed of concurrently executing processes that communicate with data channels. The concept can be thought of as nodes representing processes and arcs representing data channels. Each ASMD process executes opportunistically, operating on incoming data as it becomes available. Because the processes share no state, know nothing about the other processes in the system, and communicate only through asynchronous data transmissions along data channels, there is a high degree of independence between them.

Data-driven nature of ASMD applications poses different scheduling requirements than traditional concurrent programming techniques. If ASMD processes are awakened by every incoming data message, they may spend much of their time managing the state of their data channels and waiting for all the necessary data needed to proceed. Finally the fact that processes only communicate via data channels implies that there are frequent (but usually small) interactions between processes along these channels. Ensuring mutually exclusive access to critical state within a data channel provides another potential for increased overhead in dataflow applications. The ASMD architecture takes the advantage of dual-register sets of the underlying hardware architecture to reduce context-switching overhead. It also provides support for dynamic priorities, 'firing rules' for specifying the data channel conditions necessary for process wakeup, and typed data channels for efficient and reliable interprocess communication. The reconfigurable options of the Linux kernel facilitate selective removal of unneeded features to improve performance without requiring any change to the application code. For example,the user can remove dynamic scheduling or preemptive features to meet the high-performance requirements of a particular application.

This report shows a comparison of our designed

architecture with DSP/BIOS of Texas Instruments for different control algorithms. The results indicate an improvement of around 10 percent for our design over the TI architecture performances.

Research Contributions to embedded Linux Kernel Design:

1. Dynamic co-operative partitioning of the Linux kernel based on control strategy.

2. A message-driven scheduling algorithm that eliminates busy waiting of ASMD threads.

3. A high-speed context switching strategy that takes advantage of processors with dual register sets.

4. An interprocess synchronization technique that provides a performance increase for ASMD applications without the overhead of semaphores.

5. Minimal interrupt handling.

6. Kernel configuration options that allow the user to selectively remove unnecessary kernel features, reducing overhead by more than 80 percent.

6 More about the Embedded Real Time Linux Kernel

6.1 Kernel Requirements

The following list summarizes the main differences between ASMD based Linux Kernel design and other applications:

1. Number of Components: In a traditional embedded application,the system is divided into components that are in turn encapsulated into separate processes or threads. The issue of modularity and reusability becomes more important in an ASMD application because each node is viewed as a reusable module designed to perform a specific function. Therefore, an ASMD application tends to have more components than applications developed using other techniques.

2. Inter-Component Communication: The larger number of components or nodes in ASMD applications increases the frequency of inter-component communication. Each node communicates with its neighbors through directed data-channels and presence of data in incoming data channels triggers the receiving component.

3. Scheduling Requirements: The data-driven nature of ASMD applications poses different scheduling requirements than more traditional concurrent programming techniques. If ASMD processes are awakened by every incoming data message, they may spend much of their time managing the state of their data channels and waiting for all the necessary data needed to proceed.

These differences between ASMD and traditional applications lead to different set of requirements for the Linux kernels designed for ASMD applications. In addition to these requirements, there are also some domain-specific requirements that should be met by any kernel.

6.2 Kernel Requirements imposed by the ASMD design

The main requirements imposed on the underlying kernel by ASMD applications are:

1. High Performance: The components of ASMD are used to replace the equivalent hardware components in the sensor controllers.In these cases, the execution speed of the control software is an important factor because software is generally slower than hardware. Hence the designed kernel should have minimal overhead and high execution speed.

2. Faster Context Switching: ASMD applications tend to have a larger number of processes or threads than applications developed using other techniques. While a larger number of independent, reusable, components make application design easier, it can lead to an increase in the amount of context switching overhead. The kernel should make an attempt to reduce this overhead by increasing the speed of context switches as well as limiting the number of context switches.

3. Efficient Inter-Component Communication: The fact that processes only communicate via data channels implies that there are frequent interactions between processes along these channels. Ensuring mutually exclusive access to critical state within a data channel provides another potential for increased overhead in ASMD applications. The kernel should provide support for efficient inter-component communication with minimum overhead.

4. ASMD Scheduling: Unlike traditional processes that are scheduled based on their priorities alone, ASMD processes are scheduled on the basis of both the priorities and data in the incoming data channels. Moreover, the ASMD processes should not be awakened by every incoming message. The kernel should provide an efficient mechanism to specify when an ASMD process is ready to execute.

5. Component Execution with Dynamic Priorities: An ASMD process can wake up due to data in different sets of incoming channels. Depending on the set, it can take specific actions. The kernel should facilitate this in addition to adjusting the process priorities according to the actions they are taking.

This is a high-performance embedded Linux kernel that specifically addresses the requirements discussed in the previous sections. It implements a scheduling and context switching strategy optimized for ASMD applications based on the control strategy chosen to stabilize the flight of the ALV. Further, it takes advantage of the invoked computing hardware logic to drastically reduce context-switching overhead. It also provides support for dynamic priorities, firing rules for specifying the data channel conditions necessary for process wakeup, and typed data channels for efficient and reliable inter-process communication.

The embedded Linux kernel design is implemented in C, with a few key elements in assembly (context switching), dual register set support and interrupt handling). ASMD processes, or Embedded Control Objects (ECOs),are implemented as C functions. It uses a dynamically initialized array of ECO descriptors, together with a dynamically initialized array of data channel descriptors, to initialize the application at startup.

Our design provides options to users to selectively include real-time features in the embedded Linux kernel. The optimal fixed priority algorithm [3] is shown to be the dynamic rate monotonic priority assignment (DRMA) in which a task with a shorter period is given higher priority than a task with a longer period. In case of deadline driven scheduling algorithm, the deadlines are monitored at each clock tick to assign the highest priority to the task with nearest deadline. Our designed kernel provides support for using the fixed priority real-time scheduling algorithms. Along with the DFG (Data Flow Graph between two ECO's, that's mapped bijectively to the ASMD processes) definition, the user is given the option to specify a statically scheduling algorithm through a function handle. If the function handle is assigned null, the default scheduling provided by the kernel is used. It provides an implementation of the priority DRMA that can be optionally used by the application designer to assign priorities according to the rate monotonic approach. DRMA uses the information in the DFG to calculate the priorities and exits without starting the application if a feasible schedule cannot be found. In addition to this, the design provides a simple API to applications for monitoring their real-time deadlines.

7 Dynamic Linux kernel partitioning and reconfigurable hardware

This section describes a hard realtime Linux environment RTL that can be effectively used to design the kernel of the embedded control unit. We integrate the functionality of the MATLAB/ Simulink / Real-Time-Workshop (RTW)suite and RTL. The resulting software can be run on or off the shelf in standard personal computers, without requiring overly complex and expensive hardware architectures often associated with specialized real time systems, without any performance loss.

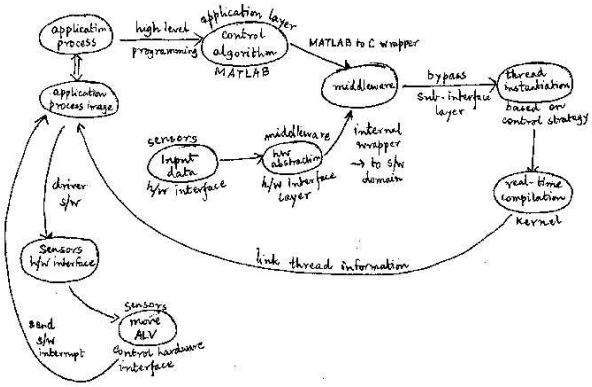

FIGURE 1: *Co-operative dynamic hardware-software partitioning of embedded kernel: Autopilot Control Logic*

Furthermore,by exploiting the new development environment, a Linux kernel module that allows building hard real time applications in user space, we can initialize the kernel real-time using the control strategy that is used for optimizing the combinatorial game function. Here lies the bridging effect of ECO data structures. This exactly maps the operational logic to the embedded system, thus creating a bijective map of the executable control transform and the invoked hardware logic, thereby reducing system calls to the kernel. Another aspect is to write the wrapper initializations from the middleware abstraction to the kernel, and to port the operator transform

code to the target hardware used by the embedded Linux system. This is one of many interesting researches in this area.

The following two diagrams show the logic design flow for the dynamic Linux kernel partitioning.

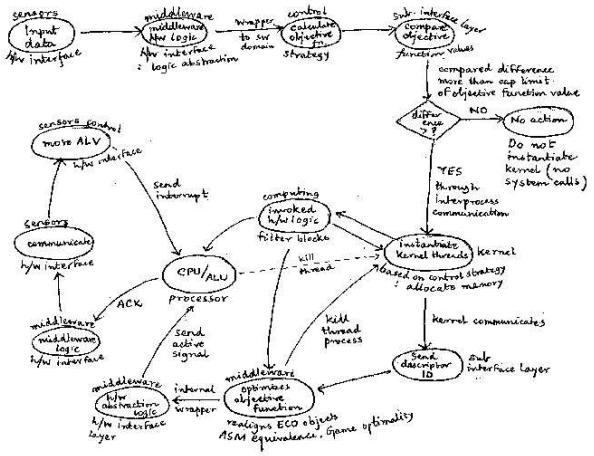

FIGURE 2: *Dynamic partitioning of embedded Linux Kernel- Real time compilation of Control Logic Code: Deterministic Control Logic: Base Station Controlled*

8 Reconfigurable Computing Hardware Mapping from the Transform Iterator Library in the Linux Kernel

The ECO data structure processing application benefits from an architecture consisting of multiple register files, each conforming to the following properties: 1. Each register file will have to support a number of multiple read/write ports that is at least equal to the number of states in the Input Data Type (IDT) sequence. Typical numbers range from 4096 to 65536. 2. The memory access pattern of the register file highly structured. Data is alternatively written in the forward and backward directions. The same applies to the read pattern. There will be 128 read accesses for every written data. 3. In-place storage and single cycle read/write, where data is read during the first half of a cycle, followed by a write during the second half.

The structured access pattern makes it redundant to implement an address decoder that is commonly found in register file and general purpose embedded memory designs. Our architecture that implements the address decode utilizes this functionality.

A novel Combinatorial Real-Time Algorithm (CRTA)is proposed for this purpose, that follows from the Logic Flow diagrams of Figure 1 and Figure 2.

The CRTA algorithm indicates that the size of memory is of the order of O(N x D x B), where N= Number of states in the IDT, D= Depth of the trace forward/backward path, B= Number of bits in the fixed-point algorithm.

The states of the IDT can be drawn from any arbitrary number of permutations of the input data sequence. Typical values of N range from 1024 to 4096. Values of D are between 512 and 1024. Bit resolution B is set at 4 or less.

Results:

CRTA time execution = 0.5 s for N 1024 and D 512 with B 4

9 Invoked Hardware Logic and the Embedded Linux Kernel

This brief section discusses about the extension of Gabor expansions to the embedded Linux system domain and the design of an efficient filter bank to provide strategy instantiation computing for the embedded kernel. This has been shown in the Logic Flow Diagrams in Figures 1 and 2. The isomorphism between this localized linear operator and the filter design fundamentals for the invoked hardware logic are examined in the framework of the ALV autopilot control unit, which is modeled using ASMD concepts. This design degenerates into a mathematical model of a quincunx filter bank for control strategy dependent thread instantiation in the kernel. This invoked hardware logic unit helps in processing the control objective function for large number of sensor data units and is currently under the focus of our research.

10 More performance metrics

10.1 Comparison of our design with a commercial real time kernel the Texas Instruments DSP/BIOS

There are a few notable differences in architectures with the TI DSP/BIOS with respect to real-time performance analysis. Some of them are:

1. The real-time Linux kernel uses buffer level metaprogramming for message event logs, where the results are shown with every thread instance that is initialized. This reduces the need of dedicated resources for explicit event logging. The target program only calls the

kernel resources implicitly when threads become ready, dispatched and terminated. Also, metaprogramming allows any specific dissertation to be logged in during failures. This particularly helps in debugging and keeping track of a large number of input data.

2. Statistics accumulators are integrated with the handles of the sub-interface layer processes. The target program does not accumulate statistics explicitly. This is also a characteristic of the architecture that reduces overhead. It is implicitly used by the kernel when scheduling threads for execution or performing I/O operations. In the DSP/BIOS, the target program accumulates statistics explicitly through DSP/BIOS API calls.

3. The host data channels in the kernel are integrated with the IPC channels between the ECOs and are mapped onto the circular buffer logic in case of our designed kernel. Different from the TI DSP/BIOS kernel where the host data channels provide the target program with standard data streams for deterministic testing of algorithms, our designed Linux kernel provides this as part of dynamic partitioning of the kernel at run-time.

4. There is no host command server in our kernel, unlike the TI DSP/BIOS. This is because of metaprogramming considerations where visibility of real-time program execution is integrated into message event logging.

10.2 Performance Metrics comparison with TI DSP/BIOS

1. Unlike the DSP/BIOS case where a small RTDX (Real-Time Data Exchange) software library that runs on the target DSP, it is the ASM library that is used to monitor the target processor. This emulates the hard and soft real-time functionalities and the mode of communications with the host is through software and not through physical JTAG interfaces. It is analogous to creating an image of the host processor in the kernel logic. Of course, this provision is flexible and can definitely be used with the scan-based hardware emulator that DSP/BIOS uses.

2. The ASM library of the middleware does not run in conjunction with the developers' Integrated Development Environment on the host platform. This is again different from DSP/BIOS. The communications metrics show an improvement factor of about 5 percent overall at process time. The image of the host supports both continuous and non-continuous modes of receiving data from a target application. This is similar to the DSP/BIOS architecture except for the fact that the logs are not stored statically unless the developer wants it to. Our design provides less flexibility in this aspect.

3. Hardware abstraction: Our designed kernel provides APIs to access and configure hardware configurations independent of physical implementation. In our design, simple logical maps provide the functional interfaces to the invoked hardware logic. In case of DSP/BIOS, the hardware abstraction is static and configured one-time. In our design, the abstraction may change depending on the alignment of the underlying filter blocks. It supports memory management and the designed APIs provide dynamic allocation and freeing within the application. The DSP/BIOS uses the logical memory map within the MEM module, our kernel does this by the ASM logic and soft state considerations.

4. Memory Management: The architecture of our designed kernel differs in the sense that there is no separate module that provides a set of runtime functions to allocate storage from one or more segments of memory as in DSP/BIOS. Most of the work in parallel transform executions on multiple input data vectors has been done for single dimensional search problems. This is because it is possible to define a function from a given ECO data structure to an ordered pair of input data vector and the corresponding transform. The mapping * serves to define this function from the system level code generated by the user. Speaking differently,the mapping * uniquely assigns the relation of a given input data to the computation that the user wants to perform on it. One important aspect of the approach followed is that it reduces the accesses in the tree only to a small closed portion of the tree. It has another advantage of having a fixed upper bound on the total number of locks held by a process. There is also a logical cached-in buffer for computational and retrieval access. This results in an improvement in throughput that is linear in the number of simple processors. Timing measurements carried out by implementing this method verified the performance of this scheme.

5. Chip Support Library: In the DSP/BIOS, the Chip Support Library (CSL) provides a C-language interface for configuring and controlling on-chip peripherals. It consists of discrete modules that are built and archived into a library file. In our designed kernel, this is provided as a link to the map of the hardware filter blocks. The hardware logic initialization is done from this library and it primarily sits in the middleware stack.

11 System Performance comparisons for our kernel and TI DSP/BIOS

Environment: Testing platform: Intel Pentium 4 CPU 1300 MHz 1.28 GHz 256 MB RAM

The component overhead in instruction cycles may be taken from the DSP/BIOS performance data listed in DSP/BIOS Timing Benchmarks on the TMS320C6000 TM DSP for CCS 2.0 (SPRA662). Calculate the number of occurrences for each component, and then add the total number of cycles. For example, a single buffer requires the total overhead of 1351 cycles on a TMS320C6000 TM. The processing period is 4 ms, so the frequency of occurrence is 250 times per second. Therefore, the total number of cycles in 1 second is 337750 or 0.33775 MIPS. On a 200 MHz C6000 DSP, this equates to 0.17 percent CPU load.

Our results show the total number of cycles to be 251735 or 0.251735 MIPS, which equates to a 0.11 percent CPU load

References: 1. DSP/BIOS Timing Benchmarks on the TMS320C6000 DSP for CCS 2.0 (SPRA662) [4]

12 Conclusion

The design effectively shows the application of theoretical mathematical tools to solve a very difficult problem in controlling the flight dynamics of an Autonomous Landing Vehicle. The control system is very generic and fundamental in nature and does not depend on any physical parameters of the ALV. The beauty of using Game Theory to real-time control systems lie in the fact that the entire flow of control can be modeled as abstract state machine maps. These maps establish a complete link from the application level to the embedded Linux kernel and link the kernel along with the invoked hardware computing blocks real-time. This linking is only a function of the control strategy that is chosen. This architecture is particularly suited for a large number of sensor data points, and it minimizes system calls to the kernel based on the objective function limit threshold, and thus prevents overload in computing resources.

Acknowledgement

This work was supported in part by the National Aeronautics and Space Administration (NASA) through the Cornell Aerospace Systems Technology and Rocket Operations Group, U.S.A., and the Korea Science and Engineering Foundation (KOSEF) through the Ministry of Science and Technology (MOST) and the Institute of Information Technology Assessment (IITA) through the Ministry of Information and Communications (MIC), Republic of Korea.

References

[1] Stengel, R.F., Nov-Dec 1993, *Toward intelligent flight control*, Systems, Man and Cybernetics, IEEE Transactions on, Vol. 23, Issue 6, pp. 1699–1717.

[2] Grimson, W.E.L., June 1989, *On the Recognition of Curved Objects in Two Dimensions*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol. 11, Issue 6, pp. 632–643.

[3] Guha, D. and Hassan, M., May 25-28, 2004, *Co-operative Dynamic Partitioning of a Real-Time Kernel in an Autonomous Landing Vehicle (ALV) Controller*, 10th. IEEE Real-Time and Embedded Technology and Applications Symposium 2004, Toronto, Canada, Work-In-Progress Paper.

[4] Texas Instruments TMS320C6000 DSP for CCS 2.0 (SPRA 662)