Portable Kernel for High-Level Synthesis of Complex DSP-Systems *

Ted Bapty and Ben Abbott
Department of Electrical Engineering
Vanderbilt University
Nashville, TN 37235, USA
bapty@vuse.vanderbilt.edu

Abstract

Modern DSPs are low cost and have integral communications, encouraging their use in large parallel systems. Effective software methods are critical to the success of this trend. A *model-based program synthesis* approach has been proven effective for developing large parallel instrumentation systems. This paper describes an essential component of this approach, the low level kernel. The kernel has been designed to support the facilities necessary for the model-based synthesis approach. The kernel supports parallel, real-time systems, requires minimal resources, and is portable across many embedded and general purpose machines.

Introduction

As DSP's gain computational power and the ability to communicate, many exciting and challenging application areas open. These applications are typically embedded into complex systems, such as manufacturing plants, aerospace systems, and testing facilities. The jobs include monitoring, analysis, and system control.

The computational requirements overrun the capabilities of a single processor, mandating the construction of large parallel DSP systems. The designers are then faced with two challenges: 1) understanding the domain where the system will be applied, and integrating into the environment; and 2) understanding the complexities of a parallel real-time DSP system and developing low-cost, reliable software systems.

While not completely solved, the first challenge has been addressed effectively using the technique of Model-Based Systems (See figure 1). This technique involves:

1. Representing the environmental factors, design goals, real-time constraints, and available hardware platforms in a consistent, domain-specific *Model*. The models are often graphically depicted, in a paradigm familiar to the intended user.

2. Interpreting these models to *Synthesize* executable systems. The products of the

*This work was supported in part by the AFOSR/AFMC, United States Air Force, contract number F49620-94-C-0076.

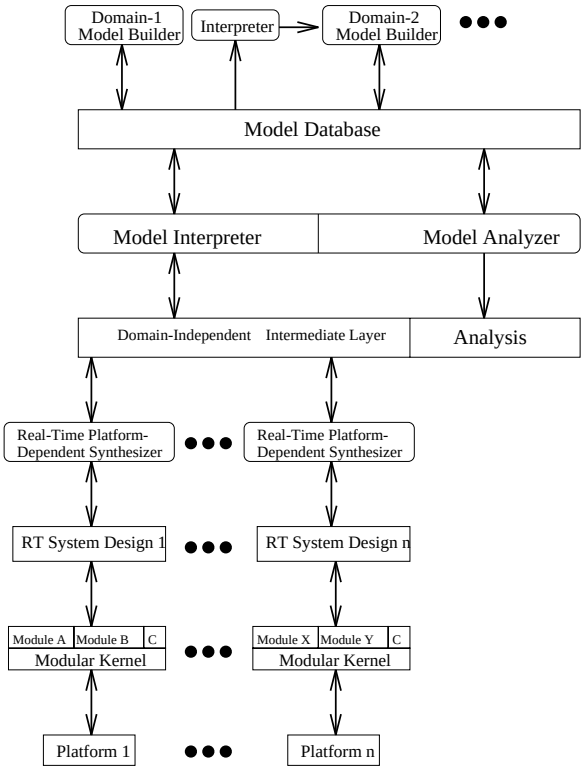

Figure 1: Model-Based Synthesis Architecture

synthesis process are hardware specifications (Processors and Connections) and software specifications (Tasks and Datapaths).

3. Wiring the synthesized hardware architecture and Building the synthesized software design upon the hardware.

4. Execution of the design in the embedded, real-time application.

The models and interpretation methods are kept domain-specific, to limit the complexity of the software synthesis and to keep representation mechanisms familiar to the domain experts. The output of the interpretation process is a graph-based formalism, representing the computational structures and information flow patterns describing the synthesized system. From this description, a variety of system implementations are possible, varying in performance, implementation technology, DSP choice, etc.

The Modular Kernel

The implementation of efficient parallel real-time, embedded, DSP-based systems is a challenging task. This paper describes a kernel with the necessary facilities to support automatic software synthesis of these systems. The goal of this kernel is quite different from other real-time kernels: programmer convenience takes a back seat to run-time efficiency and system flexibility. System configuration and programming that would be considered too tedious for a programmer to manage are automatically generated by the high-level tools. The system user never interacts directly with the kernel. We must support only the minimal set of features required to make synthesis efficient.

The design requirements for the kernel are as follows:

1. **Heterogeneous Multiprocessor:** interconnecting hundreds of different processors in a single system;

2. **Hard and Soft Real-Time:** supporting applications that interact with time-critical sensors and actuators.

3. **Size:** kernel and application must fit on embedded processors, in 32 K RAM;

4. **Efficiency:** minimal overhead in system services to reduce hardware weight, size, and cost.

5. **Portability:** supporting TMS320C3x and C4x, transputers, PC's, Workstations;

6. **Maintainability:** The number of errors within any code is proportional to the size and complexity of the code. Decreasing both factors enhances the possibility of a low-cost maintenance arrangement.

7. **Simplified Application Development and Debugging:** Errors in the application are bound to show up in side effects within the kernel, since the embedded system has little in the way of memory protection. Simplified data structures make detection and localization of these errors more manageable.

To satisfy these requirements, the Modular Multigraph Kernel (MMGK) has been implemented. An overview of the kernel structure is shown in figure 2. The primary components of the kernel are:

1. **Real-Time Scheduler:** The scheduler modules are data-driven dataflow, synchronous static scheduling, or a dynamic algorithm, such as rate-monotonic scheduling.

2. **Communications:** an asynchronous communication system providing streams between adjacent processors. Communication is guaranteed real-time for the statically scheduled case.

3. **Memory Management:** a structured buffer pool.

Key design decisions have been to integrate the memory management, communications, and scheduling in a coherent manner. Several benefits are achieved through this integration. Asynchronous communications can occur with no extra memory copying. Communications are schedulable as well, with options of fifo and priority ordering. The schedules

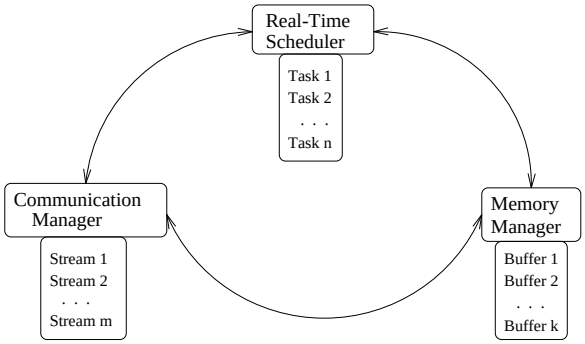

Figure 2: Basic Kernel Architecture

and communication streams can be reassigned dynamically, allowing for structurally adaptive systems.

Kernel Internals

The tasks operating under the kernel see a very simple external interface. Only a handful of functions are needed. These functions are (see figure 3):

- *long *get_input_buffer(index)* returns a pointer to the head of the input queue on the indexed input port. The data remains in the queue until explicitly removed.

- *void dequeue_input(index)* is used to explicitly remove the buffer from the input stream queue. The process is responsible for returning this buffer to the memory management system.

- *void enqueue_output(int index, int size, long *ptr)* adds the buffer *ptr to the end of the output stream queue. The buffer will be transmitted and automatically returned to the memory management system. A buffer can only be propagated or returned once. Multicasting must explicitly allocate and copy buffers.

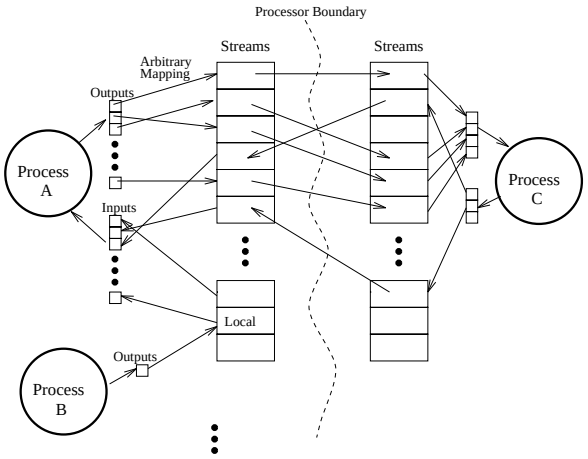

Figure 3: Kernel Task Interface

- *int output_slot_available(int index)* determines if there is space in the stream output queue. *int buffers_in_queue(int index)* returns the number of buffers within an output queue.
- *long *get_buffer(int size)* returns a pointer to a buffer of length = *size* * sizeof(long).
- *void return_buffer(long *buffer)* returns a buffer to the structured buffer pool.

Application of the Kernel

The kernel has been used in the implementation of several large-scale, real-time instrumentation system that are currently in use for the processing of engine test data at Arnold AFB and elsewhere. The largest system integrates over 60 TMS320C31 and C40 processors with transputers and PC's in an interactive processing system with a graphical front-end. This high-speed computer architecture continuously processes from 48 to 192 high-bandwidth signals, achieving a sustained performance of over 800 MFLOPS (from a theoretical peak of approximately 2 GFLOPS).

The system is named the Computer Assisted Dynamic Data Analysis and Monitoring System (CADDMAS). Other heterogeneous CADDMAS architectures use INMOS transputers for communication and general purpose processing, Zoran and Motorola Digital Signal Processors for signal processing operations, and Texas Instruments Graphics Processors for on-line graphical display of calculated data. In all, more than ten of the systems have been constructed.

Testing turbine engines involves running an instrumented version of the engine through various operational maneuvers (e.g. Acceleration or Throttle Snap). These tests are typically conducted while the engine is in a test cell (wind tunnel) capable of simulating altitude, atmospheric, and air speed conditions. In order to analyze dynamic vibrations, strain gauges (and other stress sensors) are attached to the turbine fan blades. A typical aeromechanic stress test instruments the engine with several hundred stress sensors along with a variety of temperature, pressure, flow, and revolution per second sensors. Stress sensors can generate signals with bandwidths in the tens of Kilohertz.

Historically, analysis of turbine engine stress data has been an off line process. On-line capabilities were limited to oscilloscopes showing unprocessed amplitude vs. time information and a small number of signal analyzers for simple spectral analysis on single channels. The bulk of the raw information was recorded onto analog tapes. Later, the analog tapes were digitized into conventional computers for analysis. The processing of this data was extremely compute intensive, and consequently, only a selected portion of the data was reduced. The analysis imposed a delay of several weeks on the availability of final results. Thus, vital

information was not available for on-line test planning and evaluation.

The CADDMAS system was developed to provide these capabilities on-line. The system processes all sensor readings and presents the results both graphically and in hard copy form, during the test. The immediate availability of results opens the possibility for interactive test planning.

A graphical user interface allows the user to configure various visualization screens interactively. The user can select the number of visible windows on a screen, the contents of each plot window, and the parameters of each plot, such as titles, labels, axis ranges, and plot type, and display window update rate. Stored configurations automate the operation of the user interface. The user can also print any window or all windows on a screen.

Conclusions

The model-based approach has proven to be a very useful tool to help manage the complexity of this large, parallel system. A small team of two engineers has constructed every CADDMAS system to date. A large, 24 channel system can be built in a matter of hours. The kernel is an integral part of the model-based strategy. The small size allows the use of inexpensive processor modules. The simple structure of the kernel allows easy porting to new architectures and inexpensive maintenance of the code. As soon as the the kernel is supported on an architecture, the full power of the high-level model-based synthesis tools are available.

Future plans for the tools and techniques include: better methods to map the signal flow graph to the hardware; advanced synthesis techniques; support for more real-time schedulers; support for more hardware architectures.

Future plans for CADDMAS applications include several 300 channel 50 KHz systems using Texas Instruments TI 320C4x processors in conjunction with transputer technology. An effort is underway to commercialize the technology.

REFERENCES

[1] Abbott, B, Bapty T. Biegl, C. Karsai, G., Sztipanovits, J. "Model-Based Software Synthesis", IEEE Software, pp. 42-53, May, 1993

[2] Karsai, G., Sztipanovits, J. "A Visual Programming Environment for Domain-Specific Model-based Progrmming.", IEEE Computer, March 1995.

[3] Ledeczi, A., Abbott B.A., "Parallel Systems with Flexible Topology", Proc. of the Scalable High Perfomace Computing Converence, pp271-276, Knoxville, TN 1994.

[4] Abbott,B. A., Bapty T.A., "Experiences Using Model-Based Techinques for the Development of a Large Parallel Instrumentation System", Proc. Conf. Signal Processing Applications and Technology, Cambridge, Mass, pp573-582, 1992