In ACM SIGPLAN Notices, December 1994.

ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems

June 21, 1994
Orlando, Florida
(Summary of the workshop)

Azer Bestavros (best@cs.bu.edu)
Richard Gerber (rich@cs.umd.edu)
Steve Masticola (masticol@scr.siemens.com)

The first ACM SIGPLAN Workshop on Language, Compiler, and Tool Support for Real-Time Systems was held on June 21, 1994, in conjunction with the annual conference on Programming Language Design and Implementation.

The workshop was well attended by researchers in both the programming language and real-time communities. The synergy between the two communities was remarkable and exciting; discussions were lively; and many attendees claimed that the workshop had stimulated new research ideas that they were going to pursue.

The workshop was also notable for its level of discourse. Problems and ideas were traded freely, cleanly, and with almost a complete absence of confusing jargon. "Novices" and "experts" mingled freely; there was significantly more free-wheeling participation than one finds at established venues.

For a first meeting, the turnout was also surprisingly international – with papers from the UK, Germany, France, Canada and Austria.

Based on this promising start, a second meeting of th Workshop will be held in 1995, in conjunction with the '95 PLDI conference in La Jolla, CA.

This report summarizes the presentations given at the workshop. The summaries here have been rearranged to group similar topics together. For each of the papers described here, the presenter, rather than the full set of authors, is given.

Topics, Themes and Controversial Questions

Bill Pugh and Tom Marlowe, in their roles as conference organizers, laid out the basic structure of the workshop and the topics they planned to cover. Then they asked Rich Gerber to posit a series of "controversial questions, statements, and examples," with the intention of seeding the discourse for the remainder of the day. The issues raised by Pugh, Marlowe and Gerber included:

(a) The difficulty of performing good timing analysis in the face of more complicated architectures (e.g., with hierarchical caches, multiple pipelines, shared and local memory, etc.).

(b) The desire – on the part of many managers and engineers – to have complete traceability down to the machine-code level. (Reasons for this include inevitability of instruction-level debugging, etc.)

(c) The implications of RISC architectures on embedded systems development, with respect to the compiler's effect on the code produced, its readability, etc.

(d) The effect of generic compiler transformations in general: How the common reordering optimizations can inject errors in a real-time application by "misinterpreting" the programmer's intentions.

(e) How compiler techniques can help programmers tune code (while preserving intended semantics), either automatically, or by simply identifying blocks that the programmer can move selectively – perhaps with graphical tools.

(f) How data-flow analysis techniques presented at PLDI conferences can be used in automated verification.

(g) How Ada and Posix 1003.4-based systems can be used to support real-time systems.

(h) That there are tremendous opportunities for programming language researchers in this domain, if they decide to pursue them.

Gerber, a professor at the University of Maryland, sought to stimulate lively debate. His most memorable "provocation" was the proposition that the applicability of static, *a priori* timing analysis is exaggerated. His claim is that modern architectures are incredibly complicated; modeling pipelines, caches, shared memories, register windows, etc. using vendor-supplied benchmarks has become nearly impossible. This position provoked spirited discussion which continued throughout the day, both inside and outside the sessions.

Gerber also observed that, when judging a system's performance, the end-to-end constraints are the only meaningful ones: examples are end-to-end response time, jitter, update rate and temporal correlation of input samples. He claimed that the periods of component processes – which worry software researchers – are just the artifacts used to achieve the original constraints. Since the intermediate deadlines are often selected by design engineers without careful thought, our efforts spent in meeting them are often wasted time.

Gerber identified the major requirements for a real-time programming language: a source-level syntax for timing constraints, an unambiguous semantics which characterize the constraints, compiler technologies which preserve the semantics; and solid runtime support to carry out the programmer's intentions.

Working Lunch. After the morning session, the participants shared a working lunch, during which the "compiler" and the "real time" researchers were randomly distributed at the small tables. Some of the seed questions were informally pursued. For example, at one table David Chase (Thinking Machines) and Steve Tjiang (Synopsis) suggested how special RISC compilers could help alleviate the problems identified in (c)-(d) above.

Keynote Tutorial: Ada '94

Ted Baker of Florida State University, who is working on the runtime system for the Gnu Ada project, gave a tutorial talk on the Ada '94 extensions for real-time support. Ada '94 includes extensions for object-oriented programming, numeric libraries, and real-time systems. In contrast to Ada '83, which is a single monolithic standard, Ada '94 includes a core language and extensions (including one for scheduling.) The legality of subsetting was emphasized in the new standard.

Features for real-time support include:

- Data sharing; both volatile and protected objects.
- Asynchronous transfer of control: it overcomes many of the original problems with the synchronous "rendezvous," it incorporates timeouts, and it has a clean semantics for exceptions.
- `DELAY UNTIL` ⟨fixed time⟩, i.e., for absolute timing constraints. Questions about "implementation" and "verification" issues of code containing such a construct were raised.

- Additional `TIME` types.
- Entry service reordering via "requeue" statement.
- Interrupt handling via "protected procedures."
- Priority-based scheduling, along with priority ceilings; and the ability to dynamically adjust priorities.
- Low-level tasking primitives: `SELECT` ⟨interrupt handler⟩ `THEN ABORT` ⟨abortable code⟩. This construct can support timeouts.

Presentations on Language Design

Eric Rutten: SIGNAL. SIGNAL is a data flow language with notions of instantaneous reactions (clock, signal, etc.) Its target applications are reactive systems (signal processing, robotics). SIGNAL's approach is to define tasks and time intervals as primitive constructs.

"Signals" are typed values indexed by time. "Events" are signals whose presence is the only important feature (i.e., clocks.) Equations are relations on data values and clocks.

Carlos Pereira: C++/Real-Time UNIX. Carlos Pereira (University of Stuttgart) gave a talk on a real-time C++ environment, built on top of the QNX Posix-compliant kernel. The goal of this language is the rapid prototyping of real-time systems, using standard C++ style templates. To maintain real-time control, Pereira has imposed a antisymmetric, "communicates-with" relation on object classes: Active classes map "client instances" to RT-UNIX processes, whereas interface objects define "server instances" to assist in interprocess communication.

Paul Dasiewicz: Realtime programming in RTX–PARLOG. In concurrent logic programming, there is typically no support for real-time programming. Dasiewicz addresses this in RTX–PARLOG. This support takes the form of abstract data types for "time" and "duration," a scheduler with selective scheduling parameters (priority and task duration), and process timeouts. Garbage collection can be dynamically scheduled. RTX-PARLOG can generally produce only one solution to a program, since backtracking between processors is mostly eliminated due to guarded committed choice.

Presentations on Timing Constraint Validation

Roderick Chapman: SPARK Ada formal methods. Roderick Chapman of the University of York (England) described SPARK Ada, a subset of Ada designed to be used in formal proofs of correctness of realtime systems. Chapman's proof system can not only prove the worst case execution time, but can also identify the program path which yields the worst case.

Chapman divides the set of paths within a procedure into "modes," one for each call site. This has interesting parallels with Landi aliasing and Pande def-use analysis; a mode would correspond, in abstract interpretation terms, to the (gamma) concretization of an assumed alias or assumed reaching definition.

This work stems from an ongoing partnership between York and British Aerospace; it is a nice prototype for sound industrial/university collaborations.

Alex Vrchoticky: Compilation Support for Execution Time Analysis. Compilation obscures the relationship between source and transformed code. Vrchoticky presented his timing tree data structure as a partial remedy to this problem. A timing tree is unchanged by local program transformations, and can in theory recover the timing assumptions of the original program. Presently this method cannot represent irreducible flow-graphs, or "strong" deviations between the source and generated code (e.g., those caused by loop unrolling.)

Azer Bestavros: Toward Physically Correct Specifications. Azer Bestavros of Boston University spoke of a formal model and of a simulation/programming language (called "Cleopatra") for embedded control applications. The model/language prohibit users from specifying systems with certain physically-impossible features (e.g., lossless channels, exact delays, simultaneous dependencies, and constrained input transitions.) Bestavros argued that this "ounce of prevention" at the specification level is likely to spare a lot of time and energy in the development cycle – not to mention the elimination of potential hazards that, otherwise, would go unnoticed. Plans for a Cleopatra-based programming environment to control robotics devices were mentioned.

Kannan Narasimhan: Portable Execution Time Analysis. Narasimhan described an extensible, pipeline-analysis tool for automated timing predictions. The tool uses a generic abstract architecture language to describe the CPU at hand, and can thus be applied to most RISC machines. Thus far the tool has been tested on the MPC601 (power PC), Sparc and R2000 processors.

Presentations on Cache Modeling

Swagato Basumallick: Realtime Cache Survey. Swagato Basumallick of Iowa State reviewed several approaches for dealing with caches in realtime systems. (Caches are problematic for realtime computing, since they make it hard to predict the execution time of basic blocks.) The methods reviewed were:

- Analysis by synthesis: Get the compiler to generate code whose behavior is easy to predict.
- Graph coloring: Prove that cache lines are clear of interference.
- Extended timing schema: Maintain state of cache information (must-hit, must-not-hit, may-hit) for each source statement. (See Mueller, below.)

Basumallick advocated static scheduling to give a more global view of execution time requirements. However methods like static-priority dispatching could succeed, given a priori knowledge of the worst-case preemption overhead for every task pair.

Frank Mueller: Predicting Instruction Cache Behavior. Frank Mueller of Florida State University described a tool to predict the behavior of instruction caches. Mueller uses a "static cache simulator" to determine which program lines are potentially cached. Program lines are divided into one of four categories on the basis of this analysis: *Always Hit*; *Always miss*; *First Miss, then always hit*; and *Conflicts (miss sometimes, hit sometimes)*.

Using this foundation, the instructions are annotated with a "fetch from memory" bit for schedulability analysis, etc.

In a later talk ("Debugging Realtime Applications"), Mueller described the use of the static cache analyzer during dynamic testing. The idea is to instrument the program so that, in effect, it simulates its own cache behavior. Routines are linked in to provide hooks for a debugger to access

the simulated cache state and execution time. This approach eliminates direct probe effect at the cost of a slowdown and some inaccuracy.

Presentations on Architecture and Systems

Rajiv Gupta: Non-intrusive Monitoring. Rajiv Gupta of the University of Pittsburgh presented a simple way to insert monitoring code into real-time computations. The idea is to find time periods that are necessarily idle. Instrumentation code is inserted within these periods. If it is not possible to instrument all the program points of interest at once, multiple runs are performed with different parts instrumented. This talk provoked significant debate. Some participants advocated using such methods as aggressively as possible, selectively creating (or widening) idle times in order to insert instrumentation code. Others disagreed, arguing that the "temporary" delay slots will effect the behavior of other processes in the system. These researchers suggested that once instrumentation code is placed in a module, it should remain there for good.

Kevin Nilsen: Realtime Garbage Collection. Nilsen stressed the need to manage memory (via garbage collection or explicit allocation), and to control the time and space requirements of garbage collection. He described a hardware garbage collection memory manager (GCMM) device to redirect accesses to memory during garbage collection, while it is in the process of being moved. A major requirement of the hardware is that it not require special modifications to the processor. The GCMM supports a 1-2 microsecond latency time for allocate, deallocate, and access of memory, and has an average throughput 5 times higher than programs with explicit memory allocation (`malloc`).

Mohamed Younis: Speculative Execution. Timeliness via speculation has been used in hardware design (fast adders) and in real-time databases (speculative concurrency control). Younis argued that, in general, speculative execution can be used to speed up programs, but only where retraction of a speculatively executed code block would not cause a missed deadline. Source transformations may expose opportunities for speculative execution.

Wrapup

Following the technical presentations, a set of short "wrapup" talks were given by Rich Gerber, Rajiv Gupta, Vivek Nirke, Alex Vrchoticky, and Mohamed Younis. Their objective was to identify to how programming languages and compiler techniques can help ease the real-time design and implementation process. Major points are are included in the following section.

What came out of all of this?

It was generally agreed that several new areas should be pursued, and can be translated into hard results. They are:

A. Languages and System Support:

- Languages for multimedia, generics for graphics, etc. are taken very seriously by the industrial real-time community, but are relatively neglected by researchers. Yet in many ways the problems are more challenging than those in hard real-time, and they require sound programming language support and tools.
- Statistical real-time constraints: Most control systems over-sample to conservatively maintain the real-time constraints. Radical over-sampling doesn't scale up to support large systems,

nor is it desired in multi-media systems. Constraints should allow for statistical "softness" in sampling and update jitter. Requirements languages should be capable of incorporating such constraints; programming languages to support these; and flexible runtime systems to run the applications. Also, automated tools would be useful to analyze and prototype such application.

- There has been a proliferation of industrial real-time kernels, which are well-supported, lightweight, and incorporate sound real-time principles. To name three, QNX, VxWorks and LynxOS all support very low IPC latencies (around 10 microseconds), programmable timer-interrupt response within the same range, and priority-based scheduling. This being the case, there seems to be little further justification for universities or research labs to be in the realtime kernel-building business. Instead we can focus our efforts on what we do best: new design technologies, experimental applications, networks, etc., using off-the-shelf kernels and hardware platforms for our work.

- Object-oriented programming is here to stay. C++ has emerged as the leading production language and Ada9x is incorporating many OO structuring paradigms. This being the case, we shouldn't overlook these technologies in real-time systems. The challenges lie in supporting dynamic binding, garbage collection, etc.

B. Timing analysis:

- This is becoming a harder problem, but some vendors (like Sun) are modifying their architectures to accommodate real-time concerns. For example, the unpredictability inherent in Jump-and-link has led to a modification to the register windowing. (In future architectures, register spills will not trap to the kernel.) This change was made primarily for real-time applications, since Solaris is being marketed to them.

- Nonetheless, tight bounds on timing estimates are getting harder to achieve. Compiler techniques can help in two ways:

  1. By transforming the code to achieve better worst-case performance, accounting for caches, pipelines, etc.
  2. By helping obtain better coverage in profiling samples; i.e., by lowering the variance between different execution time readings on similar traces.

- No "pure" approach will suffice in timing analysis. In the design phase, we have to develop better statistical methods – or perhaps better learn how to use the ones we have, e.g., Markovian models, simulation, etc. When the code is written, static analysis can be used as a first pass – with certain refinements for caches, pipelines, etc. But nothing beats measuring a program's time on its (perhaps emulated) platform. For this we need better profiling/monitoring techniques, and superior ways of achieving reasonable coverage.

C. Program Transformation Techniques:

- Partial evaluation (and abstract interpretation) can produce better code, and is amenable to programs where variable definitions can be determined before schedule-time.

- Transformations such as loop-invariant code motion can radically improve real-time programs, especially in obtaining schedulability. But code cannot be optimized for average case behavior;

instead worst-case behaviors should be targeted. Thus generic back ends cannot be used, and should be customized for real-time applications. (The PLDI compiler folks had constructive ideas to accomplish this.) However, it was agreed that transformation techniques should not be carried out without the programmer's knowledge; instead, programmers should selectively apply them (perhaps using a graphical front-end). This would maintain traceability.

D. Tools for Design Methodology:

- It was generally agreed that this is perhaps the most important problem. Topics addressed were appropriate models and logics, analysis techniques, and means of achieving design refinement.
- A big problem is "tuning" the system's processes to achieve the end-to-end constraints. Compiler techniques can help here, by selectively 'cloning' shared servers and eliminating dead code.
- Any system is destined to fail it is poorly designed, or if its design is incorrectly translated into an implementation Thus a significant challenge is design refinement for mission-critical systems: Checking high-level, functional specs (perhaps via theorem provers or methods like Pugh's Omega Test); refining the spec into a more concrete system spec; checking its consistency via simulation, model-checking, etc.; developing the code with the aid of the abovementioned tools, so that it conforms to the system specs; tuning it via compiler-assisted transformation tools; and building clean and lightweight runtime systems to interact with the kernel intrinsics. This community seems to have a handle on many of these issues, and it may produce solid research toward the eventual goal.
- It was generally agreed that fully automatic technologies will not suffice to solve these problems. The key is to find the right paradigms for user interaction.

Future of the Workshop

The strong consensus was that SIGPLAN should continue the workshop next year; in fact, when Tom Marlowe called for volunteers for the organizing committee, several participants quickly stepped forward. Many of the "pure" industrial compiler folks expressed interest in submitting papers and participating on panels.

Industrial kernel people might profitably be recruited to attend, especially folks from QNX, Wind River, OSF and SUN (which is trying to put real-time functionality into Solaris). Finally, it is imperative to include several applications builders (i.e., "customers"), especially in control systems and interactive graphics.

For More Information

Postscript copies of some papers in the proceedings are available via anonymous FTP:

```
ftp.cs.umd.edu:/pub/faculty/pugh/sigplan_realtime_workshop_94
```

or via the World-Wide Web

```
http://www.cs.umd.edu/~pugh/sigplan_realtime_workshop_94
```