Supported in part by an award from Hewlett-Packard Corporation, from IBM Corporation, and a NYU research challenge grant.
CORPoreaT, a Constraint Based Environment for the Rapid Prototyping of Real-Time Programs. The report outlines the overall system architecture as well as the key modules in this environment that are being currently developed. CORPereaT is the first attempt to address the need for rapid prototyping of embedded applications. The report also discusses the design and implementation of the system and provides an overview of the current status of the project.

Abstract

CORPereaT: A Constraint Based Environment for the Rapid Prototyping of Real-Time Programs

Krishna V. Palm
Compilers for RISC processors.

The coarse-grained level where scheduling technology is being used routinely in product-quality architectures. In this direction at the coarse-grained level. The situation is even better at the
context of embedded systems. In the absence of time-constraints, major progress has been
made for automatically harnessing very high performance from these platforms; in the
tools for automatically harnessing very high performance from these platforms; in the

Our response to this challenge via CORREL is to develop scheduling methodologies and
overcoming this challenge.

Our focus is on construct parallelism at two levels of
compilers, each at a fine-grained level.

Our envision scheduler can — in addition to traditional control- and data-dependence

By the programmer.

Constraints in the source program — also cope with a variety of timing constraints specified

found, it will be fed automatically into a code-generator in the back-end of the compiler.
The resulting schedule for the instruction is referred to be feasible. If a feasible schedule is

instrumented with real-time constraints, are at its core. These algorithms are an integral

a scheduling-centric system in that a suite of algorithms for instruction scheduling programs
Programming environment for prototyping real-time programs on parallel platforms.

Tools that go into CORPET will naturally form an integral part of a full-fledged
and the back-end of the compiler to generate code. We envision that the algorithms and
and in shorter prototyping cycles, since identical schedules will be used by the analysis tools
during compilation. We anticipate this “confluence” in the scheduling algorithms to
process an which CORPET is based is independent of any particular target

The methodology on which CORPET is based is independent of any particular target
Courant Institute of Mathematical Sciences

Krishna Palen

Time Programs

Environoment for Rapid Prototyping Real

CORRETI: A CONSTRAINT BASED
1. Hard Real-Time: Avionics, medical life-support, ...

Target Domains for Real-time Applications

2. Soft Real-Time: Entertainment (set-top boxes, games), ...

Utilities (microwaves, wrist-watches), ...
Electroencephalographic data in a doctor's office

Building These Systems
When one of the buffers is full, switch input buffer and processing

• Process in the other.

• Collect data in one buffer.

The High Level Approach
Some Simple Time-Constraints

- The switching of buffers has to occur in the same interval.
- Processing software has to move ahead by one data point every 0.1 m-sec.
Other constraints involved interrupts from the board.

- Had to „hand-tune“ it to fit the constraints stated above.
- All the coding for the signal-processing had to be done in assembly.

The Difficulty
met, programs are returned till feasible.

If timing "expectations" — informally specified — are not
executed.

— actual measurement where code is produced and
— synthetic analysis, possibility with some tools, and

Estimate timing behavior via

Programmers spend a lot of time designing pieces of code.

Real-time Software Development in Industry
Revolution Spurred by the RISC Processor
Therefore, performance advantages can be harnessed automatically.

Optimizations feasible.

Clean simple model and Instruction Set makes compile time to harness promised performance.

High degree of Instruction Level Parallelism (ILP) via

Opportunity in Superscalars

A few notable exceptions of course:

- Register-Register Instructions (single cycle).
- Register-Register Instructions (multiple cycles).
- Load and store to/from memory (multiple cycles).

Canonicial Instruction set

Register Bank

Pipelined Functional Unit

Super scalar (RISC) Processors
Instructions being processed simultaneously.

At peak rates, with a 71.5 MHz clock, 357.5 MIPS with 9

are 1 deep.

Pipeline depth: Floating point unit is 3 deep, and the others
and 1 branch unit.

5 functional units: 2 fixed point units, 2 floating point units

Example Of Instruction Level Parallelism

Processor has
microprocessors—Video-pumps in the information highway.

• Hand-held games with 32-bit (embedded)

Makes much more ambitious real-time applications feasible.

• Substantially lower $ per MIPS.

Performance.

Rapid evolution of processor technology in terms of

The technological trend enabling new applications:

Returning To Real-Time Computing
context of modern processors with ILP.

Further compounded by the need to schedule instructions in the potential.

limits the scale of the explosion and does not harness hardwar

Hence substantially superlinear growth in software development cost as the scale grows.

Error-prone and tedious as the complexity of the system grows.

The Overall Challenge
- Leads to long prototyping cycles.
- Large gap between prediction and compiled code’s feasibility is aimed at compilation.
- Even with some automatic support.

An Important Gap
Temporal Behavior

To Programmer

FINEASIBLE

Program Graph Annotated

Coarse-grain Scheduler

Programmer To Code Generator

Feasible Schedules

Fine-grain Scheduler

Measurement and Constraints

Front End

The "Big Picture"
The Platform
Our Envisioned Response
Temporal Behavior

Scheduler

Programmer

Front End

Constraints + Program Graph

Annotated

INFEASIBLE

Feasible

Scheduler

To Code

Generates

To Code Generator

The Proposed Environment
• Programmers are slow to *migrate away* from their favorite and stable languages.

• Radical proposals will not succeed.

• Think of technologies that enhance existing languages in providing automatic support.

• Consequently, we do *not* propose a new language.
Generate code automatically if feasible.

Platform.

Analyze the program to determine feasibility on the target

Easily expressing "timining-relationships" between parts of the program.

Provide support for
The Programming Model

Coarse-grained Instruction Synchronization

Fine-grained Instruction Synchronization

Barrier
The Proposed Expressive Framework

- Constraint system
- Write timing relationships between markers are a distinct marker shown by thick lines.
- Enhance the user program with zero-time executable
3. \( (t)^s - (t)^f \geq 100 \)

2. \( p_{m} = 50 \)

1. \( \Sigma (t)^r p + (r)^s p + (r)^f p \geq 20 \)

Integers are m-sec — might be

Given that \( s \) and \( p \) respectively denote the start, finish and the durations

The Timing Relationship

ALARM

Markers
A Simple Example

Gather Data
Analyze
Diagnose

ALARM

Exception

Normal

Must be <= 30 msec

Cardiac Arrhythmia Detector
compilation and analysis cost.
Typically, abstract representations — expensive

Integer-linear constraints are rich in their expressive power.
Clean and minimal mechanism for expressing relationships.

Abstract representations are decoupled from program’s control flow.

Constraints As Time Specifications
Feasibility via Analysis

• A key observation makes the difference.

• In general, satisfiability of Integer Linear Programs — A hard problem.

• Constraint Satisfaction.
Scheduling Problems:
All the Integer Linear Programs are
Scheduling problems are extremely “well-solved”.

• Integer linear programs run very well.
• Fast algorithms produce very good solutions.
A number and types of processor in the target processor are specified.

Edge delays on the target processor are latencies on the edges.

Instructions are the nodes.

Constraints.

Data- and control-dependence encode precedence.

In a program dependence graph representation.

The Scheduling Model
real-time applications.

– absolute deadlines to model limited form of hard

RISC processor.

– completion time to improve performance of code on the

The goal is to minimize

• Acyclic control-flow structures.

Traditional Well-Solved Settings
The entire problem is thereby reduced to a well-defined
absolute variants that have been addressed.

- relative deadlines and start-times. As opposed to the
  - cycles in the graph,

Reduced to solving a Precedence-constrained Scheduling

The Core Research Question
Instruction Scheduling

Optimization

Combinatorial

Industrial Strength

Software

– Software engineering approaches
– Powerful technical methods and
  – Code generation (to be seen)
– Analysis and

A single well-parameterized combinatorial optimization question

Significance of Reduction
the two phases — tight design cycles.
Very good libraries are available in the field — very small gap between

Hence, the feasibility of compiled code is only and intimately dependent
generation phases.
Corresponds to a closer relationship between the analysis and the code

Same scheduling engine for code generation (as for analysis).
In addition to aiding in the rapid-prototyping of real-time processors in the embedded systems market, we are enabling the prototyping of RISC software.
Estimating Execution Times

- Coarse-grained Instruction Estimator
- Fine-grained Instruction Estimator

To coarse-grained scheduler

Time-estimate of Coarse-grained
Fine-grained Scheduling Algorithm Design and Evaluation

1998

1997

1996

1995

(Done)

Prototyping Framework and Scheduling Engine Definition

Develop Constraints Front-end and Integrate Scheduling Engine into Compiler, validate and Develop GUI (with N. Nachiappan)