POSIX/Ada Real-Time (PART) Project

October 7, 1991–December 31, 1992

Ted Baker

Department of Computer Science
Florida State University
Tallahassee, Florida

baker@nu.cs.fsu.edu

Research funded by the Ada Joint Program Office under the Ada Technology Insertion Program

Summary

PART Objectives

Ada binding to POSIX Realtime Extension

Ada binding to Threads Extension

POSIX Threads-based Ada runtime system

POSIX Threads implementation

Lessons learned

Examples of technical issues

Benchmark results

Products and Transition

Conclusions

Future work

POSIX

Source level application interface to OS services

Based on UNIX

General scheme - a base standard with extensions & profiles

- 1003.1 - Base POSIX standard: Processes, signals, I/O, etc.
  (an approved ISO/IEC and IEEE standard)
- 1003.4 - Realtime extensions
  (in IEEE ballot, Draft 13)
- 1003.4a - Threads extensions = Pthreads
  (in IEEE ballot, Draft 6)
- And a host of others

PART Objectives

Aid development of Ada bindings to POSIX

- 1003.4—Real-Time Extension
- 1003.4a—Threads Extension

Evaluate POSIX as a basis for Ada tasking

- Performance
- Portability

Language Bindings to POSIX

Source level interface implies the use of a language

- Language Independent Specification - LIS
- Language binding to the LIS

Currently defined in C

1003.5 - Ada binding to 1003.1
(an approved IEEE standard)

Ada binding for POSIX Real-Time Extensions

POSIX 1003.20, Ada binding to 1003.4

Developed by PART PI

Seven drafts (0.0–0.6) distributed as FSU technical reports

Drafts circulated to Ada bindings working group

PI serving as 1003.20 technical editor

IEEE *Mock Ballot* scheduled for January 1993

Ada binding to POSIX Threads

POSIX Threads = Pthreads = 1003.4a

Allows multithreaded processes

Threads are similar to tasks

Should threads be visible to an Ada application?

- As tasks?
- As something different?

Tasking-Based Binding

Consensus view of POSIX Ada binding working group

- Binding to threads is via tasks
- Conservative binding to other features: must be
    - Useful
    - Safe

Concensus partially based on results of PART project

First draft of Ada binding completed by PART project

Pthread-Based Ada Task Implementation

Questions addressed:

- Can threads be used to implement tasks?
- Can threads and tasks coexist?
- Is Ada tasking adequate access to threads functions?

To find out—Ada RTS implementation over Pthreads

Potential Disadvantages

Standard interfaces can reduce efficiency

Standard services may not match tasking needs

RTS use of POSIX operations may conflict with application uses

C library/Ada RTS interactions

Potential Advantages

Portability of Ada implementations

Simplification of Ada Runtime System (RTS) implementation

Mix-and-match compiler and OS support

Useful extensions to Ada tasking

Allows interoperability with C

Unix system calls block only the calling task

Multiprocessor threads → multiprocessor Ada

PART Ada RTS Design

Use no extensions to POSIX

One thread per task

Pthreads does all scheduling, dispatching

Multithreaded

Compatible with uniprocessor or multiprocessor

Use machine code inserts only where unavoidable

PART Ada RTS Status

Full Ada 83 semantics implemented

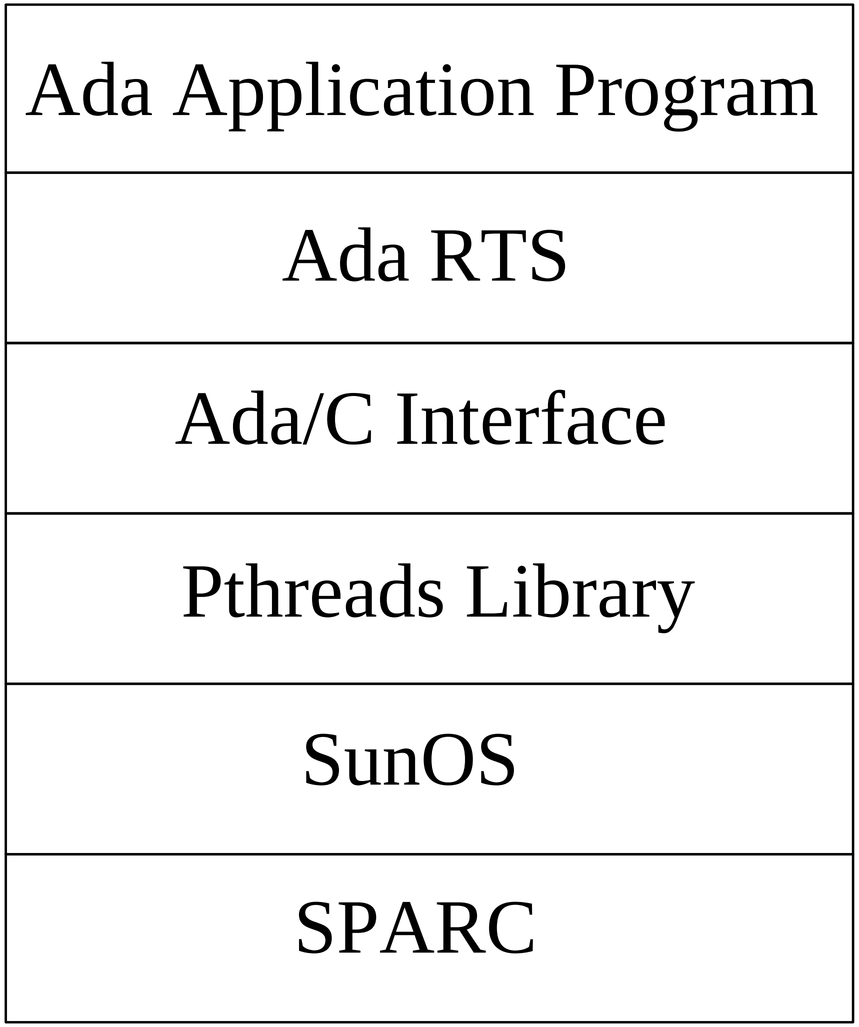
Supports Verdix SPARC compiler

Entirely new code

Passes Chapter 9 tests of ACVC 1.11 validation suite

Pthreads services from PART C library

RTS Implementation Overview



Ada Application Program

Ada RTS

Ada/C Interface

Pthreads Library

SunOS

SPARC

Library Implementation of Pthreads

Pthreads not available for SPARC in 1991

Pthreads subset library existed at FSU

Advantages of library *vs.* kernel implementation

- No kernel expertise needed
- Faster startup
- Potentially lower overhead
- More comparable to existing vendor Ada task support

Disadvantages

- Some OS calls block the entire process
- Process level scheduling takes precedence

Overview of the Pthreads Library

Full implementation of per-thread signal model

Monolithic monitor

Threads can be interrupted in most places

Threads can be preempted in most places

Implements some 1003.1 and 1003.4 functions

Read and write calls (of 1003.1) only block per-thread

Priority ceiling option not implemented

Lessons Learned

Pthreads can be used directly from Ada

Direct use from Ada is unsafe, for most Pthreads operations

Proposed Pthreads standard is very implementation-dependent

Reasonable (safe) implementation of tasks as Pthreads possible

Pthreads-based Ada RTS portable, *with limitations*

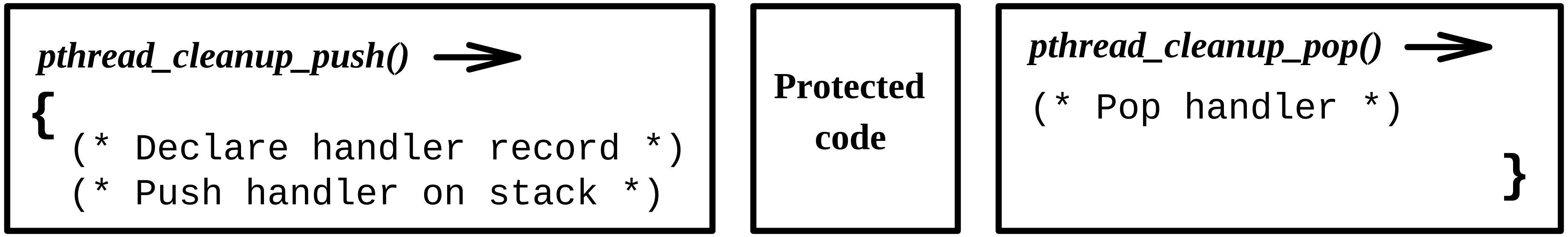
C/Ada interoperability possible, *with limitations*

Ada/C Interface Issues

PART project assumption—Pthreads operations are C functions

- Macros can be encapsulated in functions
- RTS access through pragma INTERFACE

Exceptions—*pthread_cleanup_push()* and *_pop()*



pthread_cleanup_push() →

```
{
  (* Declare handler record *)
  (* Push handler on stack *)
```

Protected code

pthread_cleanup_pop() →

```
(* Pop handler *)
}
```

Solution - Reimplement macros as C procedures

+ Macro definition must be visible—always possible
– Not portable
– Lexical rules of Pthreads cannot be enforced

Priority Inversion—How it Happens

Conflict between priority and synchronization

- High priority thread waits for low priority thread
- Compromises ability to guarantee real-time schedules

Conflict with Ada

- Without optional PIP/PCPE, mutexes allow priority inversion
- Multithreaded RTS uses mutexes
- Inversion may be visible to Ada application—*Illegal*
- Caused failure of validation test C9A006A
- Solutions reduce performance up to 2X

Not just Ada

- Mutexes in libraries cause inversion in applications
- Prediction requires internal library knowledge
- Violates information hiding principles

Priority Inversion—Solutions

Priority control protocols—Optional

- Priority Inheritance Protocol (PIP)
- Priority Ceiling Protocol Emulation (PCPE)

– Pthreads precludes most efficient implementations

RTS Priority Boost

- Sets all tasks in the RTS $\geq$ maximum priority

+ No application code can run while RTS mutexes held
– High overhead: 2–4 procedures calls needed to change priority
– Forces extra context switches

Interrupt Entries—Portability/Efficiency Tradeoff

1. RTS-defined tasks handle the signal and call accepting task
   - \+ Most “Pthreads-like” solution
   - \+ Most portable
   - – Inefficient—Tasks make heavyweight handlers
2. Accepting task waits for signal using *sigwait()*
   - \+ Intended to wait for signals
   - – Major change to rendezvous implementation
   - – Conflicts with mutexes
3. Accepting task waits for signal using *pthread_cond_wait()*
   - \+ Already used in rendezvous—Less impact
   - – Depends on a Pthreads “side effect”
   - – Conflicts with signal masks

Torture Testing—Perverse Scheduling

Multithreaded RTS—for multiprocessor Pthreads

High priority threads can't depend on precluding low priority ones

Debugging on a uniprocessor—Perverse scheduling

- Unpredictable thread interleaving
- Low priority threads preempt high priority ones
- Equal priority threads switched without blocking

Compile-time options in Pthreads to modify scheduling

- MUT_SWITCH—Switch contexts when a mutex is locked
- RR_SWITCH—Round Robbin context switch
- RAND_SWITCH—Random context switch

PIWG Benchmarks

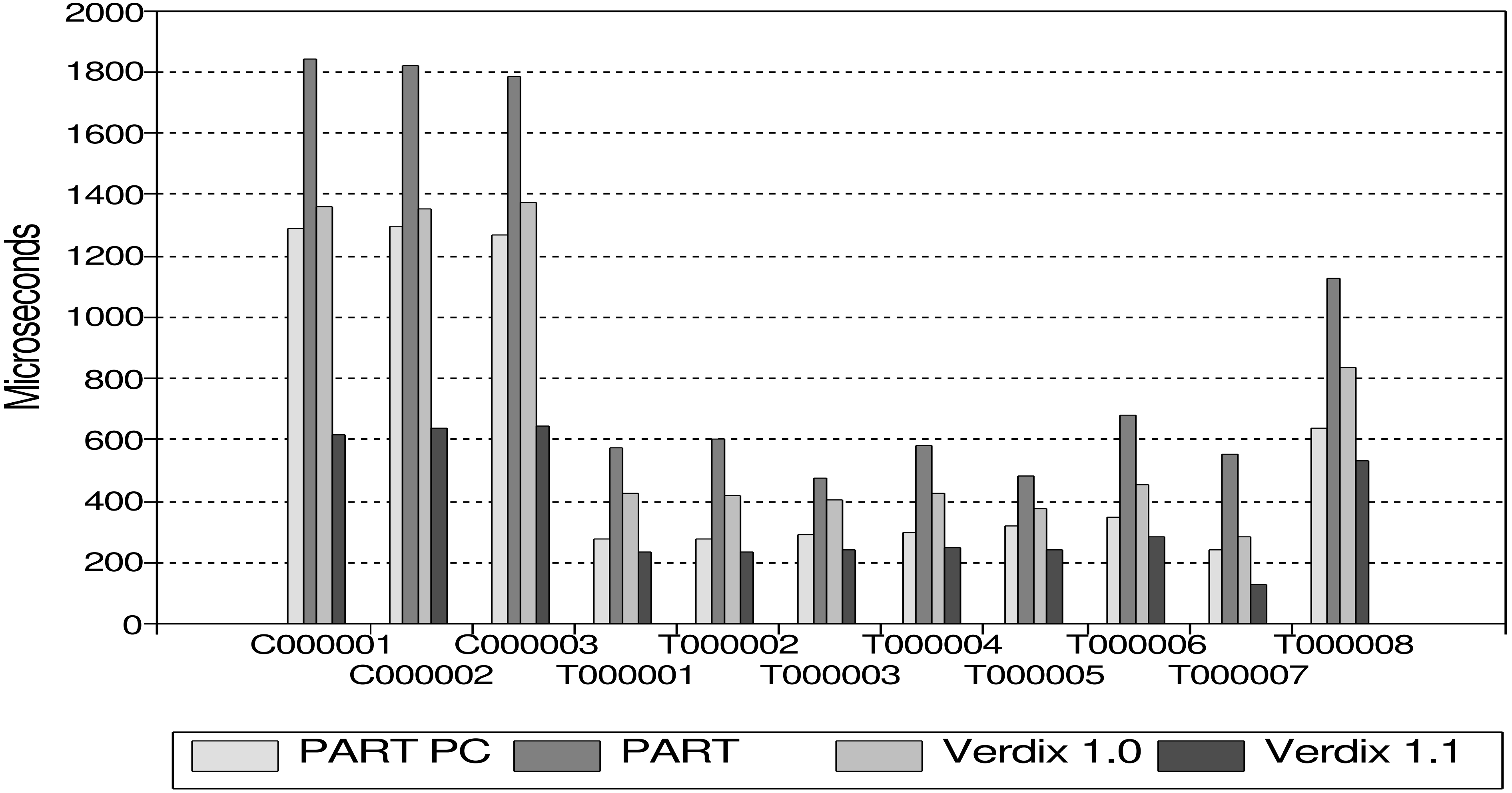
PIWG benchmarks for task creation/termination, rendezvous

- Fairly simplistic
- Used to judge cost of Pthreads interface
- Used to tune RTS, Pthreads

Caveats

- Quiescent system, no time slicing, -O9 optimization
- Sun SPARCstation SLC, 25 MHz
- Verdix 1.1 data provided by Verdix
- PART optimizations could be applied to Verdix 1.0
- PCPE simulation—Priority change doesn't force switch

PIWG Benchmark Results



Transition—PART products

Binding to POSIX 1003.4 (1003.20)

Ada RTS implementation over Pthreads

Binding to POSIX 1003.4a (Draft)

PART Quarterly Technical Reports

Publications

Pthreads library implementation

Transition—Availability of PART products

Product	Source
Binding to 1003.4	IEEE
Ada RTS implementation	ftp*
Binding to 1003.4a	FSU
Quarterly Reports	CECOM
Publications	proceedings, ftp
Pthreads library	ftp

PART ftp site—ftp.cs.fsu.edu (*128.186.121.27*): /pub/PART

*form of Ada RTS release not yet determined

Transition—PART Project Publications

Using POSIX Threads to Implement Ada Tasking (TRI-Ada)

Implementing POSIX Threads under UNIX (SERF)

A Library Implementation of PThreads under UNIX (USENIX)

POSIX/Ada Real-Time Bindings (WAdaS)

Draft bindings to 1003.4 and 1003.4a

Transition—Organizations Requesting 1003.20 Drafts

Honeywell

Harris

Convex

Boeing

CDC

IDA

Transition—Requests for the Pthreads Library

Lynx

Intrepid (Silicon Graphics)

Advanced Software Concepts

Alcatel Bell Telephone Research Center

ADALOG

CompuServe Incorporated

NYU GNAT project

Conclusions

Tasks can be mapped to threads

Tasks are the preferred binding to Pthreads

- hides some Pthread implementation dependences
- safer

Ada RTS portability possible, within limits

Mismatch of Pthreads to Ada requirements costs performance

Cost likely to be tolerable (less than expected)

Advantages in reduced blocking and multiprocessor portability

Open Issues

Ada 9X

- Additional features may improve binding to C standards
- Current tasking design will need update to support Ada 9X

Ada RTS only tested with PART Pthreads library

- Kernel implementation may be slower
- Portability of RTS between threads platforms untested

Future Work Needed

Develop Ada 9X bindings

Implement new Ada 9X tasking features with Pthreads

Catalog Pthreads implementation dependences

Adapt PART runtime system for GNAT

Test portability of PART RTS to new Pthreads implementations

- Solaris 2.1 threads
- LynxOS threads