



HPEC 2006: Looking Forward, Looking Back...

The Impact of Programming Difficulty on Hardware Obsolescence

Gedae, Inc.

www.gedae.com

856 - 231 - 4458

Programming Methodology Then and Now



1990s: SHARC DSP Boards

- N boards with M processors per board
- Program SHARC at low level, tied to architecture
- Thoroughly preplan interaction between processors

Now: Multicore and FPGA

- N processors with M cores per processor
- Program cores & FPGA at low level, tied to architecture
- Thoroughly preplan interaction between cores

Programming Difficulty Lessened SHARC's Success



1990s: SHARC DSP Boards

- Middleware hides DSP complexities but does not tap its full power
- Difficult to program DSP at a low level
- Long schedules
- High costs
- Difficult to maintain

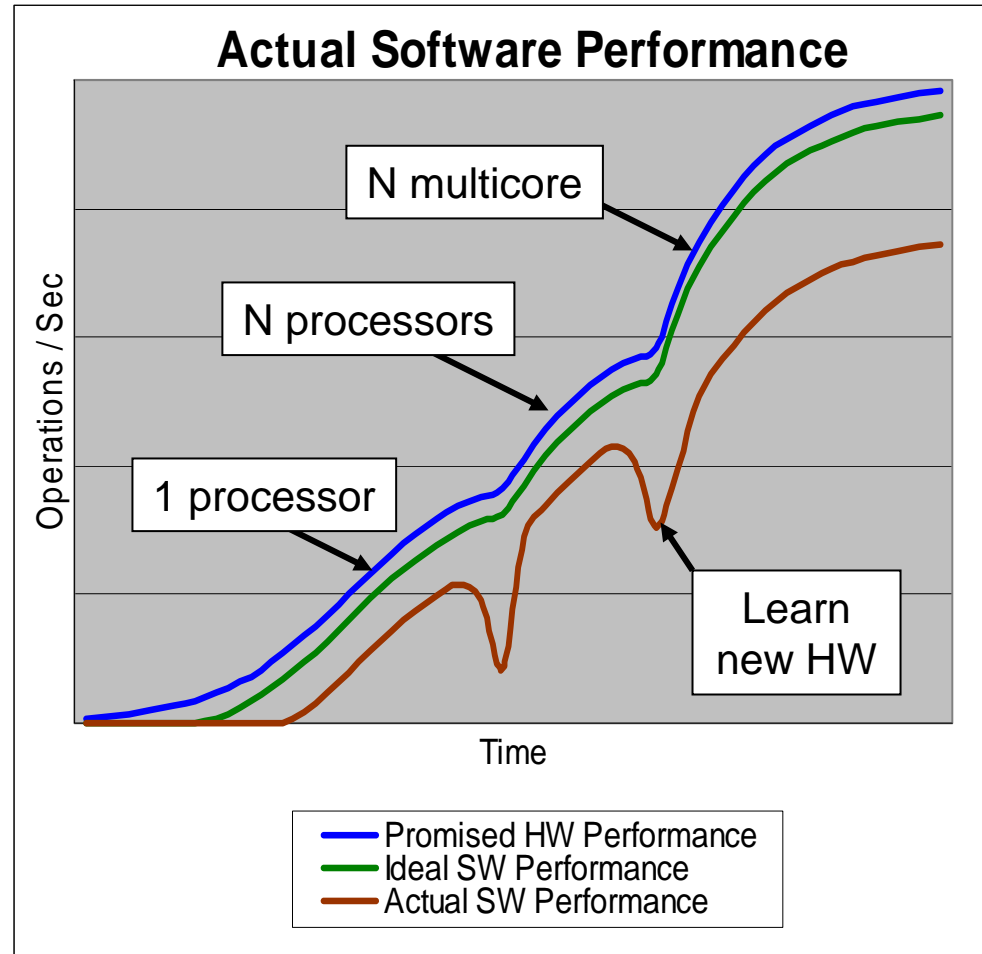
Now: Multicore and FPGA

**Are We
Repeating
the Same
Problems?**

Programming Difficulty Affects Performance



- Each new HW technology requires relearning how to program and recoding of algorithms
- Will the promise of new hardware be fully realized if software development lags behind?



Need a Solution to Programming Difficulty

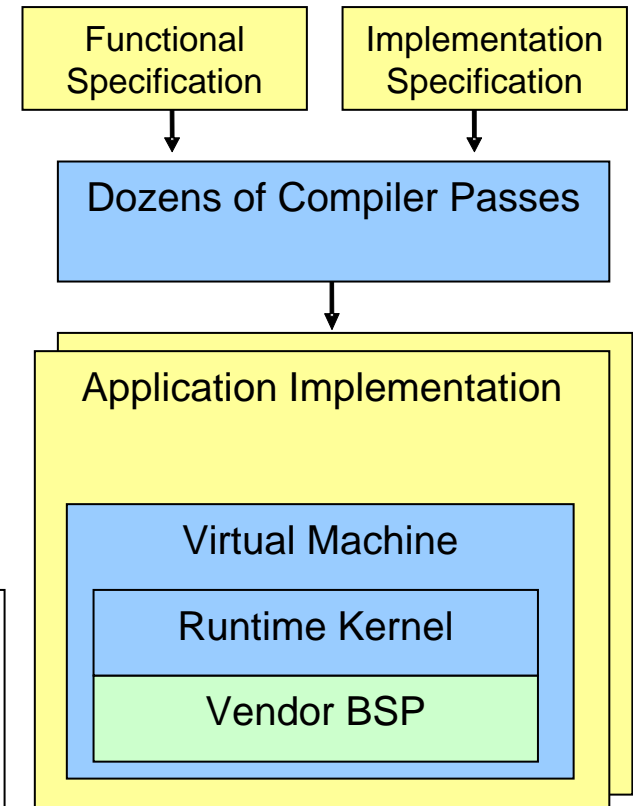
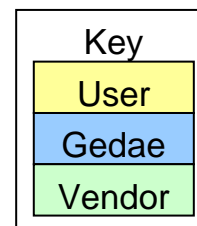


- 13.1% of all designs started are canceled
- 54% of software is completed behind schedule
- Need programming tools that alleviate this risk while not sacrificing performance and providing portability to future systems
- Checklist of tool feature/benefit metrics
 - Ease of Implementation
 - Strength of Implementation
 - Portability Between Processor Types
 - Visibility of Implementation
 - Visibility of Execution
 - Portability to Multiple Processors
 - Ease of Deployment
 - Ease of Supporting New Target Hardware
 - Breadth of Functionality

The Gedae Solution: Powerful but Elegant Language



- Specification of entire application
 - From high level (mode control) to low level (vector operations)
 - From data input (A-to-D) to display (GUIs)
- Functionality developed separately from target-specific implementation
 - Simplifies implementation process
 - IP not tied to current target hardware

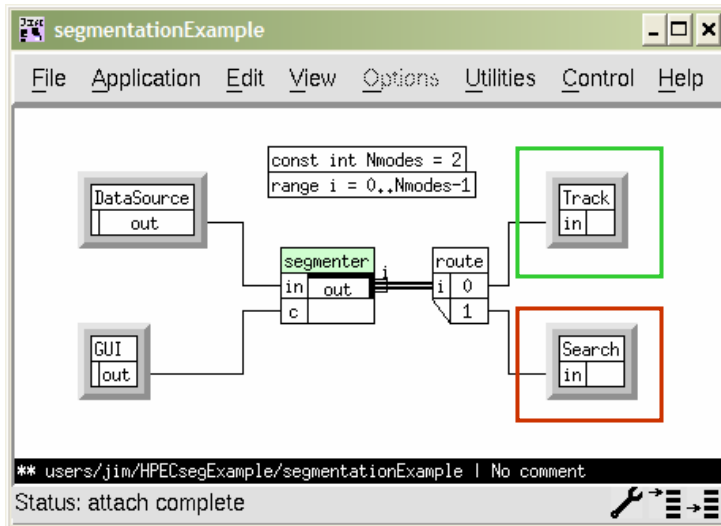


The Gedae Solution: Deployable Real Time Products

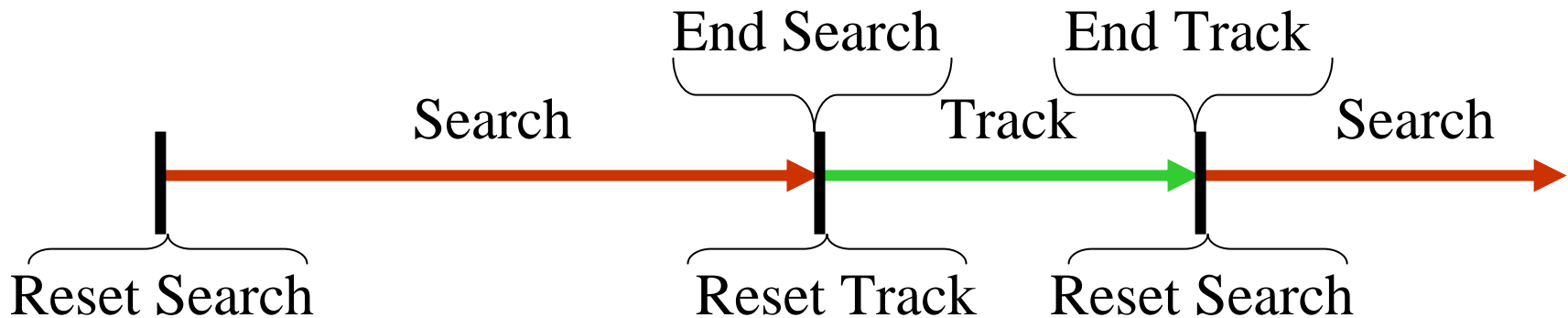


- Quality autocoding
 - Creates real time applications
 - Provides many optimizations that are difficult when hand-coding
 - Integrated with optimized vector library and optimizing cross compilers
- Implements the complete product
 - Targets heterogeneous multiprocessor systems including DSPs, Multicores, FPGAs, ADCs, Clusters, etc.
 - All interaction between components is autocoded
 - No costly code integration after code is generated

Language Simplifies Distributed Mode Control



- Break infinite streams into segments
- Segment markers cause old mode to end and new one to reset
- Exclusivity allows memory sharing between modes



Simplifies Creating Target-Specific Multiprocessor Implementation



- Create a multiprocessor implementation by setting parameters in the Group Control

Partition and Subschedule

Map to Processors

Set Data Transfer Methods

Set Static Schedule Properties (Tasks)

Set Granularity and Priority

Set Queue Size and Properties

Simplifies Debugging Complex Multiprocessor Issues



- Model provides visibility for powerful analysis

The screenshot displays the Gedae software interface for debugging a data flow deadlock. It is divided into several key sections:

- Implementation:** A data flow graph showing nodes like `brownian`, `cmplx`, `x_dcopy`, `x_rdiv`, `x_mag_1`, `brownian_1`, `x_mag`, and `runningAvg`. A **Blocked queue** is highlighted on the `x_rdiv` node, and a **Starved queue** is highlighted on the `runningAvg` node.
- Execution details:** A Gantt chart showing the execution timeline of various schedules (Schedule 1 and Schedule 2) for different components, with time markers at 0.019956 s and 0.019983 s.
- Functionality:** A block diagram of the application with explanatory text:
 - "Inserts queue for illustration purposes."
 - "Running average box artificially drops an occasional token to illustration data flow deadlock."
- Deadlocked Loop Analysis:** A dialog box showing the state of the deadlock:

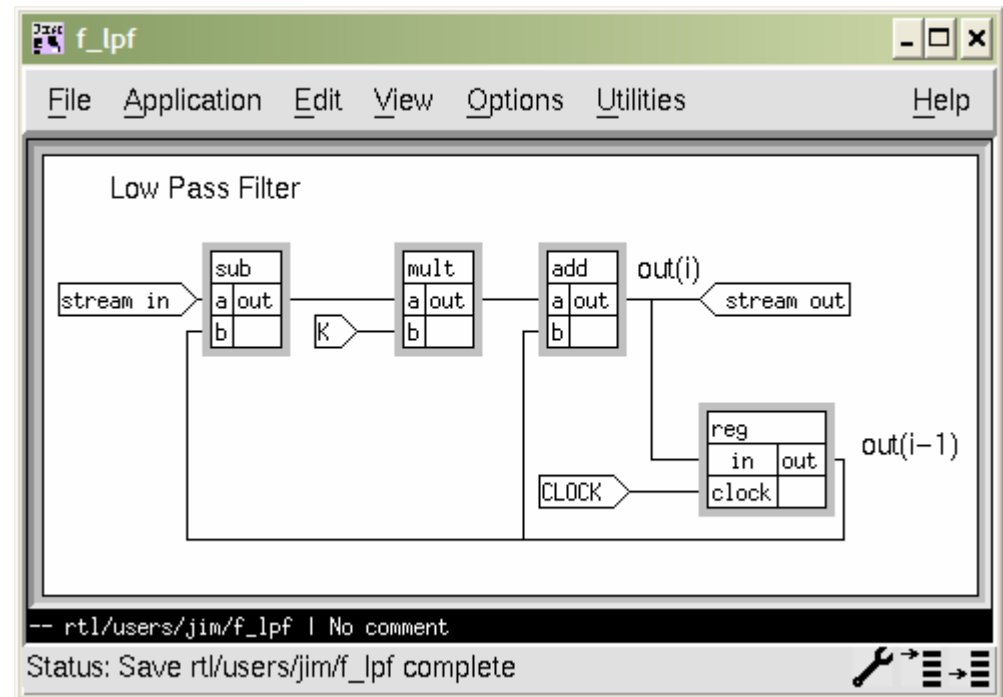
----- Deadlocked Loop:	
Blocked Schedule 1	(1,0,0,0)
2: x_rdiv<a	(1,0,1,1,0,1)
Starved Schedule 2	(0,0,0,1)
1: x_rdiv<b	(0,0,0,1,1,0)

Note:
Each blocked schedule in deadlock loop is followed by blocked outputs.
Starved schedules are followed by starved inputs

Language Provides for Heterogeneity



- Signal Flow Graph (SFG) language
 - Primitives can be coded in either C or SFG
 - Fully language-independent specification of functionality
- Language Support Package (LSP)
 - Export C code for mapping to host or DSP
 - Export VHDL code for mapping to FPGA

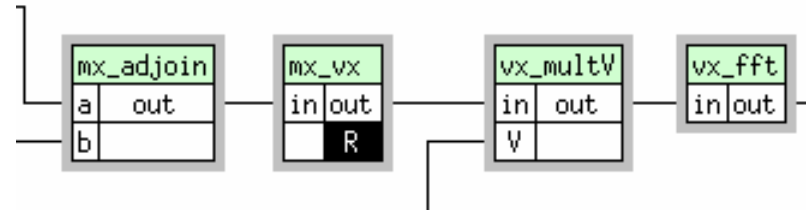


$$\text{out}(i) = K * (\text{in}(i) - \text{out}(i-1)) + \text{out}(i-1)$$

Automates Optimizations That are Difficult When Hand-Coding

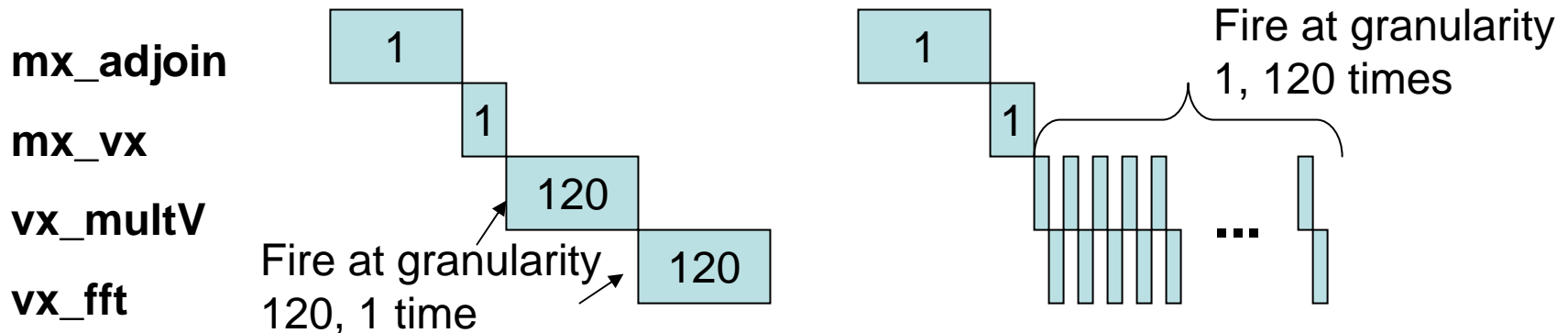


- Example: Automatic strip-mining
- Decrease overall memory usage
- Improve use of fast memory (cache)



Processing chain breaks large matrices into 120 vectors

Static Schedule Timeline Subscheduled Timeline



IP Not Tied to Target Hardware

- The method for generating the implementation is the same for all target hardware
- Board Support Package can be quickly filled-in for each new target processor
- Functionality can be easily remapped to next generation of hardware
- Less risk in maintenance

