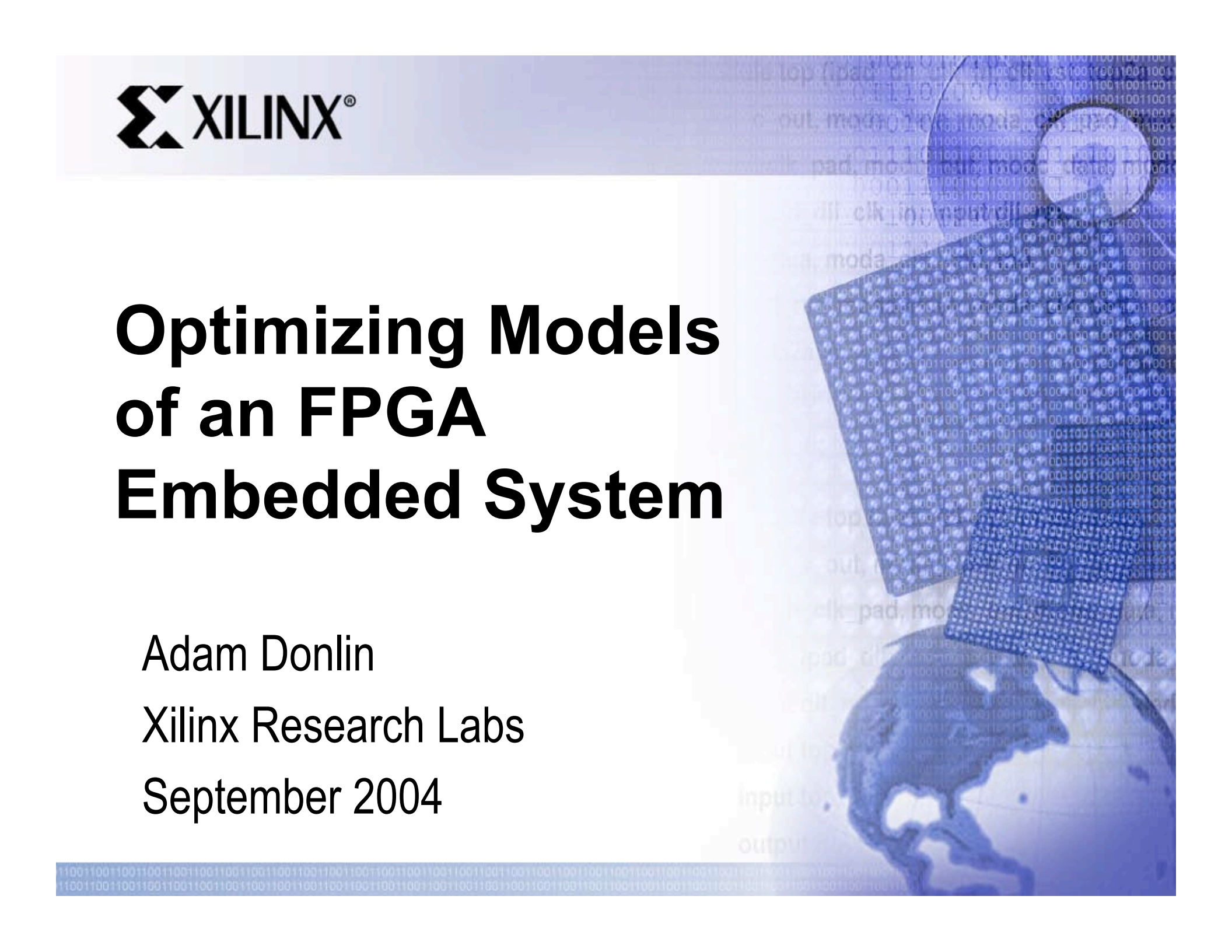


Optimizing Models of an FPGA Embedded System

Adam Donlin

Xilinx Research Labs

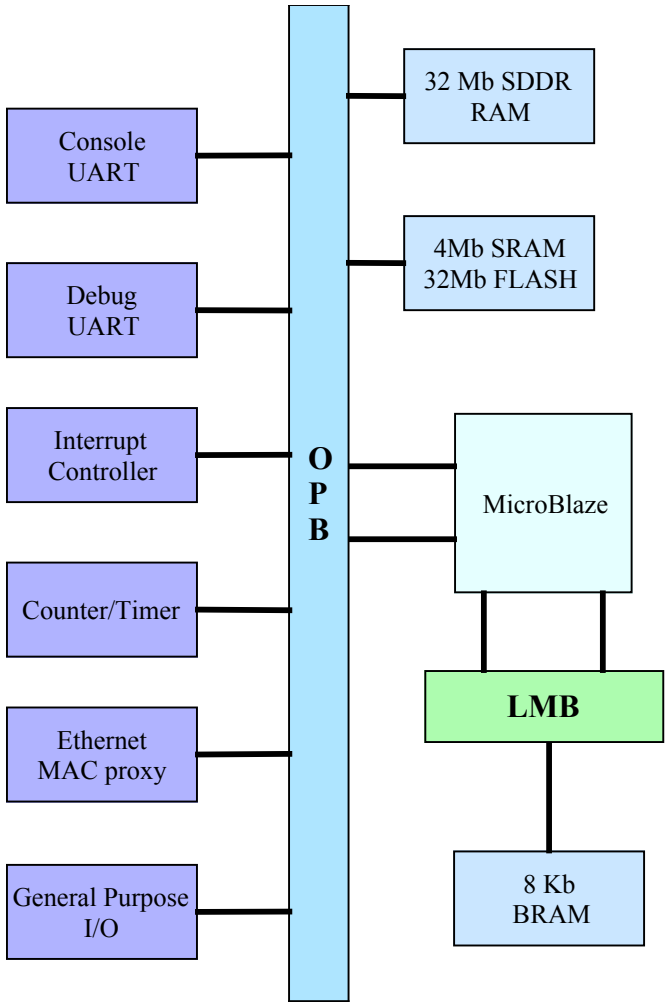
September 2004

Outline

- Target System Architecture
- Model Optimizations and Simulation Impact
  - Port Datatypes
  - Threads and Methods
  - Port Access
  - Reduce Scheduling
  - Direct Instruction Memory
  - Direct Data Memory
  - Reduce Scheduling II
  - Intercept Kernel Functions

Target System

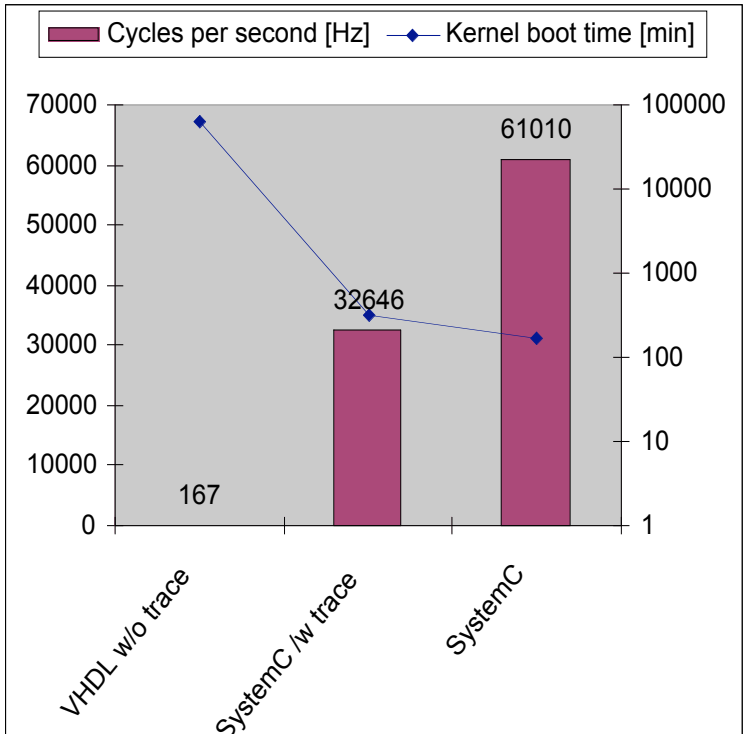
- Third party ‘Golden’ design and uCLinux binary
  - John Williams, University of Queensland, Australia
- Native C MicroBlaze ISS wrapped into a SystemC module
- Model’s Host Machine
  - Linux 2.4 Kernel
  - 3 GHz Xeon Processor
  - Negligible Load



Initial Model

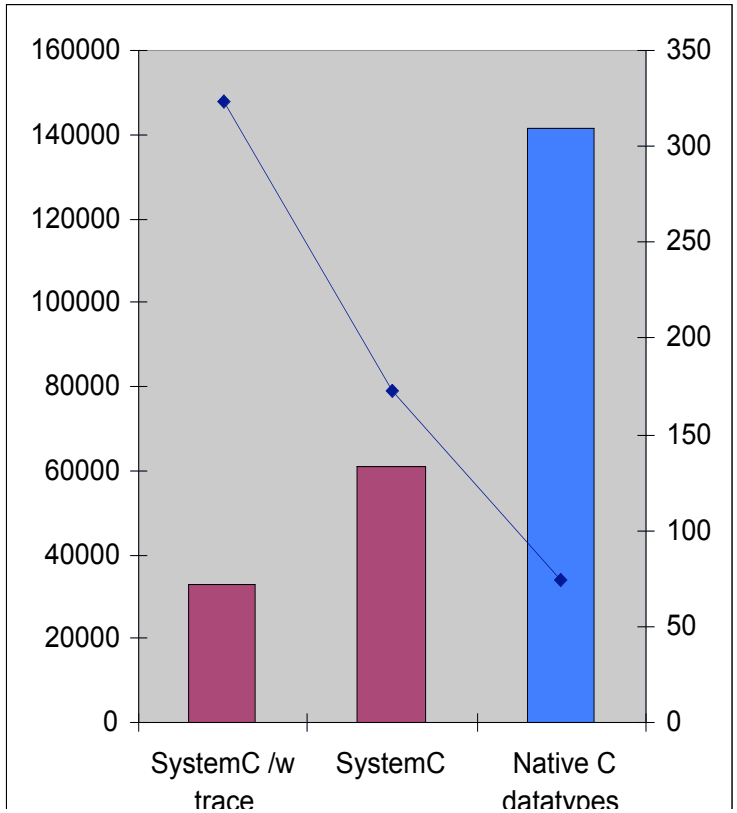
- Full target system model in SystemC
- Cycle Accurate
- Datatypes used:
  - SystemC 4-value logic
  - Boolean

- **Kernel Boot Time**
  - RTL 1 mon 15 days 22 h w/o trace
  - SystemC with trace: 5h 23min
  - SystemC without trace: 2h 52min



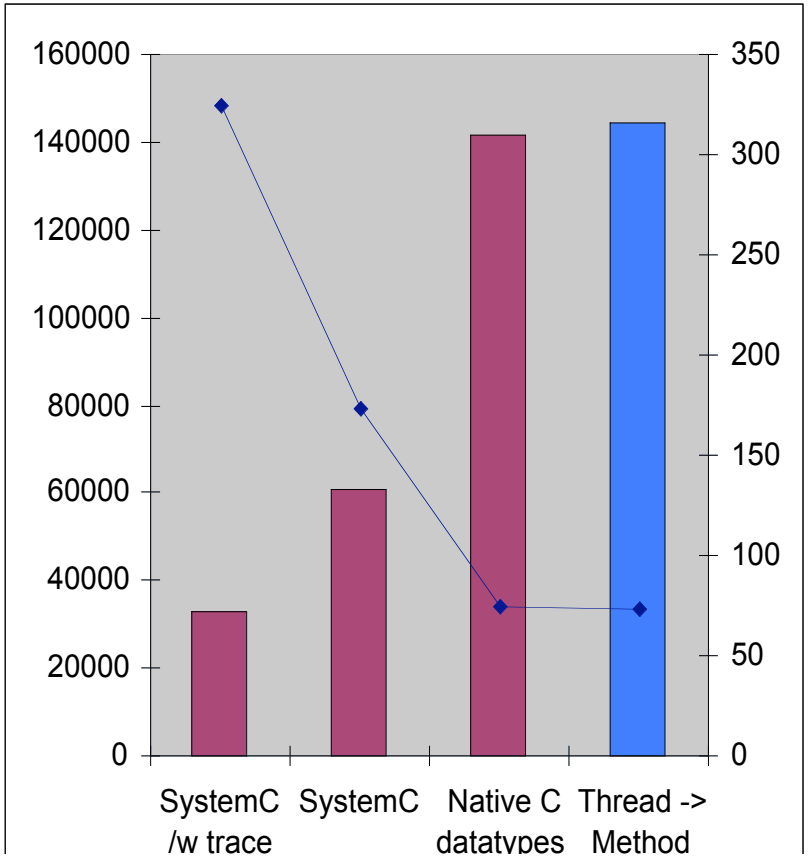
Optimization 1: Native C-types

- SC_RV types are too heavy to use in simulation.
  - Replace with Native C-types (int, char)
  - much lighter to execute
- Easy remapping of port types:
  - convenience macros
  - accessor functions
- Tradeoff :
  - Giving up ModelSim co-simulation
  - Detection of multiple drivers/ uninitialized logic
- **Kernel Boot time: 1h 14 min**



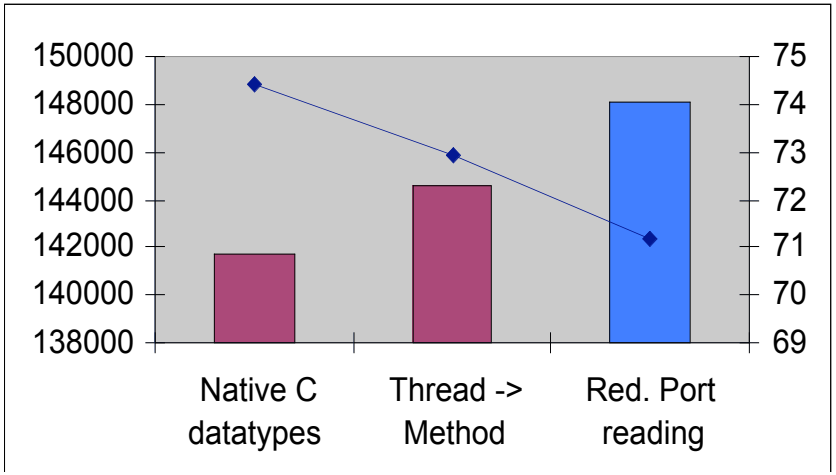
Optimization 2: Threads to Methods

- SC_METHOD is simpler process type than SC_THREAD
- Convert functionality currently modeled as SC_THREAD to SC_METHOD
- Tradeoff:
  - Appx. 30 kHz benefit
  - But multi-cycle processes less elegant…
- **Kernel Boot time: 1h 12min**



Optimization 3: Reduce Port Manipulation

- Manipulating ports can involve multiple levels of function call.
- Minimize the number of port object interactions
- **Kernel Boot time: 1h 11 min**




```
void thread1() {
   while(1) {
     if(x.read()==2) {
        z = x.read()+y.read();
     }
     wait();
  }
}
```

```
void thread1() {
   while(1) {
      uint localX = x.read();
      if(localX == 2) {
         z = localX+y.read();
      }
      wait();
...
```

Optimization 4: Reduce Scheduling

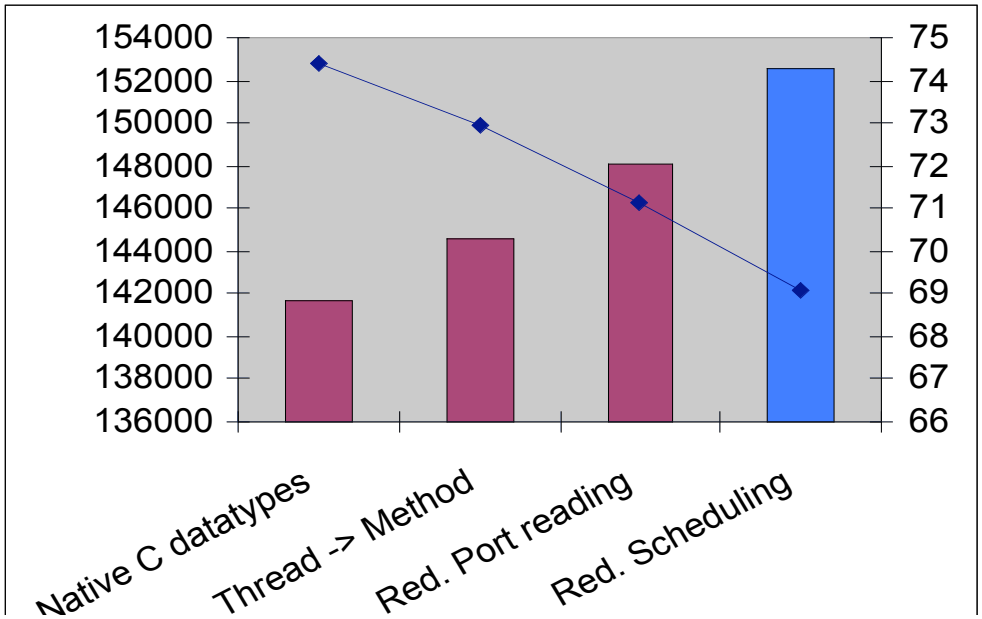
```
void method1() {
   while(1) {
      z = x+y;
      wait();
}}
```

```
void method2() {
   while(1) {
       answer = 42;
       wait();
}}
```

```
void combimethod() {
       funct1();
       funct2();
}
void funct1() {
   z = x+y; }
void funct2() {
   answer = 42; }
```

- Technique 1:
  - Combine multiple threads/methods in one container
- Tradeoff:
  - can be applied only single-clock cycle processes

Optimization 4: Reduce Scheduling

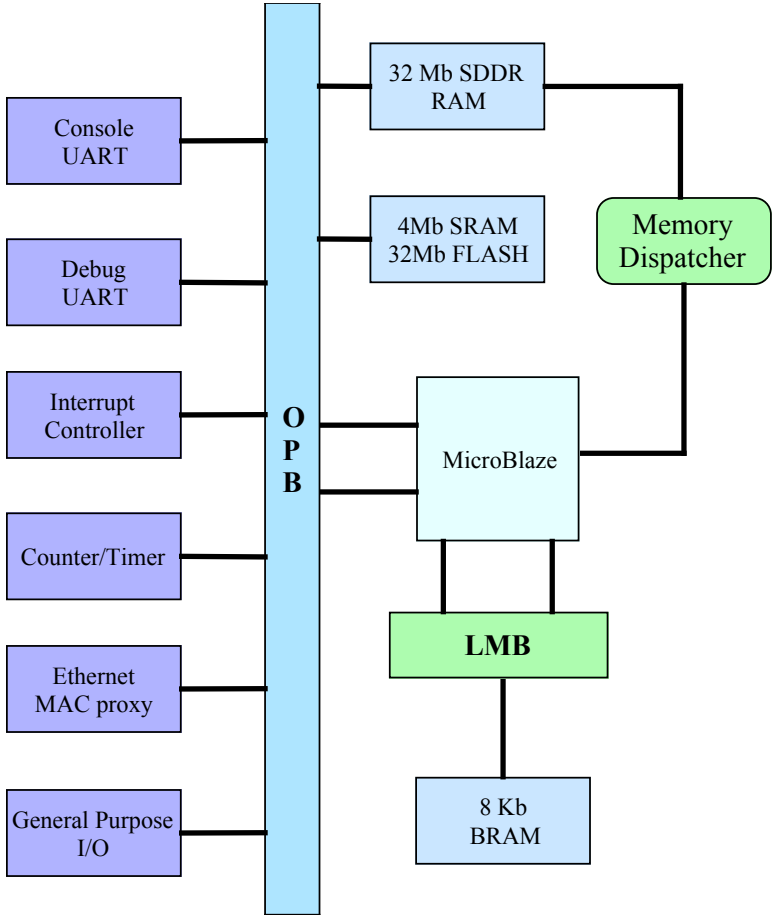


- Technique 2:
  - Multi-cycle scheduling delays for compatible threads
- **Kernel Boot time: 1h 9 min**

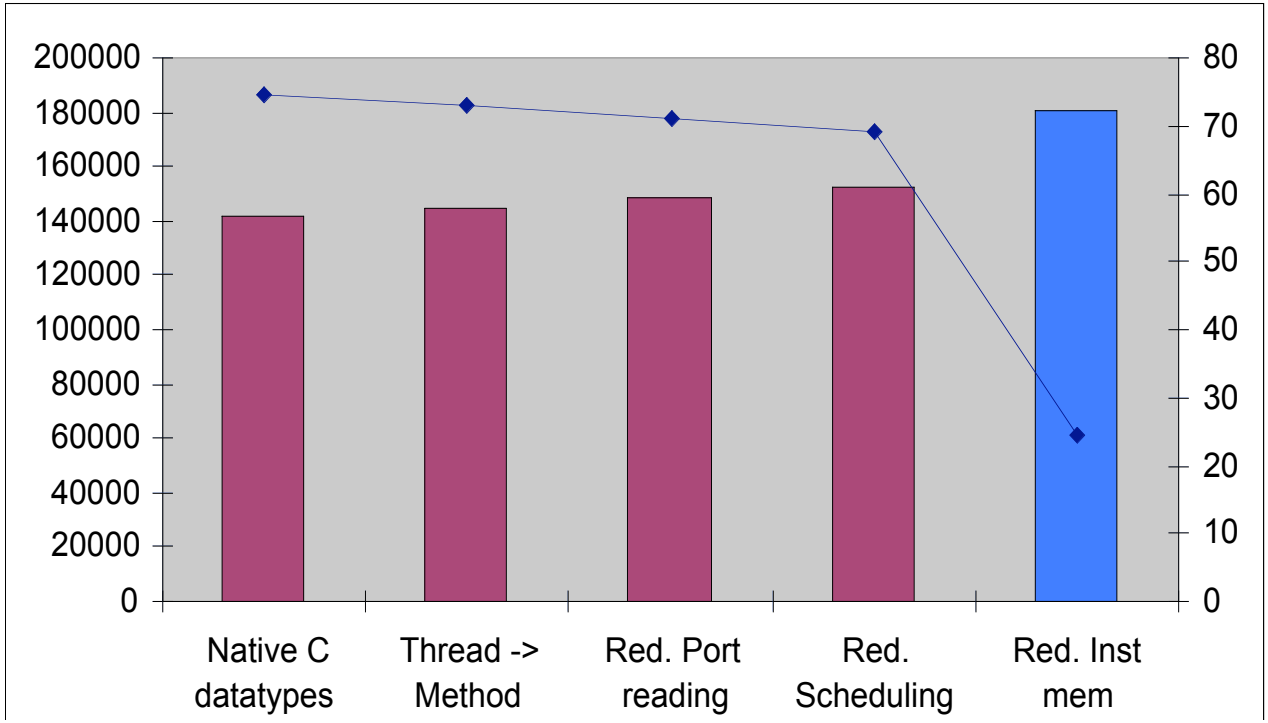
```
void UART_interface_method()
{
    while(1) {
          do_UART(); // system calls
          // wait 100 clock cycles
          wait(1,SC_US);
    }
}
```

Optimization 5:
Instruction Memory Suppression

- Vast majority of memory activity is instruction fetches over the OPB
- MicroBlaze SystemC wrapper captures instruction fetches
- Provides instructions in a single cycle
- Tradeoff:
  - lose cycle accuracy
  - Can be turned on and off at run-time

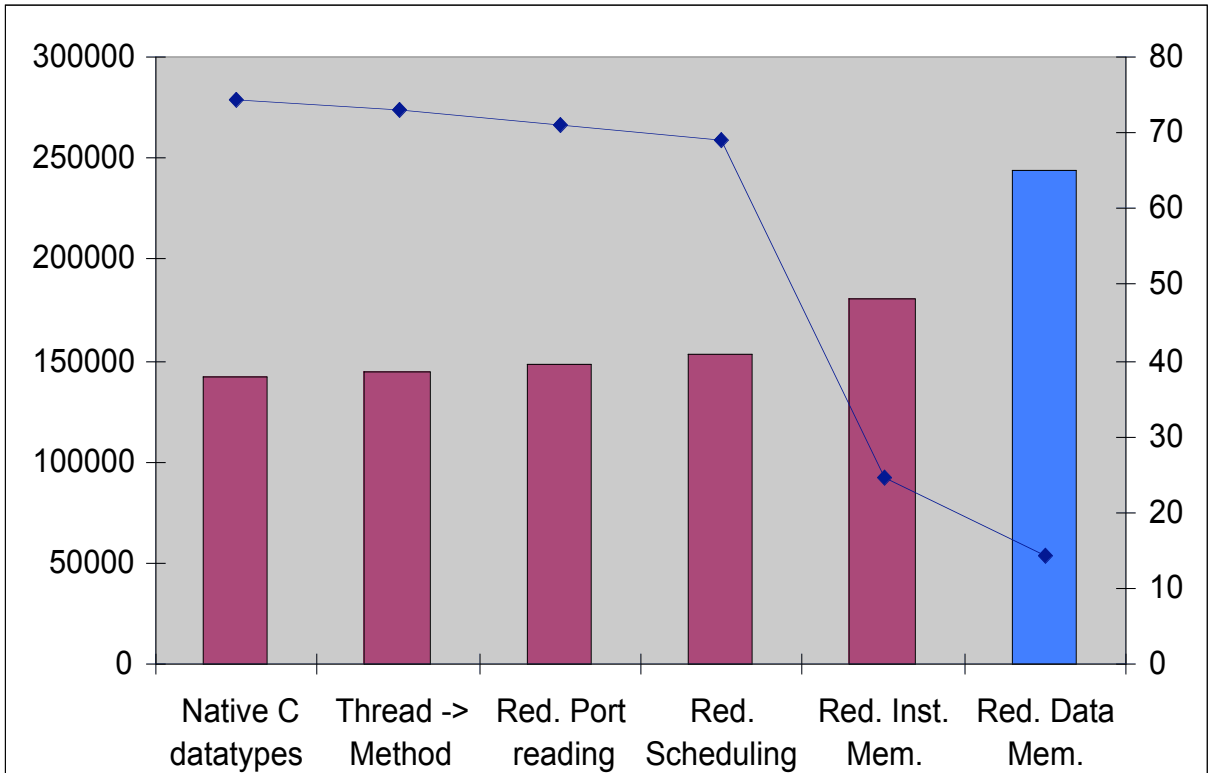


Optimization 5:
Instruction Memory Suppression



- Main speedup: less cycles per instruction
- **Kernel Boot time: 24 min 33 sec**

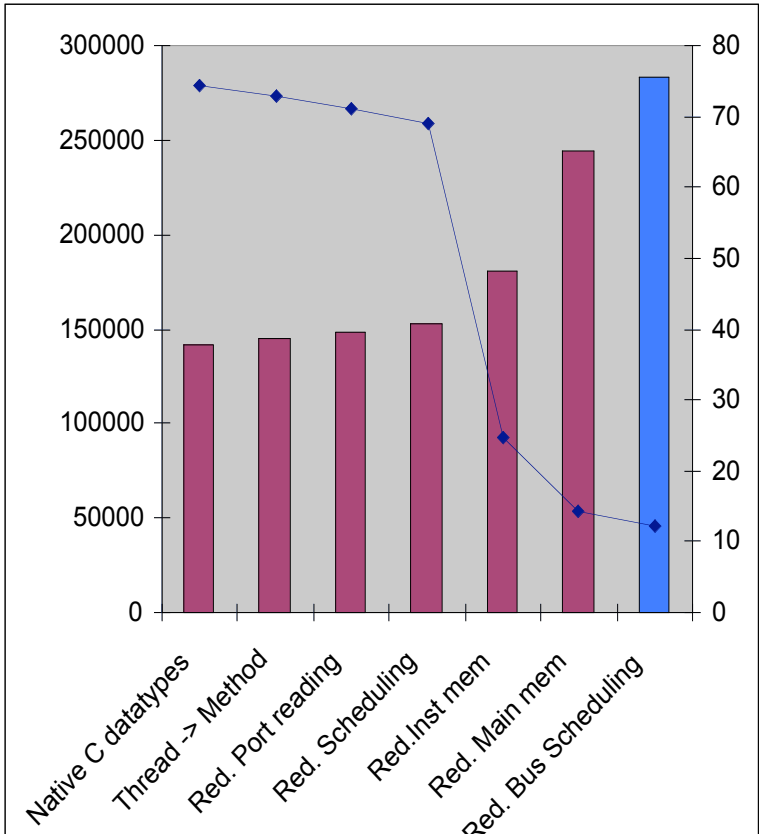
Optimization 6:
Data Memory Suppression



- Previous optimization applied also to data bus transactions
- **Kernel Boot time: 14 min 17 sec**

Optimization 7:
Reduced Bus/Slave Scheduling

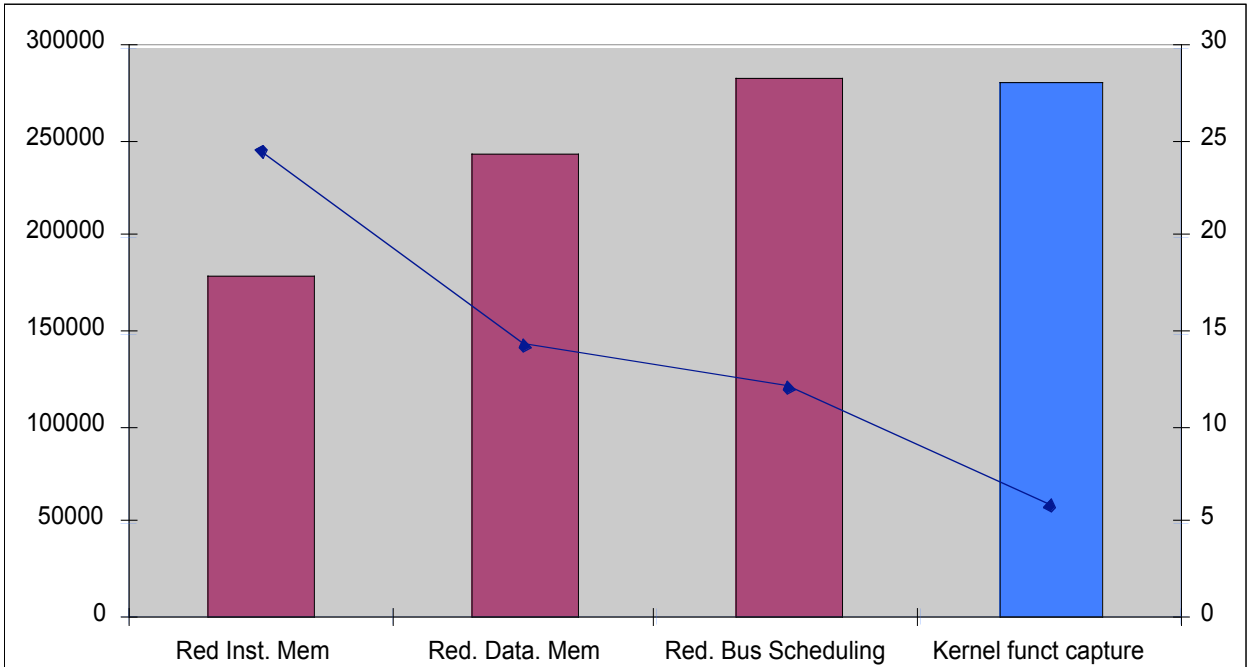
- Many peripherals are used infrequently
  - Flash, GPIO, Ethernet
- However, OPB slave interface decodes OPB bus every clock cycle
  - Schedule only when required
- Kernel Boot time: 12 min 4s



Optimization 8:
Intercept Kernel Function Calls

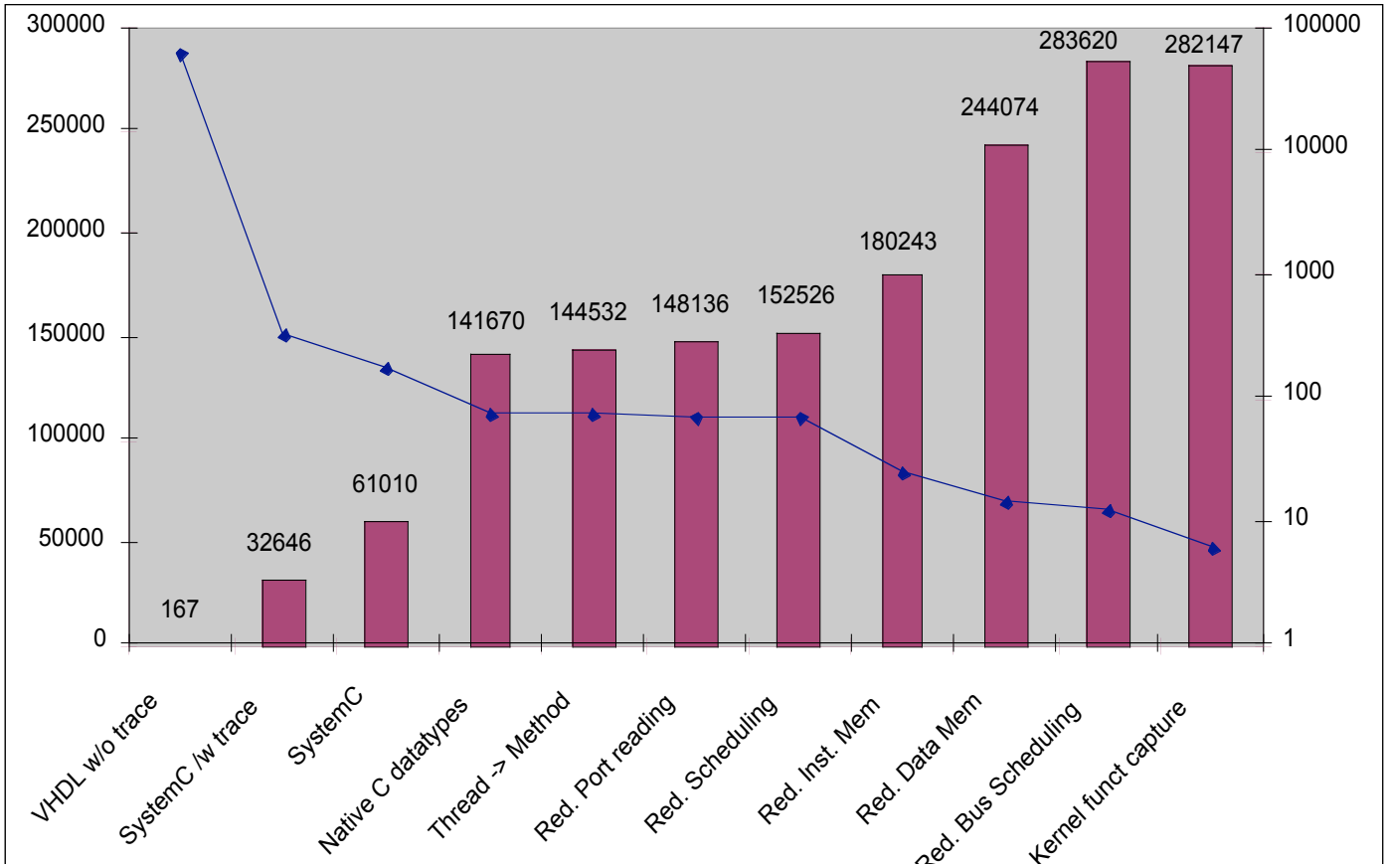
- Kernel boot profile:
  - 52% execution in two libc functions: *memset* and *memcpy*
- Tradeoff:
  - application specific benefit,
  - requires careful programming,
  - lose ISS instruction count accuracy
  - Can be turned on and off at run-time
- ISS Wrapper Intercepts and Implements with Native Functions:
  - Detects jump to a function
  - Reads function parameters from MB registers
  - Executes function in native C
  - Writes MB registers to values in normal execution
  - Make loop guard ‘always true’

Optimization 8:
Kernel Function Capture



- Cycle time is lower,
  - **but** execution time faster due to skipped instructions
- **Kernel boot time: 5 min 56 sec**

The Whole Sequence



- RTL Boot Time: ~1.5 Months
- Initial SystemC Boot Time: ~5.5 Hours
- Final Boot Time: ~6 Minutes
- Simulation rate: ~30kHz to 280 kHz