Developing Custom Firmware for the Red Storm SeaStar Network Interface

May 18, 2005

Kevin Pedretti
Sandia National Laboratories
ktpedre@sandia.gov

Trammell Hudson
OS Research
hudson+cug@osresearch.net

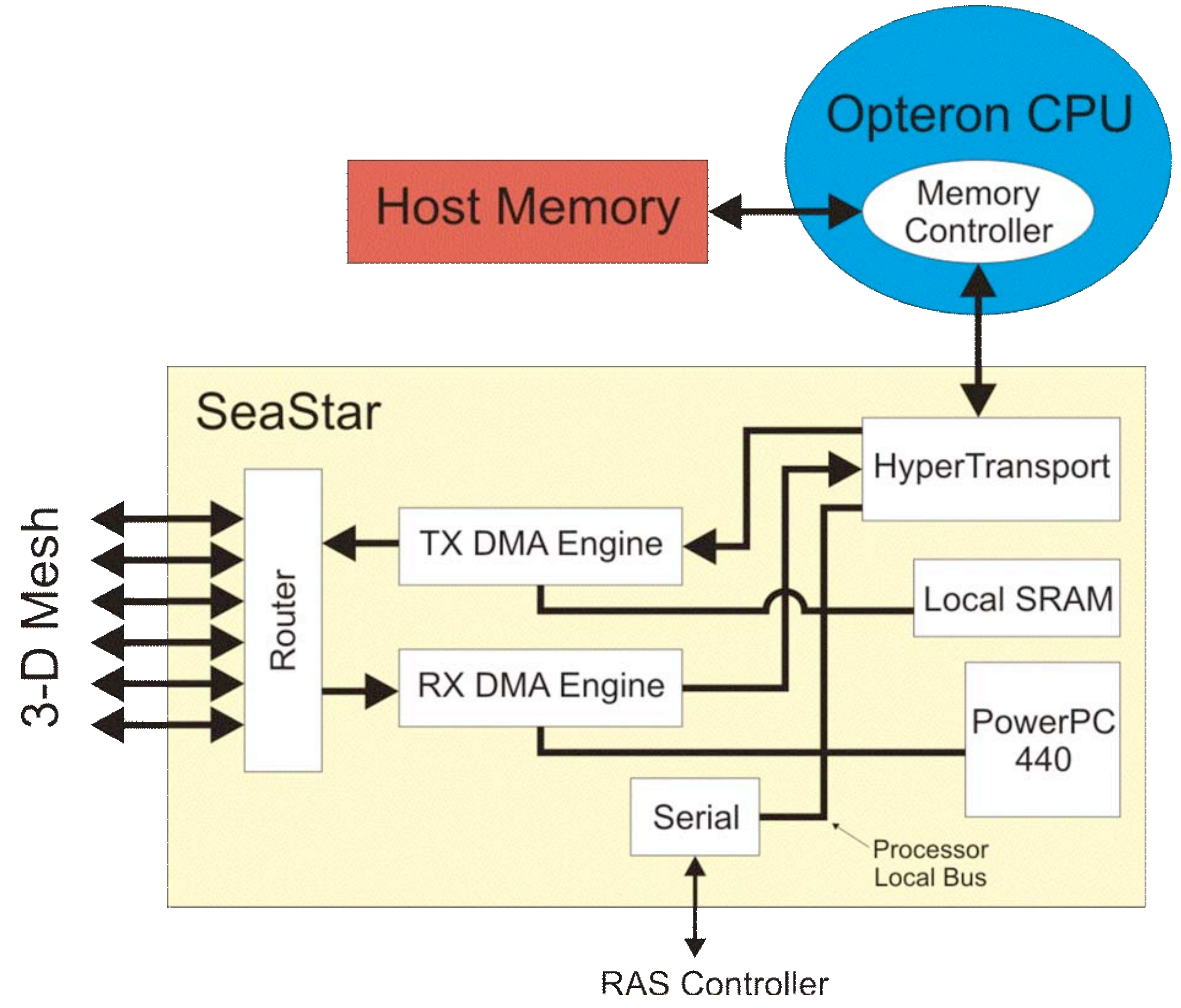
Outline

- Motivation
- SeaStar overview
- Development tools
- Description of C-based SeaStar firmware
- Optimization techniques
- Conclusion

Motivation

- Enhance attractiveness of Red Storm/SeaStar as a NIC offload research platform
- C strikes good balance of usability and performance
  - Excels at interfacing with hardware
  - Hides register allocation and instruction scheduling
  - Compiler optimizations reduce performance penalty
    - Global view
    - Function inlining
    - Passing function arguments in registers (ABI)
  - Interoperates well with assembly

SeaStar Block Diagram



Development Tools – GNU Toolchain

- Pre-packaged Binutils/GCC source distribution
  - `gas, gcc, ld, objdump, etc.`
- Targets little-endian PPC440
  - PowerPC usually big-endian, PPC440 supports both
  - Opteron is little-endian only
  - Targeting little-endian avoids endianess conversions
  - Introduces some quirks
    - GCC little-endian PPC440 support less mature
    - Some HW instructions not available (e.g., `lmw`)

Linking and Loading

- **Link using GNU `ld` and custom linker script**
- **Defines SRAM regions for:**
  - **Text (cached)**
  - **Stack (cached)**
  - **Uncached data**
  - **Cached data**
- **ELF image converted to flat binary image**
  - **166 KB ELF → 22 KB binary image**
- **Loading accomplished using Cray provided mechanism**

Debugging and Profiling

- GDB would be nice, difficult to implement
- Developed tracing tool
- Trace points inserted with:

  ```
  trace(), trace_val(uint8_t value)
  ```
- Trace records stored in uncached SRAM ring buffer
- Tool on Service Management Workstation (SMW) can retrieve any node's trace log at any time
- 10 ns overhead per trace point, no overhead if tracing turned off at compile time

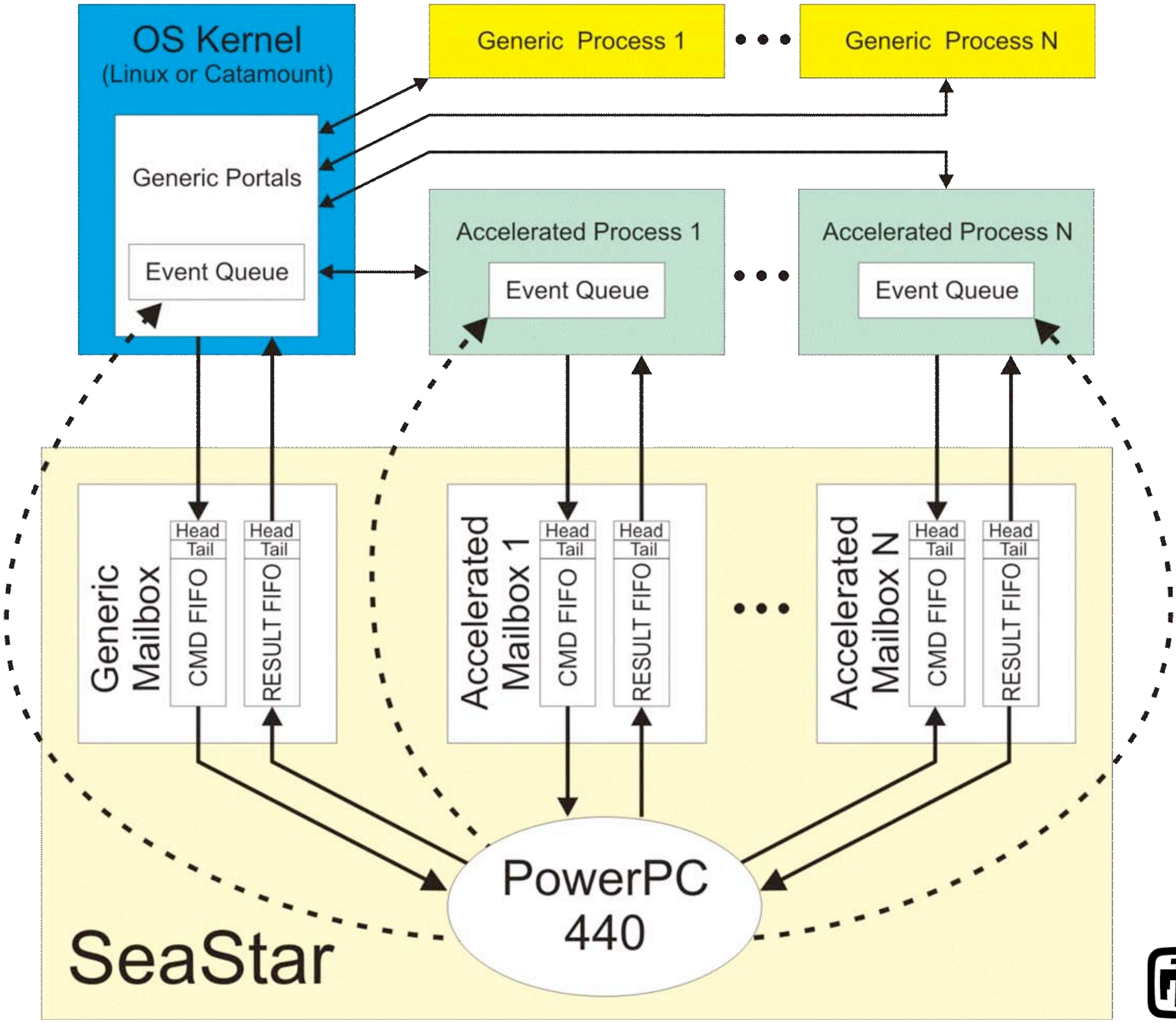
Example of `tracedump` Output

```
e4681e8b:  24: 18ff0c4c mainloop+0014 (178 ns)
e4681ee4:  99: 63ff313c rx_complete+001c (84 ns)
e4681f0e:   0: 00ff318c rx_complete+006c (24 ns)
e4681f1a:   0: 00ff3198 rx_complete+0078 (374 ns)
e4681fd5:   5: 05ff12b4 handle_command+0030 (156 ns)
e4682023:   0: 00ff24b0 goaccel_tx_command+001c (80 ns)
e468204b:   0: 00ff24c0 goaccel_tx_command+002c (40 ns)
e468205f:   0: 00ff2124 resolve_source+0030 (18 ns)
e4682068:   0: 00ff212c resolve_source+0038 (102 ns)
e468209b:   0: 00ff2504 goaccel_tx_command+0070 (290 ns)
```

C Firmware

- Supports Portals API
- 3,434 SLOC C, 253 SLOC Assembly
- 22 KB Firmware image
- Originally derived from Cray assembly-based FW in Nov. 2005

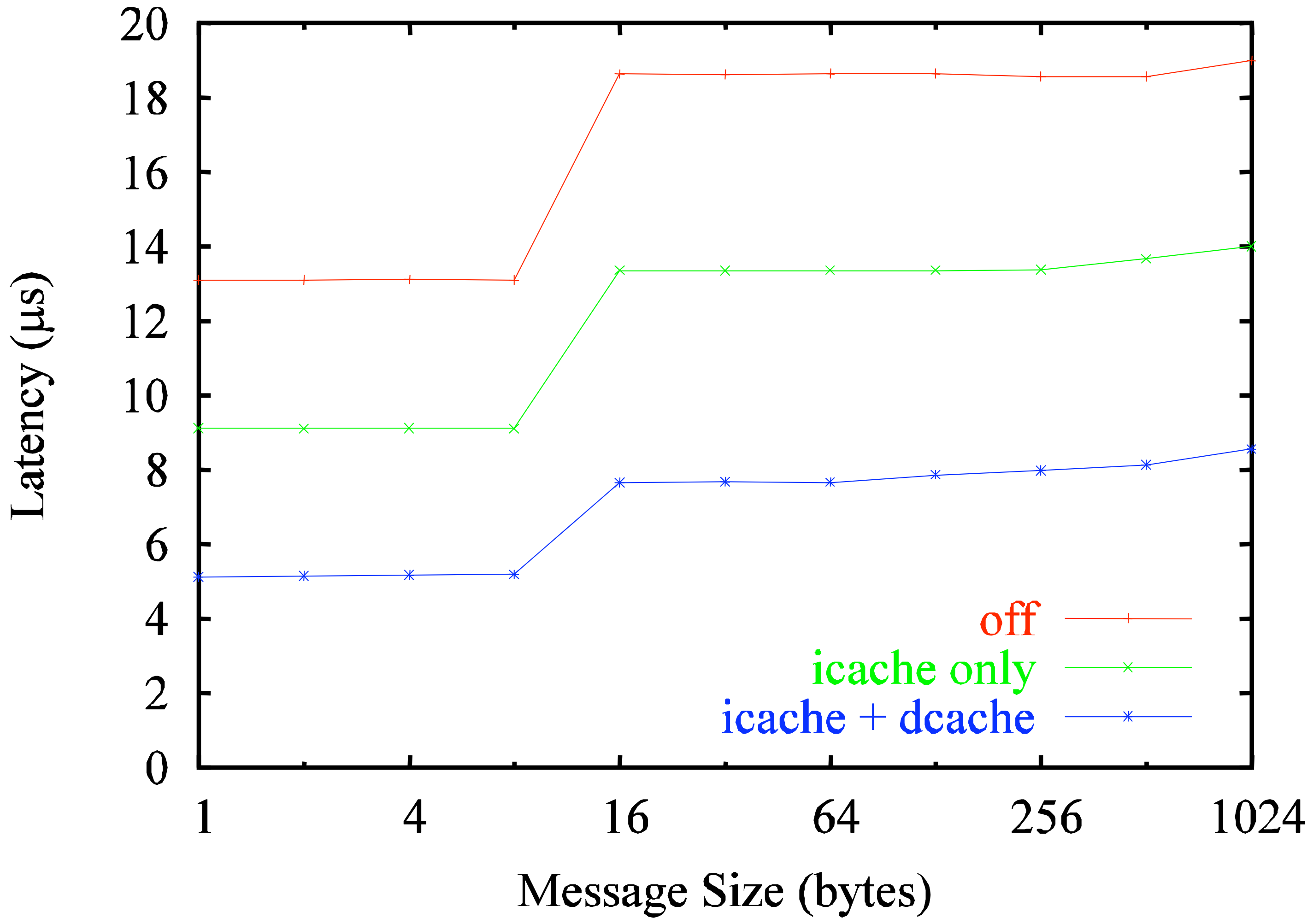
Firmware Host Interface



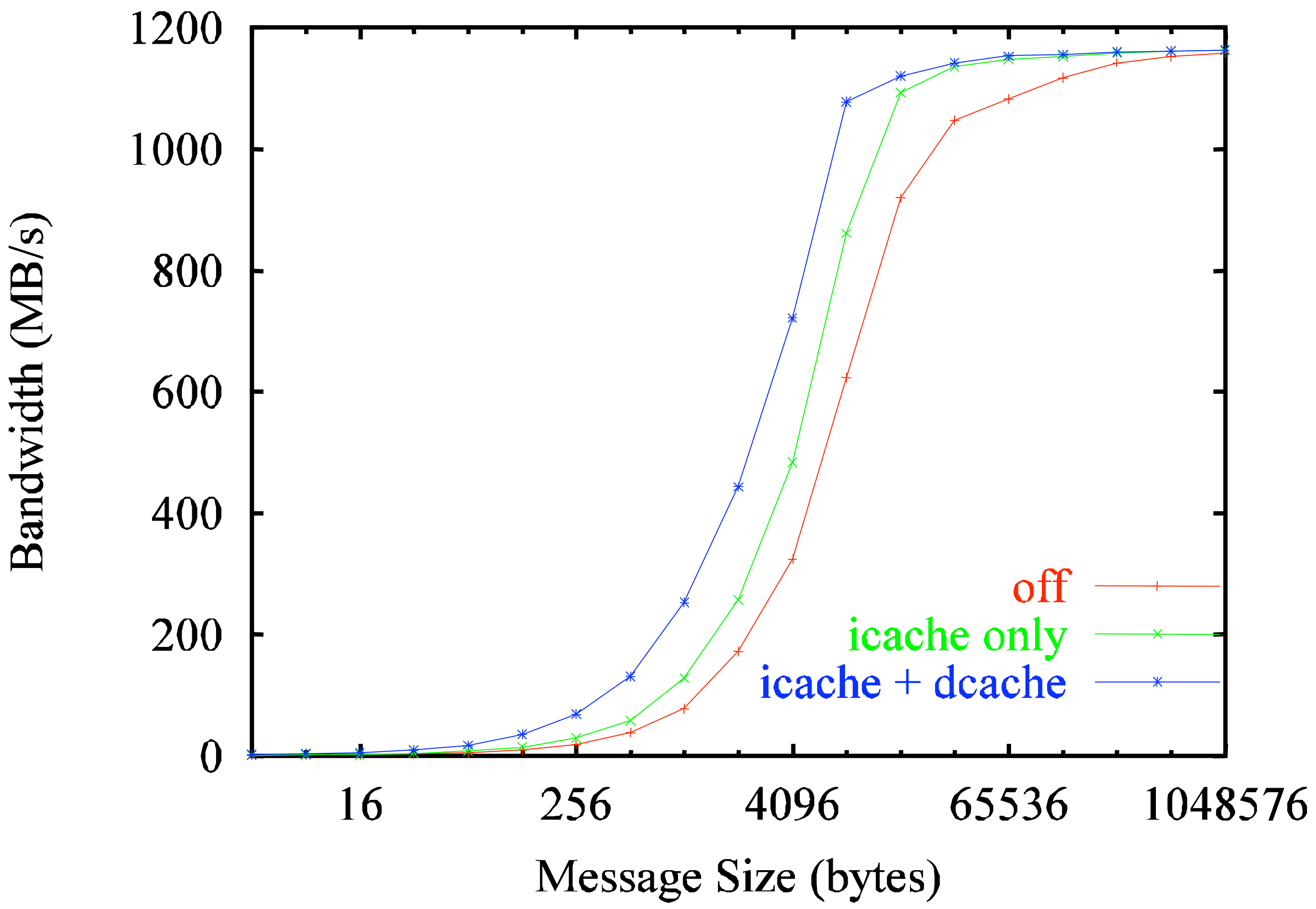
Optimization – Caching

- **PPC440 has 32 KB Icache, 32 KB Dcache**
- **Data shared with Opteron is uncached**
- **Data accessed only by firmware may be cached**
- **Use separate PPC440 address space to avoid conflicts with Cray loader**

Effect of PPC440 Caching on MPI Latency



Effect of PPC440 Caching on MPI Bandwidth

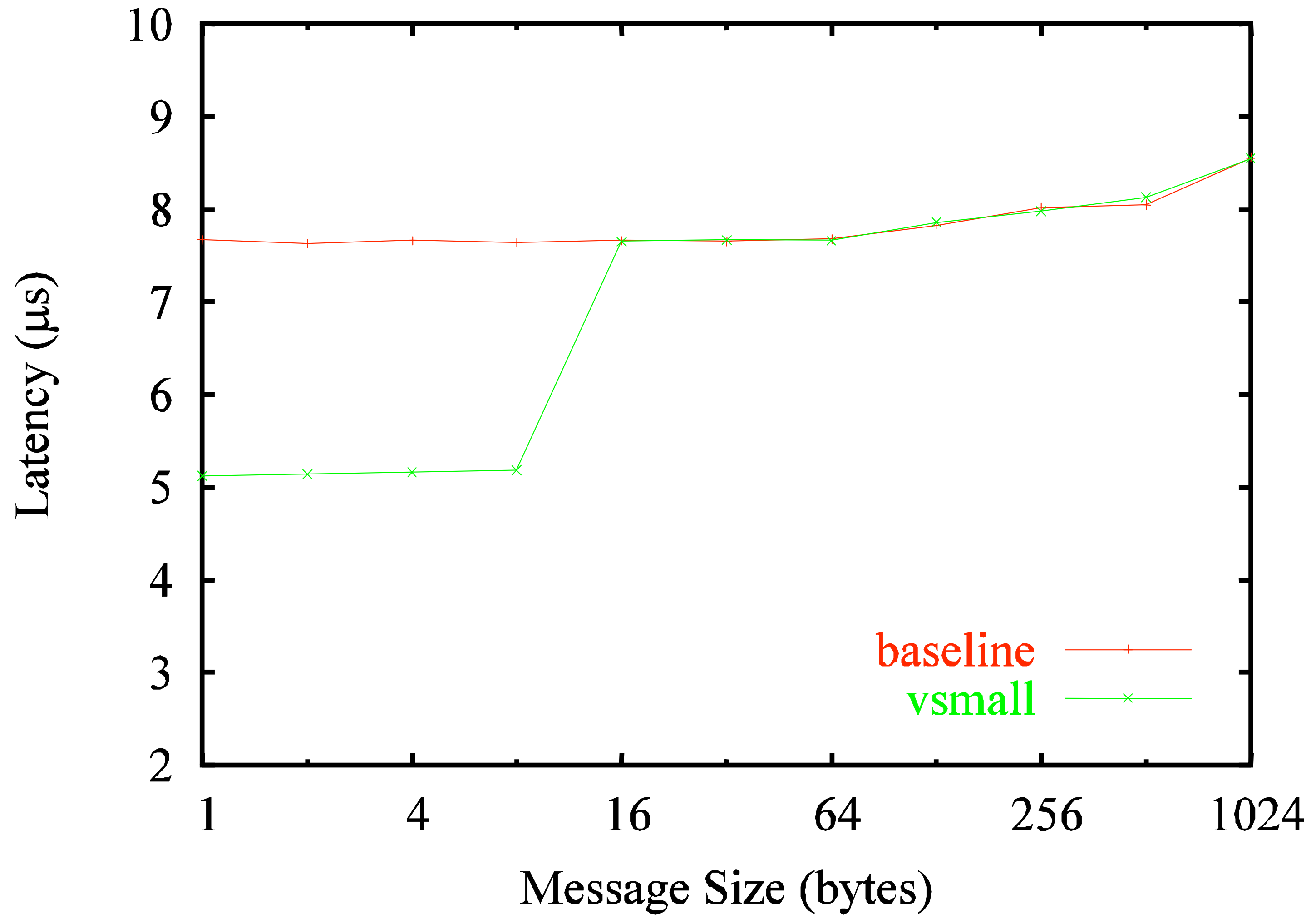


Optimization – Interrupt Elimination

- Generic mode requires 2 interrupts per RX
  - 1) Firmware asks host where to put message
  - 2) Firmware tells host when RX is complete
- Optimization – copy 'very small' messages up to the host before the first interrupt
  - Eliminates second interrupt
  - Not zero-copy
- Currently very small <= 12 bytes
- Could extend to larger message sizes with additional work

Sandia National Laboratories

Effect of Very Small Message Optimization on MPI Latency



Optimization – Write Through Techniques

- Goal is to avoid uncached reads from SRAM
- Keep shadow copy of uncached structures in cached memory:

```
niccb.heartbeat = ++cached_niccb.heartbeat;
```

- FIFO manipulation optimized in same way
  - Always read cached TAIL pointer
  - Only refresh HEAD pointer when FIFO might be full
- Saved approximately 6 us per message

Sandia National Laboratories

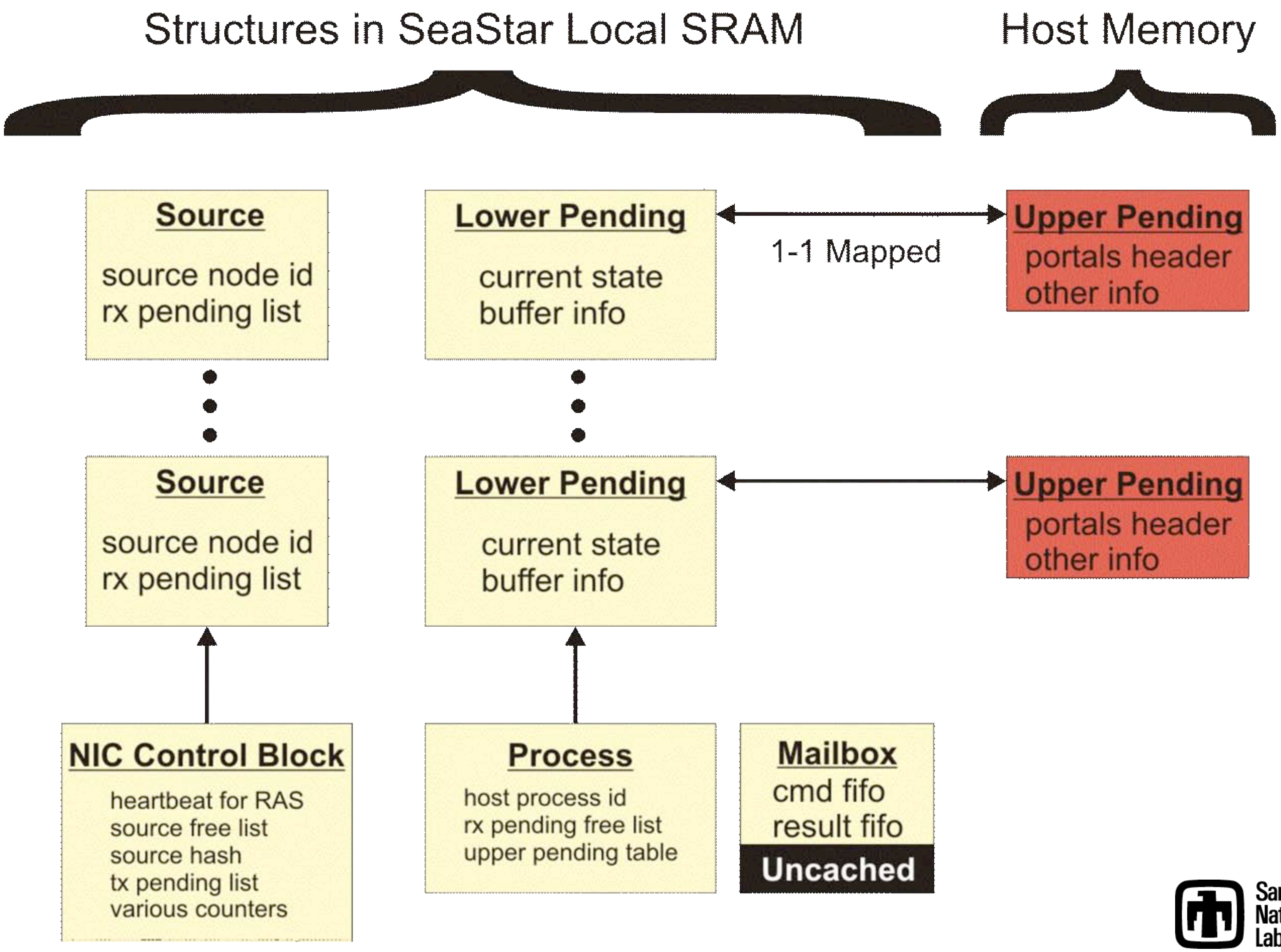
Conclusion

- SeaStar firmware can be developed in C
- Working C-based firmware that others can extend
- Significant optimization to date, more planned
  - Initial C firmware 30 us → currently 4.9 us
- “Let the firmware development begin”

Sandia National Laboratories

Backup Slides

Firmware Data Structures



Optimization – Pinning Globals in Registers

- Neat GCC-specific optimization:

```
register process_t *process asm ("r31");
register source_t  *source  asm ("r30");
register pending_t *pending asm ("r29");
```

- Eliminates loads/stores
  - Faster, especially for uncached
  - Results in smaller firmware
- Have used up to 10 registers for globals with good results