



EXOCHI: Architecture and Programming Environment for A Heterogeneous Multi-core Multithreaded System

Perry H. Wang¹, Jamison D. Collins¹, Gautham N. Chinya¹,
Hong Jiang², Xinmin Tian³, Milind Girkar³, Nick Y. Yang²,
Guei-Yuan Lueh², and Hong Wang¹

Microarchitecture Research Lab, Microprocessor Technology Labs, Intel Corporation¹

Graphics Architecture, Chipset Group, Intel Corporation²

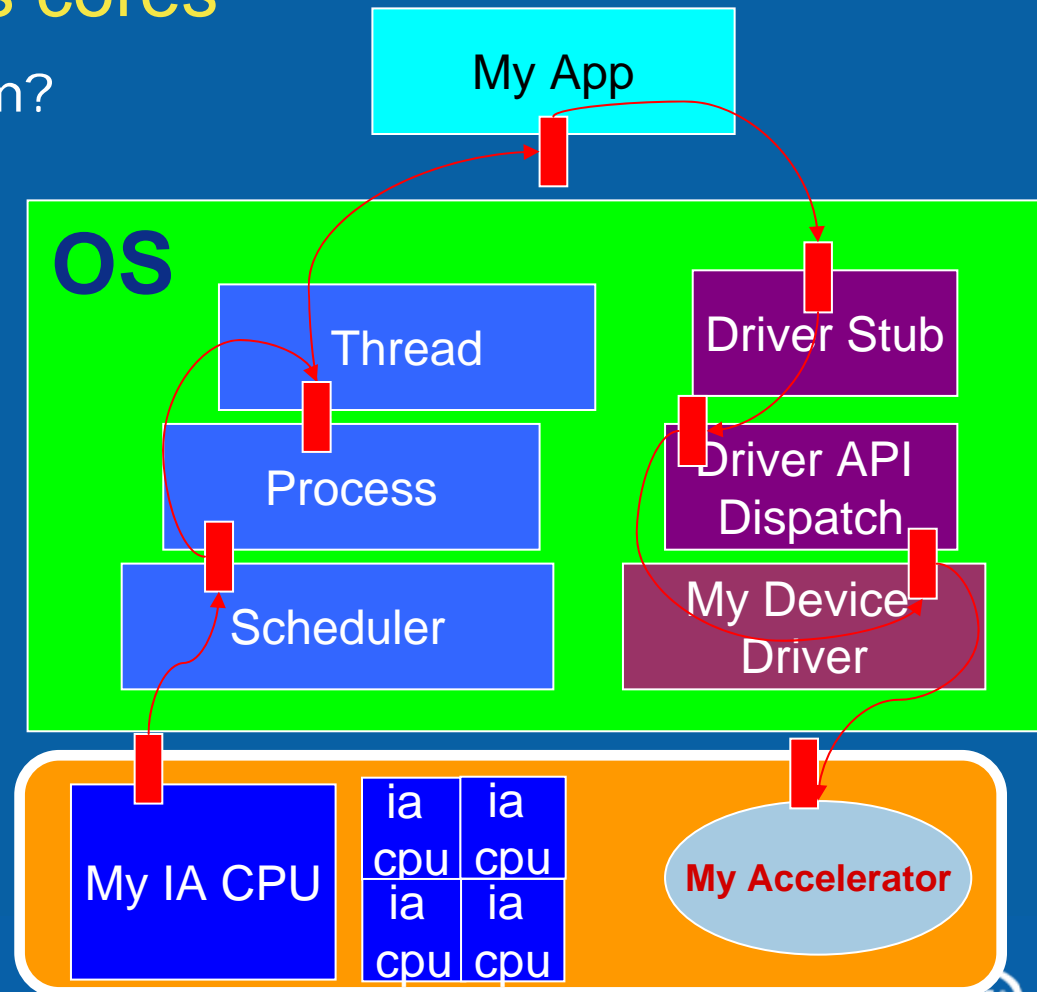
Intel Compiler Lab, Software Solutions Group, Intel Corporation³



Motivation

Future mainstream microprocessors will likely integrate heterogeneous cores

- How will we *program* them?
- Map computation to driver / abstraction API
- Unfamiliar development / debugging flow
- OS / driver overheads
- Accelerator in distinct memory space



Our Approach – EXOCHI

EXO: IA ISA extension for MIMD

- Heterogeneous engines as IA MIMD function units (exo-sequencer)
- Shared virtual memory multithreaded programming paradigm

CHI: IA programming environment for exo-sequencers

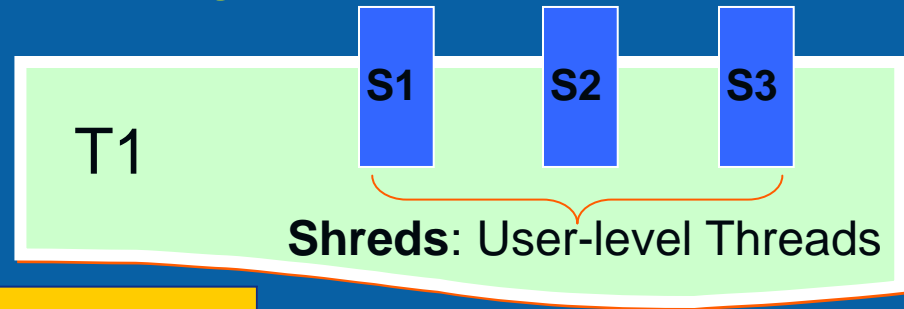
- Familiar IA look-n-feel for application software development / debugging flows
- OpenMP extension for shared memory heterogeneous multi-shredding

Outline

- Overview
- **EXO: Heterogeneous MIMD ISA Extension to IA**
- CHI Programming Environment
- Prototype
- Conclusion

MISP: MIMD ISA for *Asymmetric* IA Cores

Single-Threaded
App

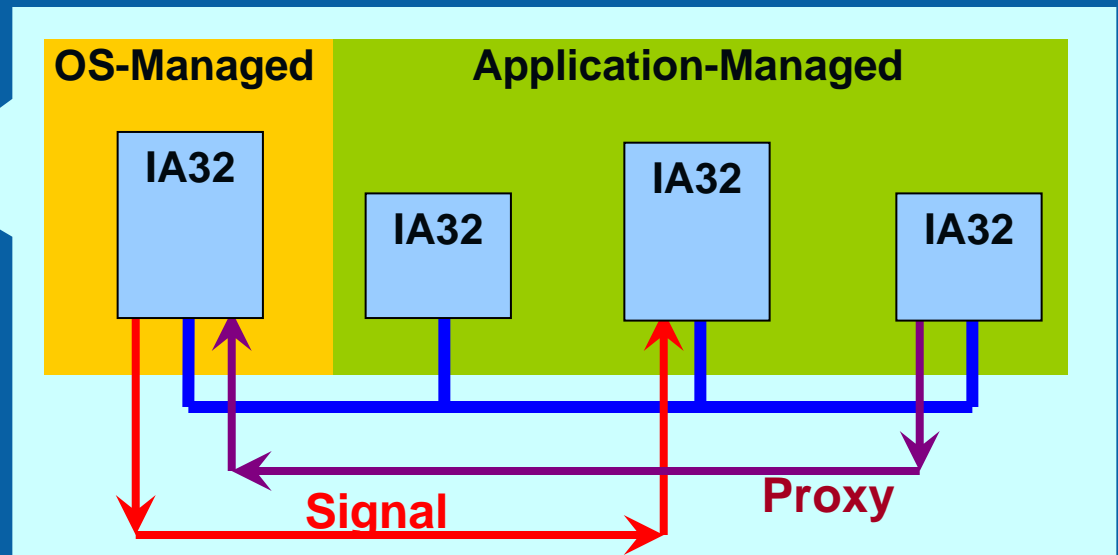


Uniprocessor
OS



“MIMD Function Units”

MISP
“Uniprocessor”



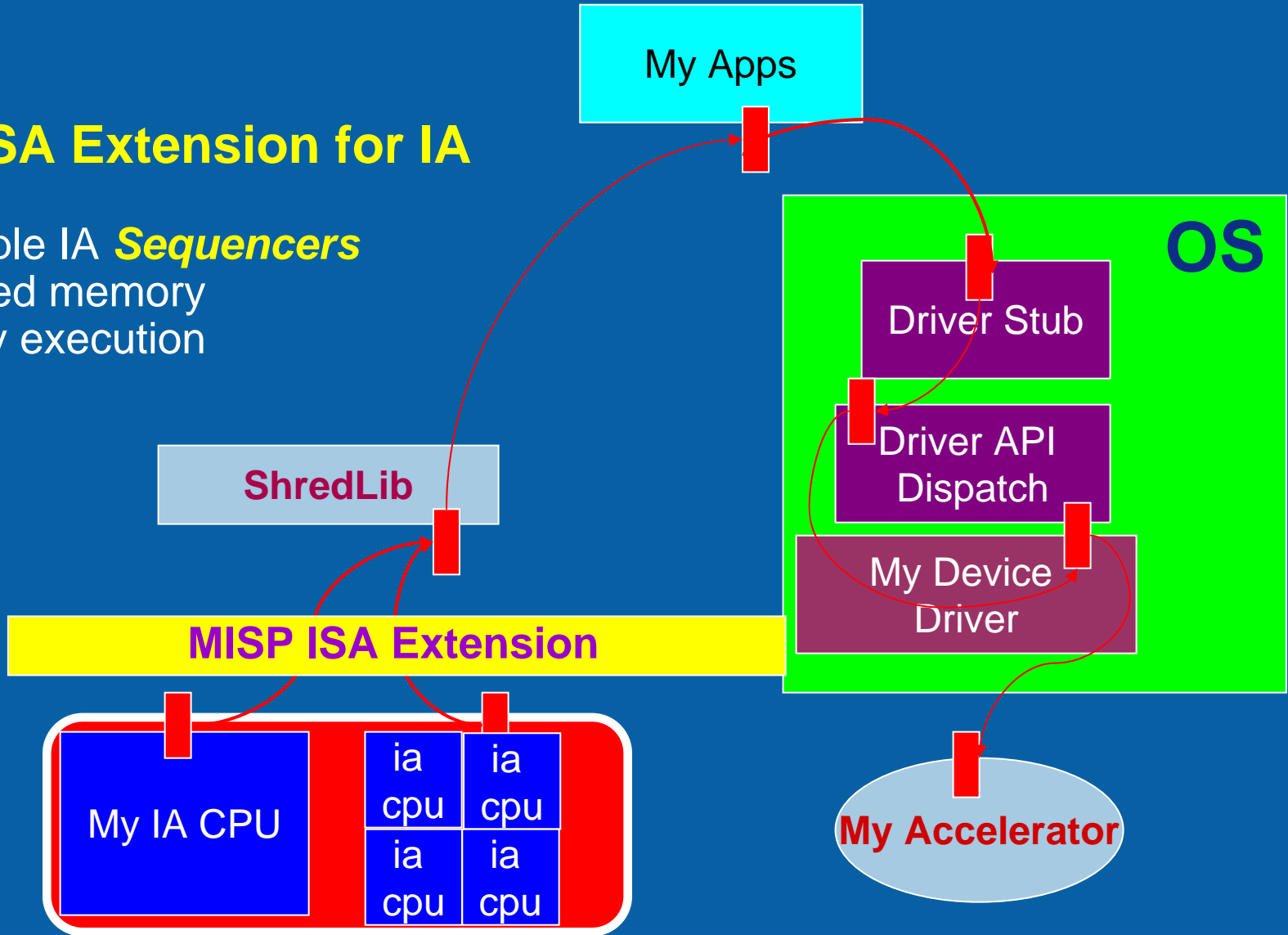
MISP “MIMD” ISA

- SIGNAL (SID, <PC, SP>)
- Proxy Execution
- Shared Virtual Memory

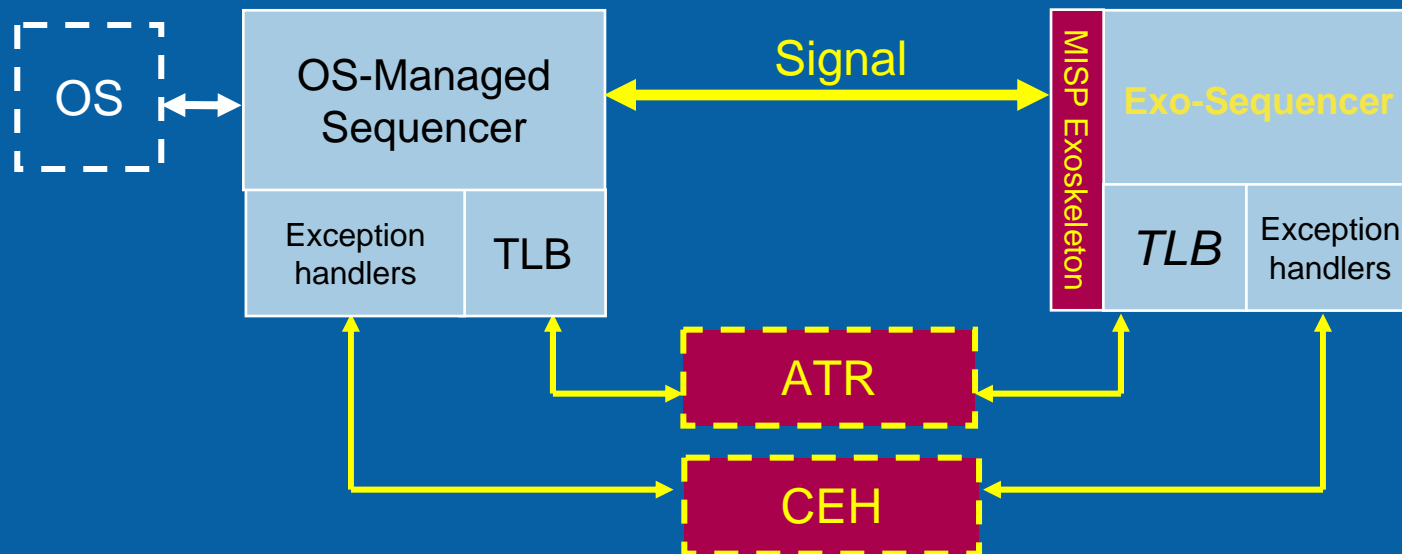
MISP: MIMD ISA for *Asymmetric* IA Cores

• MIMD ISA Extension for IA

- Multiple IA *Sequencers*
- Shared memory
- Proxy execution



Exoskeleton Sequencer – Extend MISp to Heterogeneous Sequencers

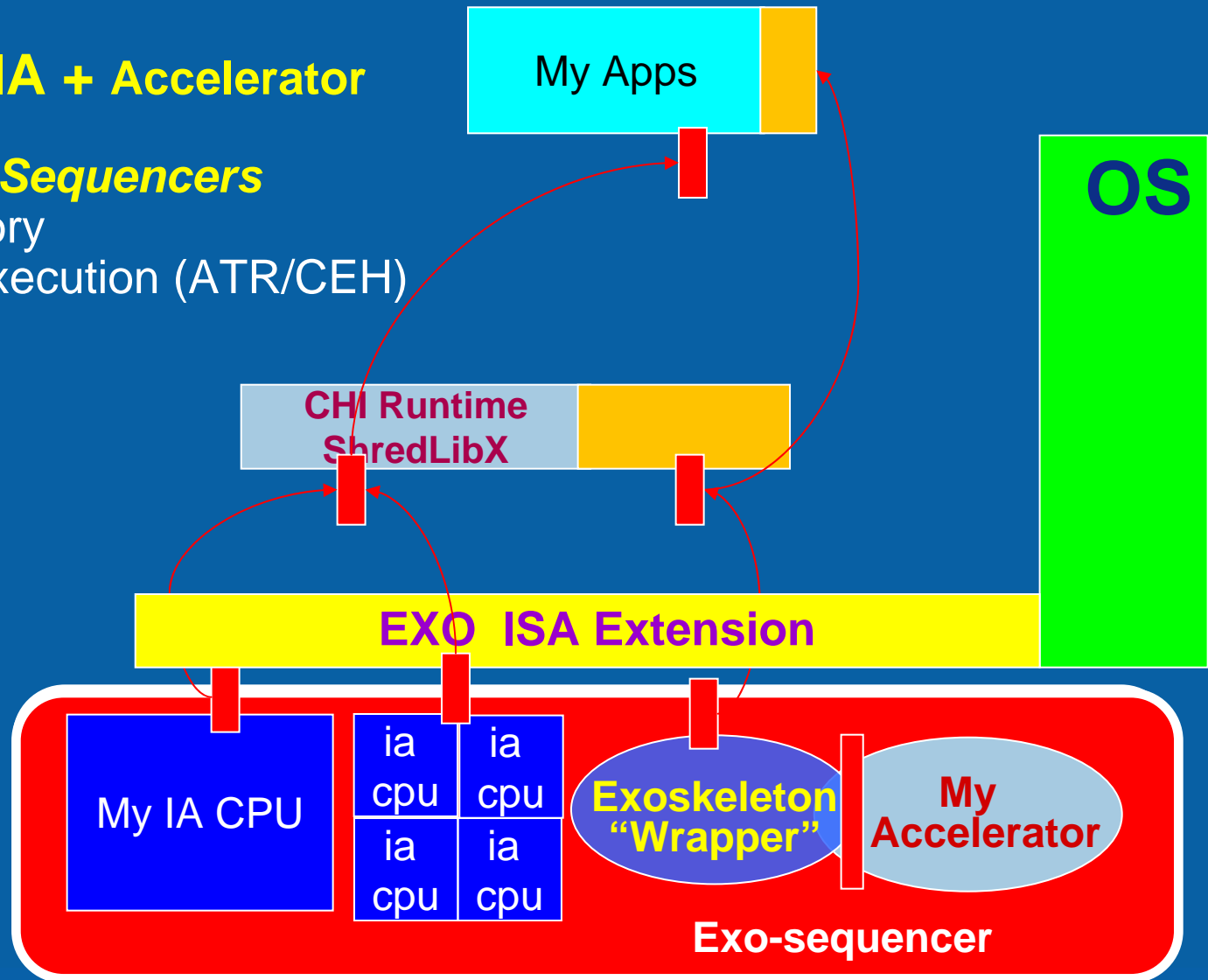


- **OS-Managed Sequencer**
 - Support for MISp interface for various different architectures
- **Exo-Sequencer**
 - Support for MISp interface for various different architectures
 - Signal OMS on page fault
- **ATR**
 - Utilize robust IA exception handling support
 - IEEE double precision floating point exceptions

EXO: MIMD ISA for *Heterogeneous* CMP

• MIMD ISA for IA + Accelerator

- *Exo-skeleton Sequencers*
- Shared memory
- EXO-Proxy execution (ATR/CEH)



CHI Programming Environment

Compiler

- Modified front-end pragmas
 - Fork/join
 - Producer/consumer
- Generates fat binaries

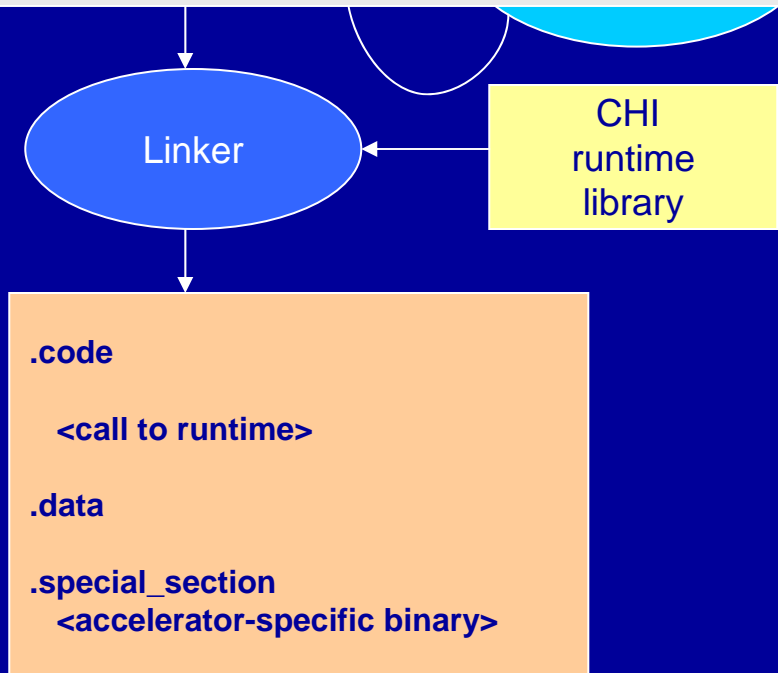
```
#pragma omp parallel target(targetISA) [clause[,]clause]...  
    structured-block
```

Where clause can be any of the following:

```
    firstprivate(variable-list)  
    private(variable-list)  
    shared(variable-ptr-list)  
    descriptor(descriptor-ptr-list)  
    num_threads(integer-expression)  
    master_nowait
```

CHI runtime

- Multi-shredding: User-level threading
- Extensible to multiple types of heterogeneous cores
 - E.g. Intel GMA X3000
 - E.g. A data streaming systolic array accelerator for communication



CHI Programming Example

```
1.  int *A = malloc(n);
2.  int *B = malloc(n);
3.  int *C = malloc(n);
4.
5.  A_desc = chi_alloc_surface(A, X3000_INPUT, n, 1);
6.  B_desc = chi_alloc_surface(B, X3000_INPUT, n, 1);
7.  C_desc = chi_alloc_surface(C, X3000_OUTPUT, n, 1);
8.  #pragma omp parallel target(x3000) shared(A,B,C)
9.      descriptor(A_desc,B_desc,C_desc) private(i)
10. {
11.     for (i=0; i<n/8; i++)
12.         __asm
13.         {
14.             shl.1.w  vr1 = i, 3
15.             ld.8.dw [vr2..vr9] = (A, vr1, 0)
16.             ld.8.dw [vr10..vr17] = (B, vr1, 0)
17.             add.8.dw [vr18..r25] = [vr2..vr9], [vr10..vr17]
18.             st.8.dw (C, vr1, 0) = [vr18..vr25]
19.         }
20. }
```

Outline

- Motivation
- EXO: Heterogeneous MIMD ISA Extension to IA
- CHI Programming Environment
- **Prototype**
- Conclusion

EXOCHI Prototype

Intel Core 2 Duo Processor + Intel GMA X3000 Media Accelerator

- IA core (**1 IA sequencer**) + 8 GMA X3000 cores (**32 exo-sequencers**)

Custom firmware to emulate EXO ISA extension

Intel C++ Compiler with CHI OpenMP runtime extensions

Execute production-quality media-processing kernels

- Highly parallel (up to 1000s of shreds)
- Measure wall-clock execution time

IA Look-n-Feel: Development and Debugging

The screenshot displays the Microsoft Visual Studio IDE. The main window shows the source code for 'accelerator_exoskeleton.c'. A 'Debug' window is open, showing the assembly code being executed. The assembly code is as follows:

```
__asm
{
    shl.1.w vr1 = i, 3
    ld.8.dw [vr2..vr9] = (A_desc, vr1, 0)
    ld.8.dw [vr10..vr17] = (B_desc, vr1, 0)
    add.8.dw [vr18..r25] = [vr2..vr9], [vr10..vr17]
    st.8.dw (C_desc, vr1, 0) = [vr18..vr25]
}
```

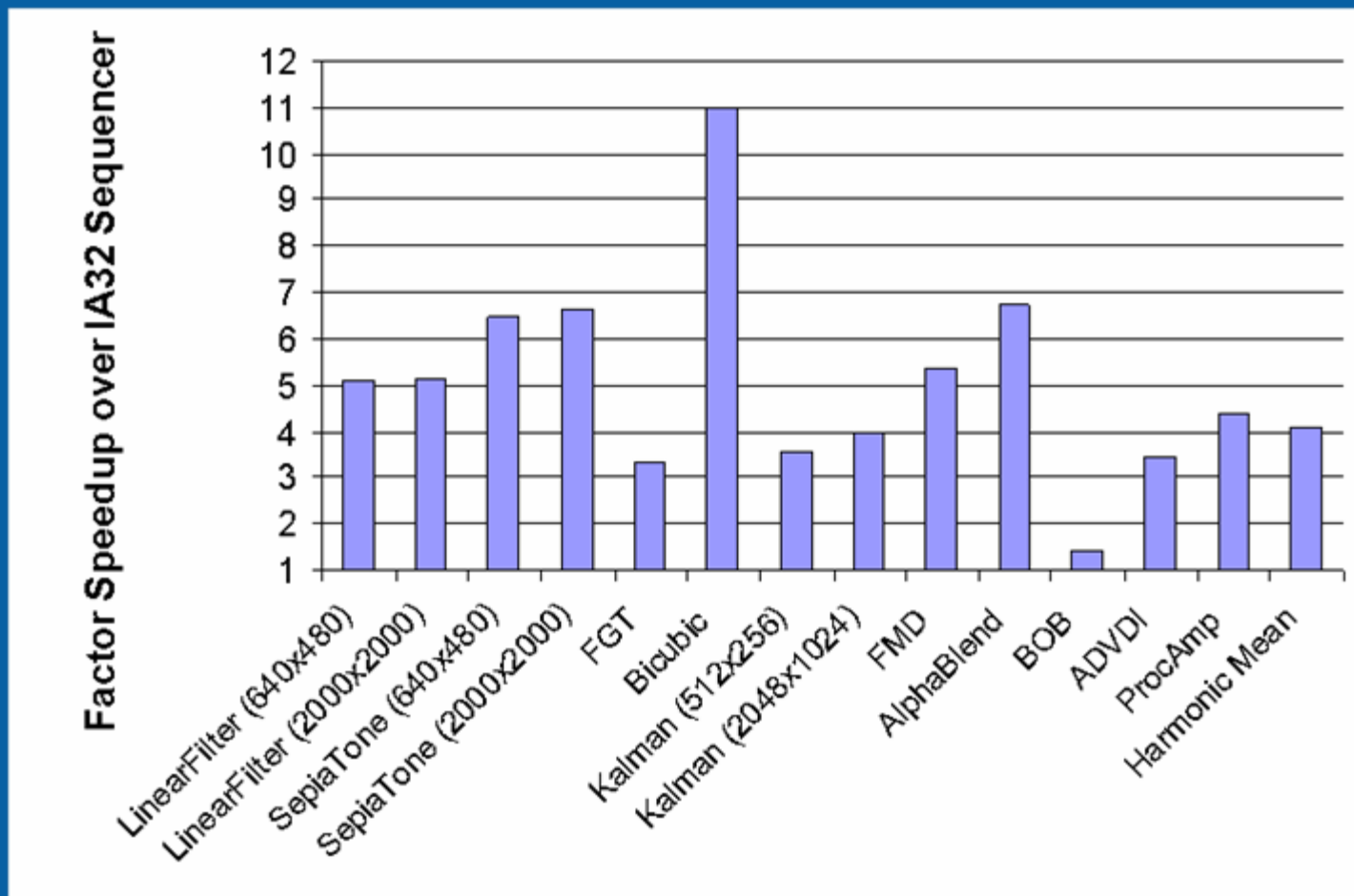
The Debug window also includes a 'GRF Register View' at the bottom right, which is currently empty. The status bar at the bottom indicates 'Building workspace: (100%)'.

IA Look-n-Feel: Compilation and Execution

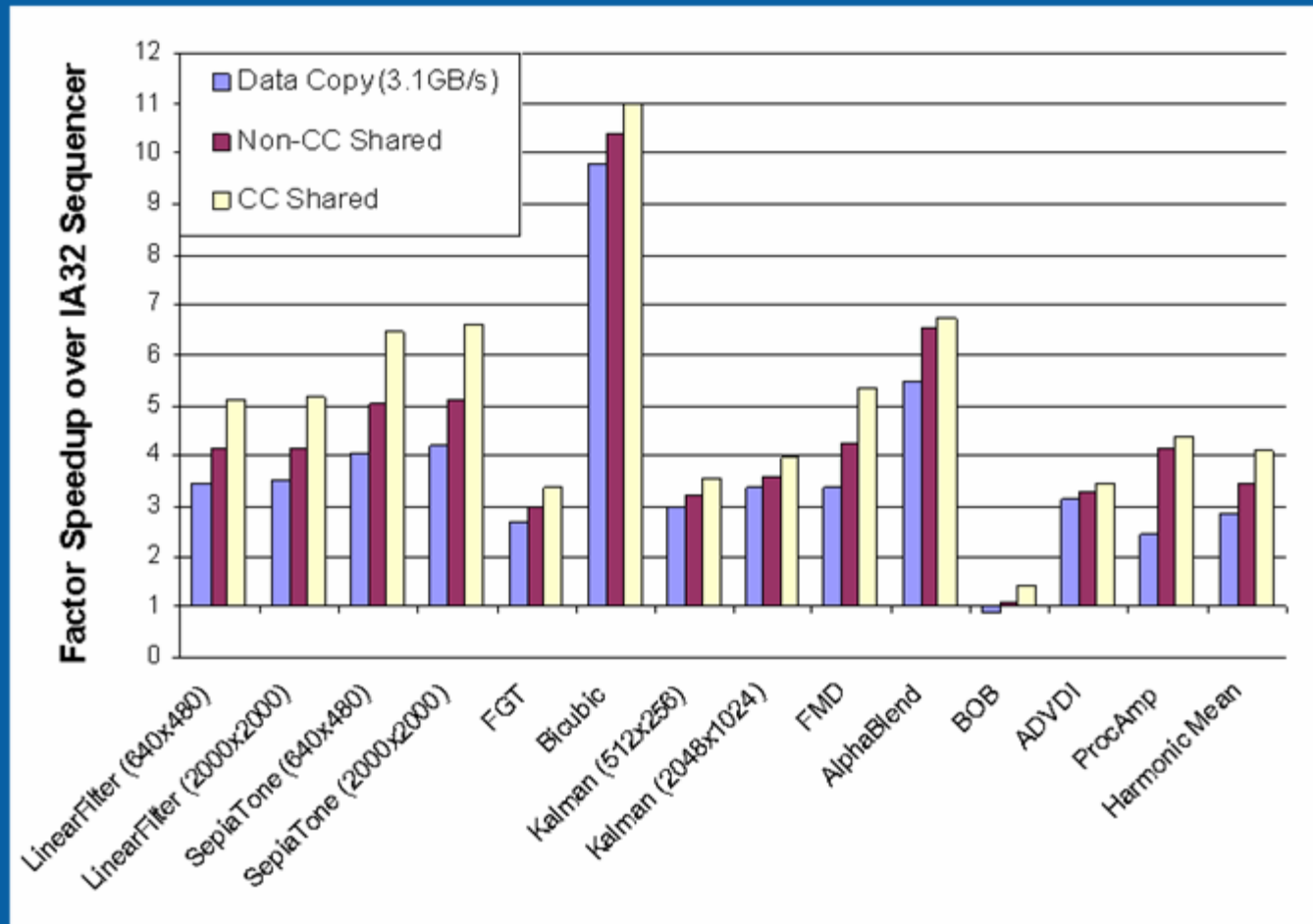
```
Visual Studio 2005 Command Prompt

C:\accelerator_exoskeleton\linear>compile
C:\accelerator_exoskeleton\linear>icl /Qopenmp /mGLOB_use_msasm ShredLibX.lib /I..\include linearfilter.c
Intel(R) C++ Compiler for applications running on IA-32, Version 10.0.0.424, November 14, 2008.  Copyright (C) 1985-2006 Intel Corporation.  All rights reserved.
linearfilter.c
Begin compiling X3000 assembly
End compiling X3000 assembly
Microsoft (R) Incremental Linker Version 6.00.50727.42
Copyright (C) Microsoft Corporation.  All rights reserved.
-out:linearfilter.exe
-nodefaultlib:libguide_stats.lib
-nodefaultlib:libguide40_stats.lib
-defaultlib:libguide.lib
ShredLibX.lib
linearfilter.obj
C:\accelerator_exoskeleton\linear>run
C:\accelerator_exoskeleton\linear>linearfilter.exe ..\input\holly.bmp out.bmp
Linearfilter processed!
C:\accelerator_exoskeleton\linear>
```

Speedup of GMA X3000 over CPU Alone



Impact of Shared Virtual Memory and Cache Coherency



Summary

EXOCHI provides an IA look-n-feel for programming heterogeneous accelerators

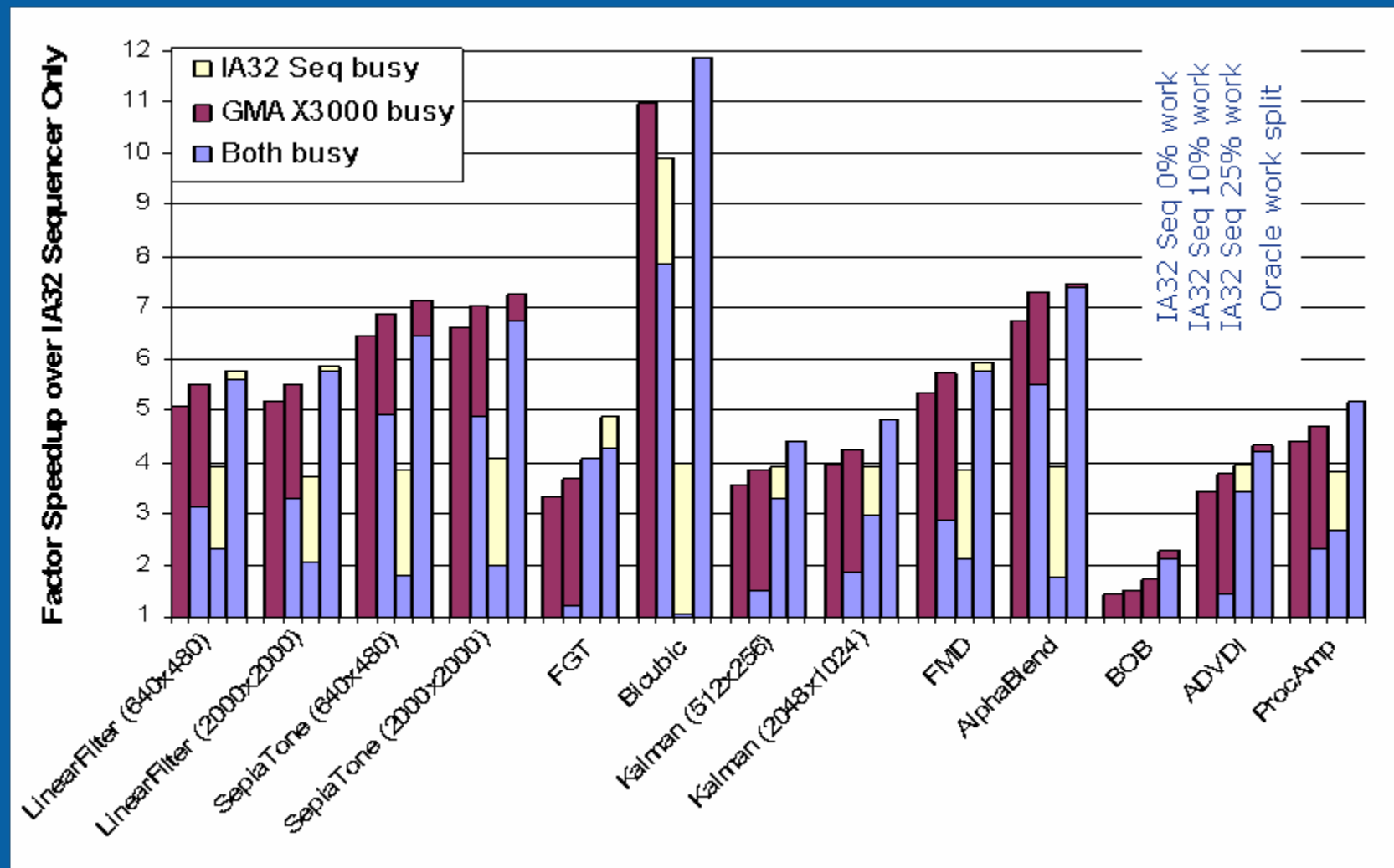
- EXO MIMD ISA extension for shared virtual memory multithreaded programming paradigm
- CHI Programming environment for IA SW development look-n-feel
 - Compiler, debugger, multi-shredding runtime
 - Essential for productivity

Complete hardware and software prototype

- Significant performance improvements possible over IA sequencer
- Cache coherency improves performance but less significant than shared virtual memory

Thank You!

Cooperative Multi-shredding Performance



Shared Virtual Memory and Cache Coherence

- Cache coherency *not* a requirement for shared virtual address space
- Use critical sections for mutual exclusion to shared data
- Flush dirty cache lines back to memory prior to releasing lock / mutex