Announcements

- Seminar talk by Prof. Kasahara for next class (10/8)
  - “A Multi-core Parallelizing Compiler for Low-Power High-Performance Computing”, DH 3076, 4pm - 5pm (refreshments at 3:30pm)
  - [http://events.rice.edu/index.cfm?EventRecord=8826](http://events.rice.edu/index.cfm?EventRecord=8826)
- REMINDER: 4-page project/study write-up due by 12/7/07
  - Report can be prepared in a group - just plan on 4 pages/person in that case
  - Send me email by October 12th with proposed topic for your write-up
- Today's class
  - Discussion of two papers
      - [http://doi.acm.org/10.1145/1250734.1250753](http://doi.acm.org/10.1145/1250734.1250753)
- Acknowledgments
  - SIGGRAPH/EUROGRAPHICS symposium presentation from Shubho Sengupta
Summary:

- EXO (Exoskeleton Sequencer) architecture for heterogeneous multi-core processors with support for a shared virtual memory.
  - Prototype implementation: Intel Duo IA32 processor w/ an 8-core 32-thread Graphics Media Accelerator (GMA) X3000
  - CHI (C for Heterogeneous Integration) compiler and runtime based on OpenMP extensions for heterogeneous processors
    - Implementation based on Intel C++ compiler
  - Speedup results reported for 10 media-processing kernels, comparing performance of IA32 + GMA relative to IA32 w/o GMA
    - Harmonic mean of speedup was approximately 4x
    - Max speedup was 10.97x for Bicubic Scaling (2700 shreds)

Comparison of EXOCHI w/ standard GPGPU programming environments

**GPGPU approach:**
- CPU managed by OS
- GPU cores managed by device driver
- Host and device run in separate address spaces

**EXOCHI approach:**
- CHI runtime library executes in user space -- GMA state included in IA32 thread state for context switching
- Shared virtual address space between IA32 and GMA sequencers
### Address Translation Remapping (ATR)

- ATR allows IA32 to handle page faults on behalf of exo-sequencers (GMA).
- Problem: how to deal with different page table formats in IA32 and GMA TLBs?
- Solution:
  - When GMA incurs a translation miss, it suspends execution and requests IA32 to service TLB miss or page fault.
  - After miss is serviced, ATR transcodes IA32 page table entry to GMA’s page table format and inserts transcoded entry into GMA’s TLB.
  - Both IA32 and GMA will map same virtual page to same physical page.
  - ATR does not support coherence.

![ATR Diagram](image)

### Collaborative Exception Handling (CEH)

- Extend TLB miss approach to other exceptions or faults that need OS services.
- Example: numeric exception on exo-sequencer.
- Problem: faulting vector instruction on GMA may not have an equivalent IA32 instruction.
- Solution: proxy execution in IA32 via software emulation of GMA.
  - Use application-level handler or OS service such as Structured Exception Handling (SEH).
  - CEH updates result in GMA before resuming execution.

![CEH Diagram](image)
CHI Programming Environment

- CHI = C for Heterogeneous Integration
- "shred" = lightweight task on IA32 or GMA
- Three main capabilities offered by CHI
  - Inline Accelerator Assembly Support
    - Method to inline region of accelerator-specific code written in assembly or domain-specific language
  - OpenMP pragma extensions
    - Supports fork-join or producer-consumer style shred-level parallel execution
  - Runtime APIs
    - Method to specify input and output memory regions and live-in values for accelerator-specific code regions

CHI Code Example #1

```c
1. A_desc = chi_alloc_desc(X3000, A, CHI_INPUT, n, 1);
2. B_desc = chi_alloc_desc(X3000, B, CHI_INPUT, n, 1);
3. C_desc = chi_alloc_desc(X3000, C, CHI_OUTPUT, n, 1);
4. #pragma omp parallel target(X3000) shared(A, B, C)
5. descriptor(A_desc,B_desc,C_desc) private(i) nontar_nowait
6. {
7.   for (i=0; i<n/8; i++)
8.     __asm
9.     {   
10.       shr.1.w vr1 = i, 3
11.       ld.8.dw [vr2..vr9] = (A, vr1, 0)
12.       ld.8.dw [vr10..vr17] = (B, vr1, 0)
13.       add.8.dw [vr18..r25] = [vr2..vr9], [vr10..vr17]
14.       st.8.dw (C, vr1, 0) = [vr18..v26]
15.     }
16. }
17. #pragma omp parallel for shared(D,E,F) private(i)
18. {
19.   for (i=0; i<n; i++)
20.     F[i] = D[i] + E[i];
21. }
```

**Figure 6.** CHI Code Example with GMA X3000 Pseudo-code
CHI Code Example #2

1. n = 600;
2. GMA_iters = 600;
3. IN_desc = chi_alloc_desc(X5000, IN, CHI_INPUT, n, 1);
4. OUT_desc = chi_alloc_desc(X5000, OUT, CHI_OUTPUT, n, 1);
5. #pragma omp parallel target(X3000) shared(IN, OUT)
6. descriptor(IN_desc,OUT_desc) private(i) master_nowait
7. {
8.   for (i=0; i< GMA_iters; i++)
9.   { __asm
10.     ...
11.   }
12. }
13. #pragma omp parallel for shared(IN, OUT) private(i)
14. {
15.   for (i=GMA_iters; i<n; i++)
16.   ...
17. }

Figure 9. Cooperative Execution Code Example which Executes 600 Loop Iterations on GMA X3000 Exo-sequencers and 200 Loop Iterations on the IA32 Sequencer

Inline Accelerator Assembly Support

Approach:
- Use "__asm" syntax for inline assembly
- Accelerator-specific assembler is linked in with C/C++ compiler
- Target ISA is specified by target clause in OpenMP pragma
- Resulting binary is embedded in special code section of IA32 executable

Figure 4. CHI Compilation Flow
OpenMP Pragma Extensions (1/2)

• Target clause specifies target accelerator ISA
• num_threads specifies number of shreds to be executed on accelerator
• master_nowait allows master IA32 thread to continue execution
• firstprivate creates a private variable on each shred, initialized to the same live-in value
• descriptor associates read/write intents with specific variables

OpenMP Pragma Extensions (2/2)

• taskq pragma initializes task queue environment by creating an empty queue of tasks
• Root shred creates child shreds for each task construct encountered in structured-block
• Sequencing in root shred helps support producer-consumer parallelism
**Runtime APIs**

- `chi_alloc_desc`: allocate descriptor with specified input/output mode, width, height
- `chi_set_feature`: makes global change to all accelerator states
- `chi_set_feature_per_shred`: changes accelerator state on a per-shred basis

---

**Performance Evaluation: Benchmarks**

<table>
<thead>
<tr>
<th>Kernel (Abbreviation)</th>
<th>Data size</th>
<th>Description</th>
<th># GMA X3000 Shreds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Filter (LFilter)</td>
<td>600x480 image</td>
<td>Compute output pixel as average of input pixel and eight surrounding pixels</td>
<td>6,480</td>
</tr>
<tr>
<td>Sepia Tone (SepiaTone)</td>
<td>600x480 image</td>
<td>Modify RGB values to artificially age image</td>
<td>6,480</td>
</tr>
<tr>
<td>Film Grain Technology (PGT)</td>
<td>1024x768 image</td>
<td>Apply artificial film grain filter from H.264 standard</td>
<td>6,480</td>
</tr>
<tr>
<td>Bicubic Scaling (Bicubic)</td>
<td>Scale 30 frames 360x240 to 720x480</td>
<td>Scale video using bicubic filter</td>
<td>2,700</td>
</tr>
<tr>
<td>Kalman (Kalman)</td>
<td>30 frames 512x256</td>
<td>Video noise reduction filter</td>
<td>2,700</td>
</tr>
<tr>
<td>Film Mode Detection (FMD)</td>
<td>60 frames 720x480</td>
<td>Detect video cadence so inverse telcine can be applied</td>
<td>2,700</td>
</tr>
<tr>
<td>Alpha Blending (AlphaBlend)</td>
<td>Blend 64x32 image onto 720x480</td>
<td>Bi-linear scale 64x32 image up to 720x480 and blend with 720x480 image</td>
<td>2,700</td>
</tr>
<tr>
<td>De-interlace 4:2:0 Avg (DIA)</td>
<td>30 frames 720x480</td>
<td>De-interlace video by averaging nearby pixels within a field to compute missing scanlines</td>
<td>2,700</td>
</tr>
<tr>
<td>Advanced De-interfacing (ADV)</td>
<td>30 frames 720x480</td>
<td>Computationally intensive advanced de-interlacing filter with motion detection</td>
<td>2,700</td>
</tr>
<tr>
<td>ProcAmp (ProcAmp)</td>
<td>30 frames 720x480</td>
<td>Simple linear modification to YUV values for color correction</td>
<td>2,700</td>
</tr>
</tbody>
</table>

Table 2. Media-Processing Kernels
**Speedup relative to Intel Duo**

![Graph showing speedup relative to Intel Duo](image)

*Figure 7. Speedup from Execution on GMA X3000 Exoservers over IA32 Sequencer*

**Impact of Data Copying vs. Shared Virtual Address Space**

![Graph showing impact of data copying vs. shared virtual address space](image)

*Figure 8. Impact of Shared Virtual Memory*
Cooperative Execution between IA32 and GMA

Power and Energy Considerations

- EPI (Energy Per Instruction) as a metric
- EPI for IA32 is ~ 10nJ
  —150W socket limits performance to below 15 GIPS
- EPI for GMA is ~ 0.3nJ
  —30x more power efficient than IA32
- Goal: reach 100 GIPS performance by judicious combination of general-purpose and special-purpose cores