Task-based Parallel H.264 Video Encoding for Explicit Communication Architectures

M. Alvanos, **G. Tzenakis**, D. S. Nikolopoulos, and A. Bilas

Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH)

SAMOS-XI 20-July-2011
Motivation and Goal

- Modern multi-core heterogeneous processors
  - Explicitly managed memory hierarchies
  - Require fine-grain parallelism
  - Non-trivial programming effort

- H.264 video encoding
  - Essential application for modern embedded systems
  - Extremely demanding for high definition videos

- Our Goal: Investigate the parallelization of a video encoder
  - Task-based parallelism
  - Performance
  - Programming effort
  - IBM Cell processor
Previous work

- Offload only the analysis phase [Park and HA, ICASSP ’06]
- Focus only on kernel optimization [Di Wu et al., ICSPCS ’08]
- Decentralized pipelined parallel encoder [Xun et al., ISCAS ’09]
H.264 encoder

Input Frame

- Input stream of raw frames
- Frames divided into macroblocks
- Tasks per macroblock
  - Analyze and encode
    - Spatial and temporal model
  - Entropy encoding
    - Further encode the encoded stream
  - Deblocking filter
    - Reduce blocking distortion on reconstructed frames
Macroblock Dependencies

- Analysis results from neighboring macroblocks
- Current macroblock uses pixels from neighboring macroblocks
- Deblocking filter uses the pixel values of neighboring left and upper macroblocks
- Entropy encoding implies frame scan order dependencies
Frame types

- **Intra-type (I-frame):**
  - Use spatial model only
  - Needs macroblocks only from current frame

- **Predicted-type (P-frame):**
  - Use temporal redundancies from previous frames

- **Bidirectional-type (B-frame):**
  - Use temporal redundancies from previous and next frames
  - Not used as reference frame
What is important?

- Normalized serial execution breakdown of x264 encoder
- Analyze and encode: 85% of execution time
- Input video resolution does not affect execution breakdown
1. Introduction
2. Design
3. Experimental methodology
4. Evaluation
5. Conclusion
Basic structure

- Master-worker model
- PPE issues tasks to be executed on SPEs
- SPEs perform DMA transfers for task data
Analyze/Encode task

Input Frame

Reconstructed Frame

DMA IN

DMA OUT

Analyse/encode task

SPE workers

PPE

Task issue

Completion

Task-based Parallelism for H.264
Entropy encode task

Input Frame

Entropy task

Analyse/encode task

DMA IN

SPE workers

SPE

SPE

... SPE

PPE

Task issue

Completion

Dependencies check

Reconstructed Frame

DMA OUT

State Table
Deblocking filter task

**Input Frame**

- Entropy task
- Analyse/encode task
- Deblocking filter task

**Reconstructed Frame**

- DMA IN
- DMA OUT

- SPE workers
- SPE
- PPE

State Table

Dependencies check

Task issue

Reference Frames

**Task-based Parallelism for H.264**
Optimizations

- Memory optimizations
  - Pad macro-blocks in main memory
  - DMA transfers aligned to 128-bytes
  - Eliminate 2KByte strided DMA transfers to avoid memory bank conflicts
  - 16MBytes memory pages to reduce TLB misses
- SPE: SIMD processor, can’t handle scalar code efficiently
  - SPE kernels optimizations based on PowerPC vectorized versions from original x264
Introduction

Design

Experimental methodology

Evaluation

Conclusion
Platform and Workloads

- PlayStation3
  - Cell processor @3.2 GHz
  - 256 MBytes of external memory

- Video workloads
  - HD-VideoBench
  - Two resolutions: 576p (small) and 1080p (large)

- Examine:
  - Overall speedup
  - Impact of optimizations
  - Dynamic behavior
  - Programming effort
1 Introduction

2 Design

3 Experimental methodology

4 Evaluation

5 Conclusion
Overall Speedup

- Speedup compared to serial PPE
- From 4.7x up to 8.6x
- Riverbed video has better computation/communication ratio

- Temporal model can’t reduce redundancy information
- More work for spatial model
- Produces larger stream for entropy encoding
Optimizations

- Large pages have improve serial execution by 8%
- Dynamic scheduling
  - Average idle time drops by half on 6 SPEs
  - Improves total time by 30%
- Memory optimizations
  - Average communication time drops by half on 6 SPEs
  - Reduces total time by another 25%
Dynamic behavior

- Analyze/encode task: 45K - 120K cycles
- Entropy encode task: 1.5K - 50K cycles
Programming effort

<table>
<thead>
<tr>
<th>Category</th>
<th>LOC</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privatize/Offload</td>
<td>6400</td>
<td>30%</td>
</tr>
<tr>
<td>Task-parallelsim</td>
<td>700</td>
<td>3%</td>
</tr>
<tr>
<td>Mem/SPE optimizations</td>
<td>900</td>
<td>4%</td>
</tr>
</tbody>
</table>

- Initial code base of 22000 LOC
- Privatize global accesses in SPE offloaded code
  - Compiler transformation can relieve programmers
- Exploiting parallelism
  - Preserving dependencies and scheduling tasks to SPEs requires architecture dependent code
  - Although techniques are portable, code isn’t
  - However, compiler or runtime support for dependency analysis could mitigate this effort
Conclusions

- Designed, implemented and evaluated a H.264 encoder on Cell processor
- Speedup up to 8.6x on six SPEs over serial PPE execution
- Evaluated impact of optimizations
  - Dynamic scheduling improves execution time by 30%
  - Memory optimizations further improves by another 25%
- Analyzed the programming effort associated with parallelizing H.264 encoder for a heterogeneous architecture
QUESTIONS ?
Queue size impact

- Three configurations
  - Optimized with one slot (left)
  - PPE thread for entropy (middle)
  - Entropy thread with 4 slots (right)
- Metadata handling time increase
- Recycling increases (locks)
- Stall time becomes sync time
- Limited prefetching
- Total execution time increases due to imbalance
Local Store Code Breakdown

- Binary: 155 Kbytes
- BSS segment: 12.8 KB

Code Segment
- Analyze: 60 KB
- Macroblock encode 43 KB
- Runtime system: 12.7 KB
Limitations

- 2D-wavefront only
  - Inter-frame: requires postponing (moving) the metadata handling on the next frame
  - Quantization tables for different frame types
- Less flexibility in encoding options
  - 16x6 block size for motion estimation/compensation
  - Inter-frame encoding only in B-frames
  - Limited search window size to 128x128
- Quality can be improved by using the missing components
  - Rate-distortion optimization
  - Smaller matching blockings
- Using only the CAVLC encoder

Task-based Parallelism for H.264
Comparison with other platforms

- Two 64-bit Dual-Core AMD Opteron Processor 2216 running at 2.4GHz
  - 64 KB L1-D and 64KB L1-I cache
  - 1024KB L2 cache per core
- The performance is similar
  - Both implementations are well optimized

<table>
<thead>
<tr>
<th>Resolution</th>
<th>x264 Cell</th>
<th>1 thread x86</th>
<th>2 threads x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>720x576</td>
<td>55.2 fps</td>
<td>40.27 fps</td>
<td>60.9 fps</td>
</tr>
<tr>
<td>1920x1088</td>
<td>10.2 fps</td>
<td>7.12 fps</td>
<td>12.5 fps</td>
</tr>
</tbody>
</table>