A Near-Memory Processor (NMP) for Vector, Streaming, and Bit Manipulation Workloads

Mingliang Wei, Marc Snir, Josep Torrellas, and R. Brett Tremaine

University of Illinois and IBM

February 2005
Motivation

- Memory system bandwidth is expensive

- Many scientific/multimedia apps
  - Are memory-bandwidth limited (caches hardly work)
  - Could use vectors
  - Manipulate bits

- Commodity processors not optimized for this!
Approach

• Previous talk: NMP for prefetching

• This design: NMP to off-load computation from main proc
  – Vector
  – Streaming
  – Bit Manipulation

• Design is very simple

• Near Memory Processor (NMP):
  – Optimized for bandwidth (e.g. no caches)
  – Not necessarily physically closer to memory
Near Memory Processor

• NMP: Simple multithreaded proc with vector/stream/bit support
  – High performance: tolerates high variability in:
    • Load/store latency of vector elements
    • Relative speed of communicating streams
  – Simple, compact design
  – Easy to program
  – Fairly general purpose
  – Can be on the main processor chip or closer to memory

• Uses scratchpad: flexible memory area to store stream buffers, vectors, and other
Vectors, Streams and Multithreading

- Vectors
  - Have parallelism
  - Caches work sub-optimally

- Streams
  - Have parallelism
  - Exploit producer/consumer
  - Space multiplexing the hardware is inefficient

- Blocked Multithreading
  - Tolerates variability in LD/ST latency of vector elements
  - Tolerates variability in relative speed of communicating streams
  - Simple implementation, efficient hardware use
Multithreading in the NMP

- Context switch when processor expects long idle time
  - Scratchpad miss
  - Stalled on synch (e.g. fast producer stream kernel)
- Only a few contexts needed
- Do not save much state on context switch:
  - Scratchpad is not saved
System Architecture

![System Architecture Diagram]

- Processor
- L2 Cache
- Fabric Controller
- Memory Controller
- Memory
- L3 Cache

Wei/Snir/Torreallas: Near Memory Processing
NMP Architecture

- In-order core
Core Architecture
Scratchpad

- High-bandwidth local memory
- Addressed with virtual addresses
- Shared by all threads running on the NMP
- Not saved on context switch
- Very efficient synchronization: Full/Empty bit per byte
- May suffer page misses
- Pages are lazily paged out to memory
Bit Manipulation

• Has Bit Matrix Register (BMR) of Cray
• Typical instructions:
  – Bmm_load: load 64x64 bit matrix
  – Bmm: bit multiply vector or scalar with the matrix in the BMR
  – Leadz
  – Popcnt
  – Sshift
  – Mix
Other Issues

- Interface with Main Processor (MP)
  - Start: MP stores into a mem-mapped location (no syscall)
  - End: NMP sets memory flag; MP polls

- Exceptions in NMP
  - Virtual memory exceptions:
    - Not precise but restartable and handled in SW
  - Each Scratchpad byte has a bit that records exceptions
  - Vector and stream buffers in Scratchpad cannot cross pages
Programming Environment

• NMP is attached to main program via system call
  \[
  NMP\_handler = NMP\_connect (\text{ObjectAddress})
  \]
  \[
  Status = NMP\_disconnect (NMP\_handler)
  \]

• NMP is invoked from main program via asynchronous (user space) method invocation
  – Processor stores parameters at NMP “doorbell” and polls flag
    \[
    Status = \text{Memthread\_create} (\text{FunctionInvoked}, \text{Params},
    \text{CompletionFlag}, NMP\_handler)
    \]
    \[
    \text{Memthread\_wait} (\text{CompletionFlag}) \text{ or } \text{Memthread\_poll} (\text{CompletionFlag})
    \]
  – NMP sets completion flag
    \[
    \text{Memthead\_end} (\text{CompletionFlag})
    \]
  – Fits X10 Asynchronous expressions and futures
Possible Programming Support

- Trivial: “hand-coded” libraries for special applications
- Easy: libraries separately compiled by vectorizing compiler
- Research: compiler that splits code into main-processor part and NMP part
  - Can leverage previous work at IBM and UIUC (FlexRAM)
Architecture Evaluated

Processor
  L1 Cache
  L2 Cache
  Fabric Controller
  NMP
  Memory Controller
  Memory

chip
# Simulation Parameters

## NMP Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>4GHz in-order</td>
</tr>
<tr>
<td>Issue Width</td>
<td>2</td>
</tr>
<tr>
<td># Scalar FUs</td>
<td>1Int FU, 1FP FU</td>
</tr>
<tr>
<td># Vector FUs</td>
<td>16Int FUs, 16FP FUs</td>
</tr>
<tr>
<td># Lanes</td>
<td>16</td>
</tr>
<tr>
<td># Pending Memory Ops (Ld, St.)</td>
<td>128, 128</td>
</tr>
<tr>
<td># Contexts</td>
<td>4</td>
</tr>
<tr>
<td>Time to Context Switch</td>
<td>4 cycles</td>
</tr>
</tbody>
</table>

## Memory Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1, L2, Scratchpad size</td>
<td>32KB, 1MB, 64KB</td>
</tr>
<tr>
<td>L1, L2 associativity</td>
<td>2-way, 4-way</td>
</tr>
<tr>
<td>L1, L2 line size</td>
<td>64B, 64B</td>
</tr>
<tr>
<td>Main proc. to L1, L2, memory round-trip latency</td>
<td>2, 10, 500 cycles</td>
</tr>
<tr>
<td>NMP to Scratchpad, memory latency</td>
<td>6, 470 cycles</td>
</tr>
</tbody>
</table>

## Main Processor Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>4GHz out-of-order</td>
</tr>
<tr>
<td>Fetch Width</td>
<td>8</td>
</tr>
<tr>
<td>Issue Width</td>
<td>4</td>
</tr>
<tr>
<td>Retire Width</td>
<td>8</td>
</tr>
<tr>
<td>ROB size</td>
<td>152</td>
</tr>
<tr>
<td>I-window size</td>
<td>80</td>
</tr>
<tr>
<td>Int FUs</td>
<td>3</td>
</tr>
<tr>
<td>FP FUs</td>
<td>3</td>
</tr>
<tr>
<td>Mem FUs</td>
<td>3</td>
</tr>
<tr>
<td>Pending Ld/St</td>
<td>16, 16</td>
</tr>
<tr>
<td>Branch Pred.</td>
<td>Like Alpha 21464</td>
</tr>
<tr>
<td>Branch Penalty</td>
<td>14 cycles</td>
</tr>
</tbody>
</table>
Simulator Infrastructure

- Architecture simulator from UIUC (SESC)
- Models ooo processors and MP memory hierarchies
- 110 K lines of C++ source code

Simulating an R4400 at 150Mhz (SGI IP22):

<table>
<thead>
<tr>
<th>Bench</th>
<th>Native</th>
<th>Simulated</th>
<th>Error</th>
<th>SimTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix</td>
<td>291ms</td>
<td>282ms</td>
<td>-4%</td>
<td>27s</td>
</tr>
<tr>
<td>crafty</td>
<td>265ms</td>
<td>271ms</td>
<td>+4%</td>
<td>43s</td>
</tr>
<tr>
<td>mcf</td>
<td>2438ms</td>
<td>2422ms</td>
<td>-1%</td>
<td>174s</td>
</tr>
<tr>
<td>mp3dec</td>
<td>2347ms</td>
<td>2521ms</td>
<td>+7%</td>
<td>185s</td>
</tr>
<tr>
<td>lat_mem</td>
<td>4005ms</td>
<td>4129ms</td>
<td>+3%</td>
<td>279s</td>
</tr>
</tbody>
</table>
# Applications

<table>
<thead>
<tr>
<th>Name</th>
<th>Vec</th>
<th>Stm</th>
<th>Bit</th>
<th>#Thd</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rgb2yuv</td>
<td>X</td>
<td></td>
<td>4</td>
<td></td>
<td>Convert the RGB presentation to YUV</td>
</tr>
<tr>
<td>ConvEnc</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>3</td>
<td>Convolutional encoder</td>
</tr>
<tr>
<td>BMT</td>
<td></td>
<td></td>
<td>X</td>
<td>4</td>
<td>Bit matrix transposition</td>
</tr>
<tr>
<td>BT</td>
<td></td>
<td>X</td>
<td>X</td>
<td>3</td>
<td>Bit twiddle</td>
</tr>
<tr>
<td>3DES</td>
<td>X</td>
<td></td>
<td></td>
<td>4</td>
<td>3DES encryption</td>
</tr>
<tr>
<td>PartRadio</td>
<td>X</td>
<td>X</td>
<td></td>
<td>3</td>
<td>Partial radio station</td>
</tr>
</tbody>
</table>

- Code length: 730 lines/app (average)
Mapping of Applications

- Temporal Locality
- Spatial Locality

Applications:
- BMT
- BT
- 3DES
- PartRadio
- ConvEnc
- Rgb2Yuv
NMP Speedup Over Main Processor

Wei/Snir/Torrellas: Near Memory Processing
NMP Speedup Over Main Processor

Weaver/Snir/Torrellas: Near Memory Processing
Results

- Nmp much faster than main: geometric mean speedup of 4.9

- Vector support is key to the speedups in many vectorizable apps (Rgb2yuv, 3DES, and PartRadio)

- Bit manipulation support is key in some apps (BMT and BT)

- ConvEnc needs both vectorization and bit manipulation support
Adding the Stream App

![Bar Chart]
Adding the Stream App

![Chart showing performance metrics for various applications and operations, including V+S+B, B, S+B, and V, with different algorithms like nmp, novec, nobit, nomt, and none.](chart_image)
Conclusions

• Selected one NMP design point:
  – Off-load vector, streaming, bit manipulation ops
  – Simple/compact design
  – Fairly general purpose core

• Evaluated design assuming NMP on processor chip

• High performance for high-bandwidth applications:
  – 4.9 x faster than main processor
  – Most cost-effective support in the NMP:
    • Vectors
    • Bit manipulation
Future Work

• Focusing on programmability:
  – Programming interface
  – Compiler support

• Interaction with the OS

• Step back: How far can we go with PowerPC ISA (and small extensions)?
  – No vector compiler
  – Libraries that perform efficient bit manipulation
Near Memory Processing (NMP)

Mingliang Wei, Marc Snir, Josep Torrellas and R. Brett Tremaine
University of Illinois and IBM

February 2005
RGB2YUV (Vector)

- Input: 1000 x 200 pixels, output is the same size
- Conversion of each pixel is independent
- 4 threads, each processes a partition of the input
- Each thread processes 16 pixels in parallel
BMT (Bit manip)

- Transpose 4 1024x1024 bit matrices
- 4 threads, each trans. 1 matrix
- Threads use:
  - Vector load to load data
  - Bmm_vec to trans. a tile
  - Vector store
3DES (Vector)

• Input: set of 64-bit word values. Output is same-size encrypted word values.
• Vectorized the load/store and the computation
• Counter mode (more parallelism)
• 4 threads, each work on a partition of the input (like RGB2YUV)
• Each thread processes 16 elements in parallel
• Bmm to perform initial and final permutation (though a small fraction of total execution time)
**ConvEnc**  *(Vector + stream + bit)*

- Input: bit stream; output: 2x bit stream
- Encode 64 bytes at a time
- Shift instruction to shift 64 bytes of data
- Mix instruction to interleave two blocks of bits
BT (Stream + bit manip)

Generate \( \{A_i\} \)
- \( \{B_i\} = \text{take 5, drop 6, take 5 \ldots \text{from} \ \{A_i\} \)
- \( \{C_i\} = \text{drop 5, take 5, drop 6, take 5, drop 6 from} \ \{A_i\} \)
- \( \{D_i\} = (C_i \lor \neg (B_{i+1})) \oplus (\neg (C_i) \lor B_{i-1}) \)
- \( \{E_i\} = D_i \oplus D_{i+37} \oplus D_{i+100} \)

Identify sequences of 0s in E that are longer than 100

- Three threads in the NMP: Generator, Splitter, Counter
- Bmm to split the \( \{A_i\} \)
PartRadio \hspace{0.5cm} (Vector + stream)

- Input: a float-point stream. Output is a float-point stream
- 3 threads (like ConvEnc)
  - Low pass filter
  - Demodulator
  - Equalizer
- Thread processes 16 elements in parallel (vector)