AltiVec Technology Programming Model

FTF-NET-F0139

Chuck Corley | DMTS

APR. 2014
Agenda

• Benchmarking
  – Dhrystone w/AltiVec
  – CoreMark
• AltiVec Enhancements
• MEPL library
• E6500 Overview – selected highlights
Benchmarks: Dhrystone
Dhrystone 2.1
Preliminary Performance Results

<table>
<thead>
<tr>
<th>Binary</th>
<th>P4080 e500mc DMIPS/MHz</th>
<th>P5020 e5500 DMIPS/MHz</th>
<th>T4240 e6500 DMIPS/MHz</th>
<th>e5500 / e500mc speedup</th>
<th>e6500 / e500mc speedup</th>
<th>e6500 / e5500 speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>dhry21.gcc471.e500mc.m32</td>
<td>2.700</td>
<td>2.960</td>
<td>3.252</td>
<td>10%</td>
<td>20%</td>
<td>10%</td>
</tr>
<tr>
<td>dhry21.gcc471.e5500.m32</td>
<td></td>
<td></td>
<td>3.257</td>
<td>4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dhry21.gcc471.e6500.m32</td>
<td></td>
<td></td>
<td>3.252</td>
<td>4%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dhry21.gcc471.e6500.m32av</td>
<td></td>
<td></td>
<td>3.761</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dhry21.gcc471.e5500.m64</td>
<td></td>
<td></td>
<td>3.418</td>
<td>3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dhry21.gcc471.e6500.m64</td>
<td></td>
<td></td>
<td>3.421</td>
<td>2%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dhry21.gcc471.e6500.m64av</td>
<td></td>
<td></td>
<td>3.682</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Performance results obtained on T4240 Rev1 Silicon
- First row shows the results of running the same binary on e500mc, e5500, and e6500. It shows how the e6500 and e5500 micro-architectural changes have improved performance over e500mc
- e6500 and e5500 result is better than e500mc
- e6500 consistently better than e5500 on all binaries
- 64-bit and AltiVec optimizations for memcpy and strcmp further improve performance
- Dhrystone doesn’t comprehend threading. It is inherently single threaded.

**Conclusion:** e6500’s first thread outperforms e500 and e5500.
Benchmarks: CoreMark
CoreMark Scalability Notes

- Multi-threaded CoreMark was measured under Linux on T4240 Rev1 silicon
- Placement of processes to CPUs was left to decision of OS scheduler
- Linux OS scheduler currently does not distinguish e6500 HW threads versus cores
- Less-than-linear scalability probably due to two benchmark threads running on the two threads of the same core
- More linear scalability can be achieved by manually defining processor affinity using taskset or running with HW threads disabled (right most column)

- **Conclusions** (from table on next page): CoreMark scales very close to linear with one task per core. CoreMark scales almost as well with two tasks per core. Two threads is almost 2x the performance as one thread.
e6500 CoreMark Intra-cluster Scalability

- Use taskset to specify processor affinity (which cores allowed to run benchmark)
- Linear Alloc: assign CPUs to adjacent CPUs and threads
- Spread Alloc: avoid bunching threads up on a core whenever possible
- Conclusion: two threads on one core almost as good as two cores one thread each.
# CoreMark 1.0 – Scalability

Preliminary Performance Results

<table>
<thead>
<tr>
<th>System</th>
<th>P4080-1500-700-1300</th>
<th>P5020-1500-700-1300</th>
<th>T4240-1500-700-1300</th>
<th>T4240-1500-700-1300 with one thread used per core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>e500mc-32b-gcc471-r721</td>
<td>e5500-32b-gcc471-r721</td>
<td>e6500-32b-gcc471-r721</td>
<td></td>
</tr>
<tr>
<td># copies</td>
<td>CoreMarks</td>
<td>speedup vs. n=1</td>
<td>CoreMarks</td>
<td>speedup vs. n=1</td>
</tr>
<tr>
<td>1</td>
<td>5707</td>
<td>1.00</td>
<td>5900</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>11584</td>
<td>2.03</td>
<td>11535</td>
<td>1.96</td>
</tr>
<tr>
<td>3</td>
<td>17301</td>
<td>3.03</td>
<td>11399</td>
<td>1.93</td>
</tr>
<tr>
<td>4</td>
<td>23085</td>
<td>4.05</td>
<td>11765</td>
<td>1.99</td>
</tr>
<tr>
<td>5</td>
<td>28334</td>
<td>4.96</td>
<td>11651</td>
<td>1.97</td>
</tr>
<tr>
<td>6</td>
<td>34005</td>
<td>5.96</td>
<td>11569</td>
<td>1.96</td>
</tr>
<tr>
<td>7</td>
<td>39660</td>
<td>6.95</td>
<td>11779</td>
<td>2.00</td>
</tr>
<tr>
<td>8</td>
<td>45328</td>
<td>7.94</td>
<td>11570</td>
<td>1.96</td>
</tr>
<tr>
<td>12</td>
<td>42972</td>
<td>7.53</td>
<td>11788</td>
<td>2.00</td>
</tr>
<tr>
<td>16</td>
<td>46346</td>
<td>8.12</td>
<td>11786</td>
<td>2.00</td>
</tr>
<tr>
<td>24</td>
<td>45594</td>
<td>7.99</td>
<td>11581</td>
<td>1.96</td>
</tr>
<tr>
<td>@ best freq</td>
<td>46346</td>
<td>@1500 MHz</td>
<td>15717</td>
<td>@2000 MHz</td>
</tr>
</tbody>
</table>

@ best freq | 46346               | @1500 MHz           | 15717               | @2000 MHz                                       | 179763         |

Best result
AltiVec Features

• Provides SIMD (Single Instruction, Multiple Data) functionality for embedded applications with massive data processing needs
• Key elements of Motorola’s AltiVec technology include:
  - 128-bit vector execution unit with 32-entry 128-bit register file
  - Parallel processing with Vector permute unit and Vector ALU
  - 162 new instructions
  - New data types:
    ▪ Packed byte, halfword, and word integers
    ▪ Packed IEEE single-precision floats
  - Saturation arithmetic
• Simplified architecture
  - No interrupts other than data storage interrupt on loads and stores
  - No hardware unaligned access support
  - No penalty for running AltiVec and standard PowerPC instructions simultaneously
  - Streamlined architecture to facilitate efficient implementation
• Maintains PowerPC architecture’s RISC register-to-register programming model
AltiVec Features

• Supports parallel operation on byte, halfword, word and 128-bit operands
  - Intra and inter-element arithmetic instructions
  - Intra and inter-element conditional instructions
  - Powerful Permute, Shift and Rotate instructions
• Vector integer and floating-point arithmetic
  - Data types
    ▪ 8-, 16- and 32-bit signed- and unsigned-integer data types
    ▪ 32-bit IEEE single-precision floating-point data type
    ▪ 8-, 16-, and 32-bit Boolean data types (e.g. 0xFFFF % 16-bit TRUE)
  - Modulo & saturation integer arithmetic
  - 32-bit “IEEE-default” single-precision floating-point arithmetic
    ▪ IEEE-default exception handling
    ▪ IEEE-default “round-to-nearest”
    ▪ fast non-IEEE mode (e.g. denorms flushed to zero)
• Control flow with highly flexible bit manipulation engine
  - Compare creates field mask used by select function
  - Compare Rc bit enables setting Condition Register
    ▪ Trivial accept/reject in 3D graphics
    ▪ Exception detection via software polling
AltiVec’s Vector Execution Unit

- Concurrency with integer and floating-point units
- Separate, dedicated 32 128-bit vector registers
  - Larger namespace = reduced register pressure/spillage
  - Longer vector length = more data-level parallelism
  - Separate files can all be accessed by execution units in parallel
  - Deeper register files allow more sophisticated software optimizations
- No penalty for mingling integer, floating point and AltiVec technology operations
128-bit Vector Architecture

• 128-bit wide data paths between L1 cache, L2 cache, Load/Store Units and Registers
  - Wider data paths speed save and restore operations

• Offers SIMD processing support for
  - 16-way parallelism for 8-bit signed and unsigned integers and characters
  - 8-way parallelism for 16-bit signed and unsigned integers
  - 4-way parallelism for 32-bit signed and unsigned integers and IEEE floating point numbers

• Two fully pipelined independent execution units
  - Vector Permute Unit is a highly flexible byte manipulation engine
  - Vector ALU (Arithmetic Logical Unit) performs up to 16 operations in a single clock cycle
    ▪ Contains Vector Simple Fixed-Point, Vector Complex Fixed-Point, and Vector Floating-Point execution engines
  - Dual Altivec instruction issue: One arithmetic, one “permute”
AltiVec Enhancements on the e6500 core
Agenda

• Overview of AltiVec enhancements in e6500 core
• More detail on each change
• Some limitations
• Unaligned loads and stores in-depth - memcpy
AltiVec Enhancements in e6500 core

- AltiVec e6500 core technology is essentially the same as AltiVec technology from the 74xx processors except the following:
  - Adds new instructions for computing absolute differences
    - `vabsdub` – absolute differences (byte)
    - `vabsduh` – absolute differences (halfword)
    - `vabsduw` – absolute differences (word)
  - Adds new instructions for moving data from GPRs to VRs
    - `mvidsplt <64>` and `mviwsplt` move data from 2 GPRs into a vector register
  - Adds new instructions for dealing with misaligned vectors more easily
    - `lvtlx[l], lvtrx[l], stvflx[l], stvfrx[l]` – load/store vector to/from left [LRU]
    - `lvswx[l], stvswx[l]` – load/store vector with left/right swap [LRU]
  - Adds new instructions for dealing with elements of vectors
    - `lvexbx, stvebx` – load/store vector element indexed byte
    - `lvexhx, stvehx` – load/store vector element indexed halfword
    - `lvexwx, stvewx` – load/store vector element indexed word
    - These allow loading/storing of arbitrary elements to arbitrary addresses
<table>
<thead>
<tr>
<th>Feature</th>
<th>Original AltiVec Definition</th>
<th>Power ISA AltiVec Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little-endian</td>
<td>Supported</td>
<td>Not Supported</td>
<td>Little-endian byte ordering is not supported on Power ISA AltiVec definition.</td>
</tr>
<tr>
<td>Data stream instructions</td>
<td>Supported</td>
<td>Not Supported</td>
<td>dss, dssall, dst, dstt, dstst, and dststt instructions are not supported on Power ISA AltiVec definition.</td>
</tr>
</tbody>
</table>
## Summary of Changes from Original AltiVec Definition

<table>
<thead>
<tr>
<th>Feature</th>
<th>Original AltiVec Definition</th>
<th>Power ISA AltiVec Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVORs</td>
<td>Not Supported</td>
<td>Supported</td>
<td>IVORs added for AltiVec unavailable interrupt and AltiVec assist interrupt.</td>
</tr>
<tr>
<td>Move from GPR to VR</td>
<td>Not Supported</td>
<td>Supported</td>
<td>mvidsplt &lt;64&gt; and mviwsplt instructions move data from 2 GPRs into a vector register.</td>
</tr>
<tr>
<td>Absolute differences</td>
<td>Not Supported</td>
<td>Supported</td>
<td>Absolute difference instructions vabsdub, vabsduh, and vabsduw compute the unsigned absolute differences. These are useful for motion estimation in video processing.</td>
</tr>
</tbody>
</table>
## Summary of Changes from Original AltiVec Definition

<table>
<thead>
<tr>
<th>Feature</th>
<th>Original AltiVec Definition</th>
<th>Power ISA AltiVec Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended support for misaligned vectors</td>
<td>Not Supported</td>
<td>Supported</td>
<td>Load vector to left and right (lvtlx[l], lvtrx[l]), load vector with left-right swap (lvswx[l]), load vector for swap merge (lvsm). Store vector from left and right (stvflx[l], stvfrx[l]), store vector with left-right swap (stvswx[l]).</td>
</tr>
<tr>
<td>Extended support for handling head and tail of vectors</td>
<td>Not Supported</td>
<td>Supported</td>
<td>Load vector element indexed [byte, half-word, word] indexed (lvexbx, lvexhx, lvexwx) loads specified elements from an arbitrary address zeroing the rest of the register. Store vector element indexed [byte, half-word, word] indexed (stvexbx, stvexhx, stvexwx) stores specified elements to an arbitrary address.</td>
</tr>
</tbody>
</table>
## Summary of Changes from Original AltiVec Definition

<table>
<thead>
<tr>
<th>Feature</th>
<th>Original AltiVec Definition</th>
<th>Power ISA AltiVec Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External PID instructions for loading and storing VRs</td>
<td>Not Supported</td>
<td>Supported</td>
<td>Load and store vector by external PID ($\text{ivp}x[l]$, $\text{stvp}x[l]$) for moving data efficiently across address spaces.</td>
</tr>
</tbody>
</table>
Impact of No Support for Data Stream Instructions

- **dss** (Data Stream Stop), **dssall** (stop ALL prefetch engines), **dst** (Data Stream Touch), **dstt** (Touch Transient), **dstst** (Data Stream Touch for STore), and **dststt** (Touch for Store Transient), instructions were present in the first definition of AltiVec technology for PowerPC processors. These instructions provided software initiated streaming prefetch controls.

- In Power ISA these instructions are no longer defined, and streaming is performed by variants of the **dcbt** instruction or by hardware prefetchers. Cache stashing could be considered an alternative as well.

- For Freescale EIS, these instructions are treated as no-ops since they may be present in older code and do not change architectural state.
Impact of Adding IVORS Added for AltiVec Interrupts

- IVORS added for AltiVec unavailable interrupt (IVOR32) and AltiVec assist interrupt (IVOR33).

- Original AltiVec technology on the e600 core included an AltiVec unavailable exception. IVORS are the equivalent exception mechanism in e500-based cores.

- The AltiVec unavailable interrupt occurs when an attempt is made to execute an AltiVec instruction and MSR[SPV] = 0.
  - This can be useful in reducing context switch overhead by not saving AltiVec registers unless a process actually uses AltiVec instructions.

- The AltiVec assist interrupt occurs when no higher priority exception exists and a de-normalized floating-point number is an operand to an AltiVec floating-point instruction requiring software assist. An AltiVec assist exception is presented to the interrupt mechanism. The instruction handler is required to emulate the interrupt causing instruction to provide correct results with the denormalized input.
  - In Original AltiVec Java mode, denormalized inputs were accepted and gradual underflow results were generated. In non-Java mode, denormalized inputs are (quietly) flushed to the correctly signed zero (±0) before being used in an instruction.
  - In general, AltiVec vector instructions generate very few exceptions.
Impact of Moving GPRs into a Vector Register

• This was a source of frustration in original AltiVec technology. The explanation was that the interconnect between GPRs and VPRs was not warranted when data could be moved quickly via store and load from L1 cache. Still, the capability was desired by many customers.

• These new instructions will make it simpler to program with AltiVec.
  – mvidsplt vD,rA,rB - Move to Vector from Integer Double Word and Splat <64> Place the contents of rA into high-order 64 bits of vD and place the contents of rB into the low-order 64 bits of vD.
  – mviwsplt vD,rA,rB - Move to Vector from Integer Word and Splat Place the contents of the low-order 32 bits of rA concatenated with low-order 32 bits of rB into the low-order and high-order 64 bits of vD.
Impact of Absolute Difference Instructions

- \texttt{vabsdu\{b|h|w\} vD,vA,vB} - Vector Absolute Difference Unsigned \{Byte | Half Word | Word\} - Each integer element in \(vB\) is subtracted from the corresponding integer element in \(vA\). The elements of \(vA\) and \(vB\) are treated as unsigned integers. The absolute value of the result is placed into the corresponding element of \(vD\).

- These are useful for motion estimation in video processing.
e6500 AltiVec Limitations

- AltiVec e6500 limitations
  - Operates in big-endian only
  - Does not have data streaming (\texttt{dst} type instructions)
    - They are executed as NOPs
  - The AltiVec execution units are shared between threads (but the registers are private)
- AltiVec execution units will go drowsy (reducing static power) when not used
  - Comes out of drowsy automatically when an AltiVec instruction is executed and the MSR AltiVec available bit is set.
  - All AltiVec state is retained when the unit is drowsy
- \texttt{stvflx} and \texttt{stvfrx} will take an alignment exception for cache-inhibited stores of >8 bytes.
Impact of New Load and Store Instructions

• Reduces the effort to load and store unaligned (not quad-word aligned) data
• Reduces number of registers needed for permute and mask vectors
• Reduces the effort to deal with the head and tail of unaligned strings or vector arrays
• Improves performance through:
  – Fewer instructions
  – Less register pressure
  – Less context to save
• Makes programming AltiVec technology simpler
Unaligned loads and stores in-depth - memcpy

- Reference H1109 Implementing and Using the Motorola AltiVec Libraries, Smart Network Developer’s Forum 2003
- Detailed explanation of memcpy using permutation
- Became libmotovec.a with libc functions:
  - memcpy, bcopy, memmove, memset, bzero, strcmp, strlen, memcmp, strcpy, __copy_tofrom_user_vec, __clear_user_vec, csum_partial_vec, csum_partial_copy_generic_vec.
- Motivated the Power AltiVec MEPL (Mentor Embedded Performance Library)
MEPL Library of AltiVec Functions
Mentor Embedded Performance Library (MEPL) for Freescale’s Altivec Technology

- Optimized for QorIQ AMP Series Altivec Technology

- Over 800 math and signal processing functions
  - Element-wise Vector Functions
  - Reduction Functions
  - Real/Complex Conversion Functions
  - FIR and IIR Filter Functions
  - 1D and 2D FFT, Convolution & Correlation Functions
  - General Signal and Image Processing Functions
  - Linear Algebra Functions

---

News Release

Freescale Selects Mentor Graphics to Develop Software Libraries for Enhanced Altivec Engine

Highly advanced software intended to optimize performance of Freescale’s QorIQ AMP processors

AUSTIN, Texas, Sep 06, 2011 (BUSINESS WIRE) --

Freescale Semiconductor (NYSE:FSL) has selected Mentor Graphics Corporation (NASDAQ: MENT) to develop a software library of advanced mathematical and signal processing functions optimized for the latest version of Freescale’s Altivec processing engine. The Mentor(R) Embedded Performance Library for Freescale’s Altivec technology is designed to help Freescale’s recently announced QorIQ Advanced Multiprocessing (AMP) processors achieve maximum performance from the newest Altivec engine while simultaneously supporting software developer productivity.
Overview of AltiVec Library Contents

• Basic Linear Algebra Subprograms (BLAS)
  – Standard building blocks for performing basic vector and matrix operations
    ▪ Level 1: scalar, vector, & vector-vector operations
    ▪ Level 2: matrix-vector operations
    ▪ Level 3: matrix-matrix operations

• C Linear Algebra PACKage (CLAPACK) standard functions
  – 1300 linear algebra functions – most benefit from BLAS AltiVec optimizations

• Mentor Embedded Performance Library (MEPL)
  – Includes general signal processing, FIR and IIR, FFT, convolution and correlation, image processing, etc.

• All in binary form as part of MEPL
Using AltiVec Performance Library

• In your c source code:
  
  ```c
  #include <mepl.h> (possibly #include <mepl/cblas.h>?)
  ```

  Then provide data and call desired function in accordance with API described herein or in references.

• In your gcc compiler command line:
  
  ```bash
  -I/<install directory>/MEPL/bin/mepl-1.0/e6500-32/include
  ```

  Or for 64b:  
  ```bash
  -I/<install directory>/MEPL/bin/mepl-1.0/e6500-64/include
  ```

• In your gcc linker command line:
  
  ```bash
  -L/<install directory>/MEPL/bin/mepl-1.0/e6500-32/lib -lcblas -latlas -lmepl
  ```

  Or for 64b:  
  ```bash
  -L/<install directory>/MEPL/bin/mepl-1.0/e6500-64/lib -lcblas -latlas -lmepl
  ```
Matrix – Matrix multiply

• BLAS Level 3 operation – cblas_sgemm
• C operation: $C \leftarrow \alpha AB + \beta C$
• Implemented in scalar:

```c
for ( i = 0 ; i < m ; i++ ) {
    for ( j = 0 ; j < n ; j++ ) {
        for ( e = 0 ; e < k ; e++ ) {
            sum = sum + A[i][e]*B[e][j];
        }
        C[i][j] = sum;
        sum = 0;
    }
}
```

• Implemented from mepl library:

```c
    cblas_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, m, n, k, \alpha, A, lda, B, ldb, \beta, c, ldc);
```
Two-Dimensional Discrete Fourier Transform

- Fast Fourier Transform (FFT) Functions:
  
  \[
  mepl\_dft2d\_create\_cf, \ mepl\_dft2d\_compute\_ip\_cf, \ mepl\_dft2d\_destroy\_cf
  \]

- C Operation:

\[
Y_k \leftarrow \text{scale} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn} W_M^{mp} W_N^{qn} \quad \text{for } u = 0,1,\ldots,M-1; \text{ for } v = 0,1,\ldots,N-1
\]

Sequence of MEPL Code:

```c
mepl_dft2d_cf* mepl_dft2d_create_cf(mepl_length, mepl_length, mepl_dft_direction);
void mepl_dft2d_compute_ip_cf(mepl_dft2d_cf const*, mepl_cfloat*, mepl_stride);
void mepl_dft2d_destroy_cf(mepl_dft2d_cf*);
```
2D 256x256 Matrix Example

- Input:

- Output:
Examples (using input vector of 8192 elements)

• **Natural Logarithm**
  - Standard C Library Function completed in ~1,800,000 clock cycles
  - MEPL Function completed in ~80,000 clock cycles

**Conclusion**: Because MEPL uses AltiVec, the function was 23 times faster

• **Fill a Vector with Scalar Value**
  - C code completed vector fill in ~35,500 clock cycles
  - MEPL Function complete in ~4,300 clock cycles

**Conclusion**: Because MEPL uses AltiVec, this operation was performed 8 times faster
AltiVec is back! In e6500 core.

- MEPL library enables AltiVec acceleration by simple invocation of common library function.

![Acceleration w/AltiVec](image)

**Input Size:**
- 8 Elements
- 256 Elements
- 8192 Elements
e6500 Overview
Agenda

• E6500 overview
• Shared L2 cache
• Multithreading
• Memory subsystem enhancements
• MMU enhancements
• General enhancements
• Each thread: Superscalar, seven-issue, out-of-order execution/in-order completion, Branch units with a 512-entry, 4-way set associative Branch Target/History
• Execution units: 1 Load/Store Unit per thread, 2 Simple integer per thread, 1 Complex for integer Multiply & Divide, 1 Floating-point Unit, Altivec
• 64 TLB SuperPages, 1024-entry 4K Pages, 36 or 40-bit Physical Address

• 64-bit Power Architecture
• 28nm Technology
• Superset of e5500
• Two threads per core (SMT)
• Dual load/store units, one per thread
• Shared L2 in cluster of 4 cores (8 threads per cluster)
  - 2048KB 16-way, 4 Banks
  - High-performance eLink bus between to Ld/St and instruction fetch units
• Power
  - Drowsy core and caches
  - Power Mgt Unit
  - Wait-on-reservation instruction
• Enhanced MP Performance
  - Accelerated Atomic Operations
  - Optimized Barrier Instructions
  - Fast intra-cluster sharing
• Altivec SIMD Unit
• CoreNet BIU
  - 256-bit Din and Dout data busses
• 36/40-bit Real Address
  - 1 Terabyte physical addr. space
• Hardware Table Walk
• LRAT
  - Logical to Real Addr. translation mechanism for improved hypervisor performance
Observation:

- **Quad** decode/dispatch (two per thread)
- **Quad** completion (two per thread)
- 14 issue (7 per thread)
- 12 execution units
Shared L2 Cache
Shared L2 Cache Cluster

- Each L2 cache in the system has 4 e6500 cores attached
  - The cores “share” the L2 cache and are connected by a synchronous interface called eLink.
  - The combination of the L2 cache and the cores which are connected to it are called a core cluster or sometimes just a cluster. Everything in the core cluster other than the cores is internally called Kibo.
  - Kibo is also used to build core clusters with DSP cores (for example in the 9164).

- Each cluster contains a single CoreNet port which connects the cluster (and the cores) to the rest of the system.

- Each cluster also contain other core specific logic for the cores connected to it:
  - Reservations for each core thread
  - Logic specific to power management to awaken sleeping cores when messages are received (from msgsnd)
Shared L2 Cache

- L2 cache is inclusive of L1 caches
- L1 data caches are write through
- Coherency handled in L2 cache
- L2 cache can be partitioned
- Each bank can process requests simultaneously
- The L2 cache and its cores are called a cluster
- There can be multiple clusters connected via CoreNet
- Coherency among cores in a cluster is faster (lower latency)
- L2 cache protected by ECC on tags and data
- L1 caches protected by parity on tags and data
- Single bit errors are fully recoverable in any cache
General Enhancements
General Enhancements

• Improved branch prediction and additional link stack entries
• Pipeline improvements:
  – LR, CTR, **mfocrf** optimization (LR and CTR are renamed)
  – 16 entry rename/completion buffer
• New debug features:
  – Ability to allocate individual debug events between the internal and external debuggers
  – More IAC events
  – Other miscellaneous improvements
• Performance Monitor
  – Many more events, 6 counters per thread
  – Guest performance monitor interrupt
Conclusion

• Power Architecture e6500 core is high performance
• Second thread is almost the same as adding an additional core
• AltiVec is a powerful SIMD execution unit that is easily integrated with scalar operations and control code.
• Mentor Embedded Performance Library (MEPL) enables AltiVec functionality with less effort
• Compilers – even gcc – are improving at auto-vectorization.