nCUBE3 Integrated MPP Node Processor
Bob Duzett, Barry Keane
Naoki Kamiya, Stan Kenoyer, Steve Page, Russell Randolph
John Maneely

I. System/Node Architecture
II. Memory Hierarchy
III. Optimizing Data flow
IV. Optimizing Ctl flow
V. Interprocessor Communications
VI. Status/Performance

nCUBE3 System Architecture

MIMD Hypercube
Adaptive Routing

IO Node
PCI

Compute Node

Integrated Node
64-bit CPU
nCUBE3 Processing Node

Processor
- 345-pin CPGA; 1.85" x 1.85"
- 2.5M transistors
- 0.5 micron CMOS, 3-layer metal
- 3.3 volts
- 6-8 watts

SDRAMs
- 1 or 2 Banks; 9/18/36 SDRAMs
- 16Mbytes - 1Gbyte
- LVTTL - to 100MHz

nCUBE3 Single-chip Node Computer

<table>
<thead>
<tr>
<th>Component</th>
<th>die area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>56%</td>
</tr>
<tr>
<td>Instr Fetch &amp; Decode</td>
<td>12%</td>
</tr>
<tr>
<td>Opr Fetch &amp; Queueing</td>
<td>19%</td>
</tr>
<tr>
<td>Instr &amp; Data Caches</td>
<td>10%</td>
</tr>
<tr>
<td>64-bit Integer Execution</td>
<td>3%</td>
</tr>
<tr>
<td>64-bit FPU</td>
<td>9%</td>
</tr>
<tr>
<td>64-bit MMU</td>
<td>3%</td>
</tr>
<tr>
<td>Communications</td>
<td>20%</td>
</tr>
<tr>
<td>Adaptive Msg Router</td>
<td>2%</td>
</tr>
<tr>
<td>16 DMA Channels</td>
<td>8%</td>
</tr>
<tr>
<td>18 Comm Ports</td>
<td>10%</td>
</tr>
<tr>
<td>SDRAM controller</td>
<td>5%</td>
</tr>
<tr>
<td>Clock generator</td>
<td>2%</td>
</tr>
<tr>
<td>Pad Ring</td>
<td>17%</td>
</tr>
</tbody>
</table>
SDRAMs
- high bandwidth: 1 data word per cycle (to 100MHz)
- internal interleaving
- burst access w/ line-wrap; page-mode

Integrated SDRAM Controller
- page mode, internal interleaving, line-wrap
- 1 or 2 memory banks; duplicate addr & ctl
- 50MHz to 100MHz: 400 MB/sec to 800 MB/sec

NO Secondary Cache - Cost, Power vs Performance
Caches

Instruction and Data Caches
- 8K bytes each
- 2-way set associative
- 32-byte line size (4 64-bit words)
- 1-Cycle Access

Instruction Cache
- 3-Line Instruction Queue feeds Decoder
- Prefetch

Data Cache
- Non-blocking, pipelined design; Miss buffer (4-deep)
- Write-back, write-thru, or non-cacheable; Write buffer
- DMA snoop port

Operand Queues

Source Queue (depth=16)
- decouples instr decoder from mem and exe pipelines
- data fetched from dest_queue, dcache, wr-q, or mem
- exe results forwarded to matching src-q entries
- mem-read data forwarded to matching src-q entries

Destination Queue (depth=16)
- buffers execution results
- writes to data cache or wr-q or both
  - write-back, write-thru, or non-cacheable
Optimizing Data Flow

Vector_Move (VMOVD/VMOVW):
- moves 32 bytes to/from mem/regs/imm
  - eg, move cache line to/from registers in one cycle
- eliminates cache-line write allocate
- cache override: force either or both operands non-cach.

Prefetch (PREF)
- prefetches memory line into data cache
- bypasses exe pipeline; out-of-order
- overlaps/hides memory latency

Flush-line (FLLN)
- flushes memory line from data cache

Flush-dcache (FLDC)
- flushes indicated cache line

Vector Load/Store

memcpy():
** (r1) = source (in DRAM) **
** (r3) = destination (in DRAM) **
** (sp) = cache temporary **

avec:

vmovd (r1)+@dram, 0(sp)
vmovd (r1)+@dram, 32(sp)

vmovd 0(sp), (r3)+@dram
vmovd 32(sp), (r3)+@dram

bg avec

vector load/store cache management:
- minimizes DRAM ping-pong (page break penalties)
- avoids cache collisions and cache pollution
- avoids read-modify-write of result array (write allocate)
DAXPY Throughput

\[ [Y] = [Y] + C \times [X] \]

Scalar - 0.67 FLOPs per cycle

Vector - 2.0 FLOPs per cycle

\[ [Y] = [Y] + C \times [X] \]

\[ C \times X \times Y = 64\text{-bit Constant or Vector Element} \]
Optimizing Control Flow

Dynamic Branch Prediction
- 256-entry table, direct map
- 2-bit state

Conditional Move - CMOV <cond>,src,dst
- if <cond> dst = src
- supports speculative execution

Convert_cond_code - CCC <cond>,dst
- dst = <cond> ? 1 : 0
- set boolean result based on flag condition

Boolean_AND - BAND src1,src2,dst
- dst = src1 && src2
- boolean expression evaluation without branches

Speculative Execution

transform: into:
if (condition) block_1
    block_1 block_2
else CMOV
    block_2 CMOV

depending on:
average branch cost
execution costs of block_1 and block_2
branch profiling data (if available)
criteria: spec_exec cost < branch cost
combine two cheap blocks into one
combine an infrequently-executed, cheap block
    with an oft-executed, expensive block
nCUBE3 Communications

Hypercube Network: $2^{**n}$

18 full-duplex ports on each processor
n hypercube connections + m folded connections
messages cut-through without CPU intervention

16 DMA channels: 8 Send, 8 Receive
double-buffered DMA control
32-byte data buffer per channel
scatter-gather support

End-to-End Reporting

Status packet delivered from target node, back to src
- status = h/w error status or s/w status
- retraces back along xmission path, using same links
- xmission path held until ETE packet unravels it

Benefits of hardware ETE
- Quick and Reliable
- no additional xmissions (no CPU and s/w overhead)
Routing Methods

Oblivious Wormhole routing
- only 1 fixed path - dimension-order, 'e-route'
- address pkt advances the path until blocked
- data flows directly behind address pkt, wormhole fashion

'Maze' adaptive routing
- exhaustive search of min-length paths until free path found
- addr pkt traverses any min-path unallocated link, backtracking and deallocating when blocked
- uses ETE reporting network to back-track/report
- data xmission begins after path established
nCUBE3 Processor Status and Performance

(to be provided at time of presentation)