The Reconfigurable Streaming Vector Processor (RSVP™)

Silviu Ciricescu, Ray Essick, Brian Lucas, Phil May, Kent Moat, Jim Norris, Michael Schuette, and Ali Saidi†
Motorola Labs, Motorola, Schaumburg, IL
The Mitre Corporation, Bedord, MA†
Introduction

- **Coprocessor**
  - A special-purpose processing unit that assists the CPU in performing certain types of operations

- **Vector Architecture**
  - Traditionally used in supercomputers
  - Heavily pipelined architecture that operates on vectors and matrices
  - Vector processors are machines built primarily to handle large scientific and engineering calculations
Introduction (Cont.)

- Multimedia Functions
  - Computationally intensive
  - Data streaming
  - Applications: Image/Video capturing, Handwriting recognition, Voice recognition
  - Many more portable/embedded applications

- Streaming Data
  - Data is produced/acquired as a stream of elements
  - Relevant for a short period of time
  - Undergoes same set of computation
  - High degree of spatial locality
  - Relatively poor temporal locality
  - Data access patterns allow prefetching of data ahead of computation
Introduction to RSVP

- RSVP is a vector coprocessor architecture
- Accelerates streaming data operations
- Targets multimedia functions
- RSVP Programming model
  - Streaming data description – Vector shape and size
  - Computation description – Data Flow Graphs
  - Above descriptions are intuitive and independent of each other
  - Machine Independent
Motivation for RSVP

- Vector Processing Architectures
  - In recent years, targeting multimedia rather than Supercomputer
  - MMX extensions to Intel IA32 architecture
  - AltiVec extensions to PowerPC architecture
  - “Wide Word SIMD”
    - RISC like Load-store Programming model
    - Wide fixed-sized vector registers
    - Level of abstraction low
  - RSVP better Architecture than “Wide Word SIMD”
- Performance Gap between memory and processing
Streaming Vector Data Processing using RSVP Architecture

- A coprocessor to operate synchronously with an existing host CPU
- A programming model that separates the description of data from computation
  - Data described by the location and shape in memory
  - Computation described by Data Flow Graph
Streaming Computation Model

- Decoupled Operand Fetch
- Deep Pipelining (function unit chaining)
- SIMD Processing

Figure 1, Decoupled operand prefetch allows data to be fetched ahead of the computation.
Streaming Computation Model

- RSVP architecture utilizes a stream-oriented approach to vector Processing
- Decouples and overlaps data access and data processing
- Decoupled Operand Fetch
  - Vector stream units – independent load/store unit
  - VSUs communicate with processing unit through interlocked FIFO queues
- Deep Pipelining
  - Processing unit split into N-stage pipeline by chaining multiple function units together
  - Allows higher clock frequencies and increased resource utilization
- SIMD processing
  - Multiple taps on each of the VSUs
  - Parallelism limited by resource limitation, algorithm characteristics, vector size
Describing Data

- Vector processing handled by Vector Streaming Unit (VSU)
- Vector description consists of pointer to first element in each vector in memory and description of vector shape
- Shape of vector consists of three scalar values: Stride, Skip, Span
Vector Shape

Contiguous vector (stride = 1)

Striding vector (stride = 2)

Repeating vector (stride = 1, span = 5, skip = -5)

Subarray (2x4 in 8x8, row order) (stride = 1, span = 2, skip = 6)

Subarray (column order) (stride = 3, span = 5, skip = -14)

Figure 3, The skip, span, and stride shape descriptors allow a range of vector shapes to be specified.
Describing Computation Using Data Flow Graphs

- Synchronous DFG language expresses vector operations in a machine independent manner
- Dependencies explicitly stated to facilitate parallel execution
- DFG Node description
  - Input Operands – Reference to previous nodes rather than named registers (Data Dependence)
  - Operation to be performed
  - Minimum precision of its output values
- Iteration-to-Iteration dependence
  - Tunnel nodes – source and sink of data flow same
- Order dependence
  - Sequential execution (linear DFG) should be matched by any parallel execution
Scheduling and Binary Compatibility

- Binary form of linear DFG executed by RSVP
- Linear DFG may not be best suited for direct execution on a particular RSVP implementation
- Universal Fat Binaries (UFB) provided
  - More than one binary form of DFG
  - DFG Compiler creates them
  - Linked list with linear DFG appearing last
  - The binary form that first executes will be used
Quant Programming Example

- Compresses video images through quantization

```c
void quant(short *out, short *in, int n, short qp)
{
    long rq, b, c;
    rq = ((1 << 16) + qp) / (qp << 1);
    b = qp - !(qp & 1);
    while (--n >= 0)
    {
        c = *in++;
        if (c < 0) c -= b;
        else if (c > 0) c -= b;
        *out++ = (c*rq) / (1 << 16);
    }
}
```

Figure 4, Original quant routine written entirely in C.

```c
void quant(short *out, short *in, int n, short qp)
{
    long rq, b, c;
    rq = ((1 << 16) + qp) / (qp << 1);
    b = qp - !(qp & 1);
    _vhalfl(in);
    _vohalf0(out);
    _vset(1, rq);
    _vset(2, b);
    _vloop(&rsvp_quant, n)
}
```

Figure 5, Rewritten routine setting up the RSVP processor to execute the inner loop using our provided API.
Quant Programming Example - Data Flow Graph

```c
rsvp_quant:
Q1: vld.s16 (v1)   // c = *in++;
Q2: vsign.s16 Q1   
Q3: vscalar s2     // s2 is b
Q4: vscalar s1     // s1 is rq
Q5: vimm 16        
Q6: vmul.s16 Q2,Q3 // if (c<0) c+=b;
Q7: vsub.s16 Q1,Q6 // else if (c>0) c-=b;
Q8: vmul.s32 Q7,Q4  // c *= rq;
Q9: vsar0.s16 Q8,Q5 // *out++ = c/(1<<16);
Q10: vst.s16 Q9,(v0)
```

Figure 6. The loop code is described in an intuitive data-flow graph form (linear (text) form and graphical form shown).
Architecture

Figure 8, The RSVP architecture (white solid boxes), and supporting structures hidden from the programmer (gray boxes).
Architecture

- Input and output VSUs
  - Minimum 1 and 3 respectively
  - Maximum 64
  - VSUs handle all issues related to loading/storing data

- DFG
  - No more than 256 nodes
  - “reach back” no more than 63 nodes

- Accumulator and scalar/tunnel registers
  - Minimum 2 and 16 respectively
  - Maximum 64

- Scheduler
  - Converts DFG to machine dependent form

- Control
  - Implements machine dependent form of DFG

- Function units
  - Should support all operations allowed by DFG
Implementation

- First Implementation of RSVP
  - Low cost, low power solution
  - Fabricated in TSMC 0.18um CMOS Technology

- Tool set
  - Compiler, assembler, linker for ARM
  - Compiler for RSVP linear DFG

- Area
  - Comparable to ARM9 host processor
  - In effect, doubling area resulted in greater than 2x increase in performance

- Power Dissipation
  - Comparable to ARM9 host processor
  - Clock gating done to reduce system power dissipation
First Implementation

Figure 10, RSVP implementation block diagram.
Reconfigurability

- Fabric and Queue
  - Reconfigurable interconnect
  - 20 links, each link can transfer 16-bits of data from its source to its destination
  - Links can be reconfigured every cycle

- Function units
  - 64 bits wide, sliced on 16-bit boundaries
  - Fully pipelined with result latching
Results - Kernel Speedups

Magnitude of speedup results from large effective Instruction width of RSVP Implementation

Issue Rate: ARM9 - 0.78 IPC, RSVP - 9 IPC

WWSI IMD - 4 Instructions per cycle

Figure 11, RSVP processor kernel speedups (operating out of tile buffer).
Results - Application Speedups

![Bar chart showing speedup over ARM9](chart.png)

**Figure 12,** RSVP application speedups (ideal memory and realistic memory).
Conclusion

- RSVP architecture
  - vector coprocessor/accelerator architecture
  - improves general purpose CPU performance on streaming data applications
  - Improves time to market because of ease of programmability
  - Speedups for kernels and applications range from 2 to over 20 times that of a host processor alone
Thank you

Questions ???