

# *Dynamic Parallel Media Processing using Speculative Broadcast Loop (SBL)*

---

**Jason Fritts**

*Assistant Professor*

*Department of Computer Science*

*Co-Author: Prof. Wayne Wolf*



## *Overview*

---

- **Current Media Processing Solutions**
  - Increasing demand for multimedia
  - advanced media apps demand greater throughput and flexibility
  - likely solution: *media processors*
- **Parallelism in Media Processing**
- **Speculative Broadcast Loop (SBL) run-time method**
- **Evaluation Environment**
- **Results**
- **Conclusions and Future Research**

## Digital Multimedia

- **Ever-growing range of applications**

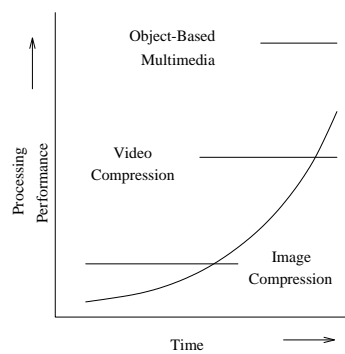
- *Communication*
  - video conferencing & telephony
  - World Wide Web
  - digital libraries
  - advertising & marketing
- *Entertainment*
  - games
  - home theater
- *Computer Vision*
  - image understanding
  - surveillance & tracking
- *Education*
  - interactive learning
  - virtual classrooms
- *Ubiquitous Computing*
  - telecommuting
  - home networks & internet appliances
  - military
- *Art and Architecture*

**Multimedia is primarily a communication media**

3

## Future of Multimedia

**Multimedia industry evolves with processor performance.**



**Multimedia is moving towards advanced representations**

4

## *Current Media Processing Methods*

---

- **Application-specific processors**
  - good performance
  - low cost & power
  - BUT, limited flexibility
- **Multimedia extensions to general-purpose processors**
  - utilizes both ILP and subword parallelism
  - speedups of 2-8x possible using subword parallelism
  - minimal additional cost
  - BUT, processor optimized for GPP apps, not media apps
- **Current “programmable” media processors (PMPs)**
  - good performance
    - special functional units: DCT, VBR coder, motion estimation
  - moderate frequency
  - reasonable flexibility
    - special programming libraries or paradigms
  - current processors target a subset of media types
    - graphics & video / video & audio / audio & speech / etc. ...
    - ideally, desirable that processor be as general as possible

5

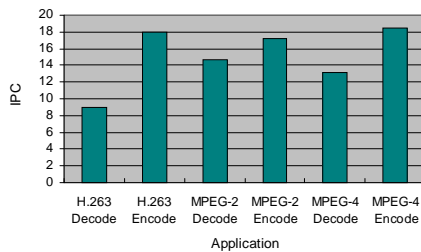
---

## *Parallelism in Media Processing*

6

## Potential Parallelism

- **Studies indicate significant parallelism**
  - Trace-driven studies (Oracle experiments)
- **Where is parallelism available?**
  - Data Parallelism

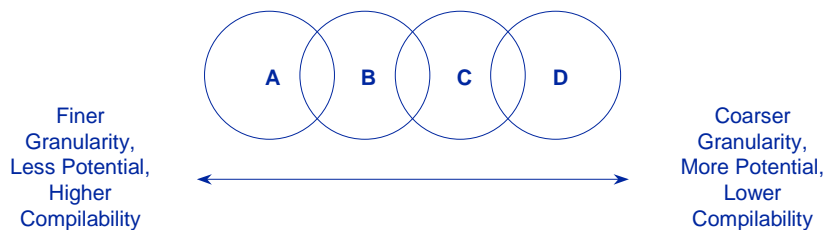


- video applications indicate significant available parallelism

7

## Parallelism in Multimedia

- **A) Instruction Level Parallelism (ILP)**
  - Typical speedup of 2x
- **B) Subword Parallelism**
  - 2-4x speedup with hand scheduling
  - Effective compilation unavailable (under development)
- **C) Multi-processor (Loop-Level) Parallelism**
  - Requires parallel compiler techniques
  - Currently most underutilized means of parallelism
- **D) Task-Level Parallelism**
  - Typically requires explicit parallel programming by user
  - Often used in hardware



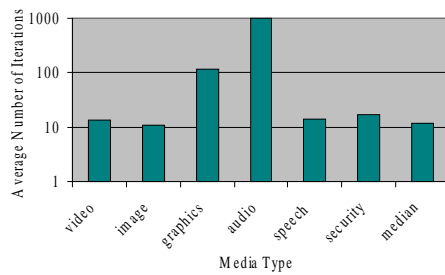
8

## Existing Programmable Media Processors

Media Processor	Specialized Hardware	ILP	Subword Parallelism	Parallel Processing	Clock Frequency
Texas Instruments MVP (C8x)	-	-	✓	✓	50 MHz
Texas Instruments VelociTI (C6x)	-	8-wide	✓	-	300 MHz
Philips TriMedia TM-1000	✓	5-wide	✓	-	166 MHz
Philips TriMedia TM-2000	✓	5-wide	✓	-	~300 MHz
Equator/Hitachi MAP1000A	✓	4-wide	✓	-	220 MHz
NEC V830R/AV	-	2-wide	-	-	200 MHz
Chromatic Research Mpact-1	✓	5-wide	✓	-	?
Chromatic Research Mpact-2	✓	6-wide	✓	-	125 MHz
MicroUnity MediaProcessor	✓	-	✓	-	-
Samsung MSP-1	✓	-	✓	✓	-

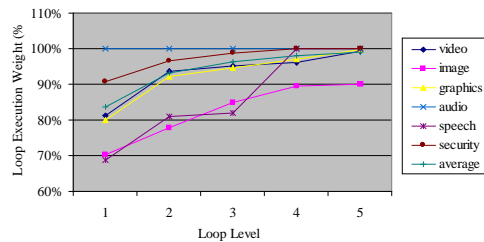
9

## Loop Characteristics



- large number of iterations per loop

- highly loop centric



10

## *Speculative Broadcast Loop (SBL) Method*

11

## *Hand-Scheduling Experiment - Traditional DCT*

```

for (m = 0; m < 8; m++)
  for (n = 0; n < 8; n++) {

    y[m][n] = 0;

    for (i = 0; i < 8; i++)
      for (j = 0; j < 8; j++)
        y[m][n] += y[m][n] + x[i][j] * c[i][m] * c[j][n];
  }

```

```

r0 = m      r10 = &x[0][0]
r1 = n      r11 = &x[i][0]
r2 = i      r12 = &c[0][n]
r3 = j      r13 = &c[0][m]
r4 = m * 4  r14 = &c[j][m]
r5 = n * 4  r15 = x[i][j]
r6 = i * 4  r16 = c[i][m]
r7 = j * 4  r17 = c[j][n]
r8 = i * 4 * 8  r20 = y[m][n]
r9 = j * 4 * 8  r21 = &y[m][n]

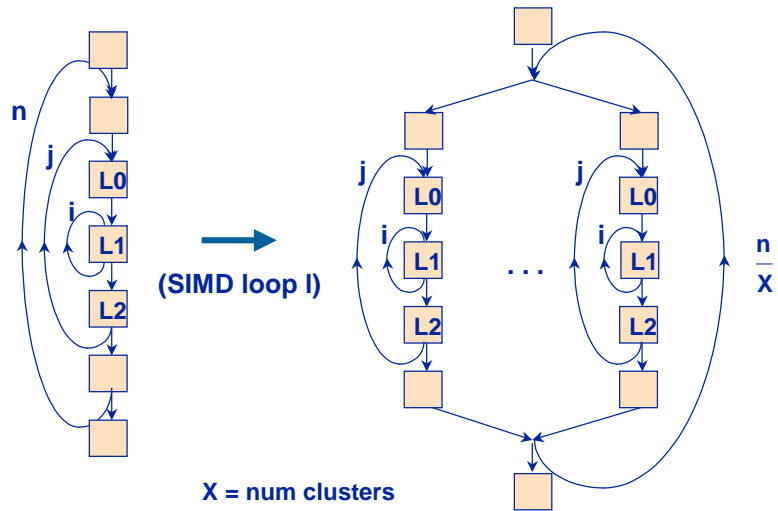
```

### Two inner loops:

L0:	mult r6, r2, 4	(1)
	mult r8, r6, 8	(2)
	add r11, r10, r8	(3)
	load r14, r13, r8	(4)
	add r20, 0, 0	(5)
L1:	mult r7, r3, 4	(6)
	mult r9, r7, 8	(7)
	load r15, r11, r7	(8)
	load r17, r12, r9	(9)
	mult r18, r16, r15	(10)
	mult r19, r18, r17	(11)
	add r20, r20, r19	(12)
	add r3, r3, 1	(13)
	bge r3, 8, L1	(14)
L2:	store r21, r20	(15)
	add r2, r2, 1	(16)
	bge r2, 8, L0	(17)

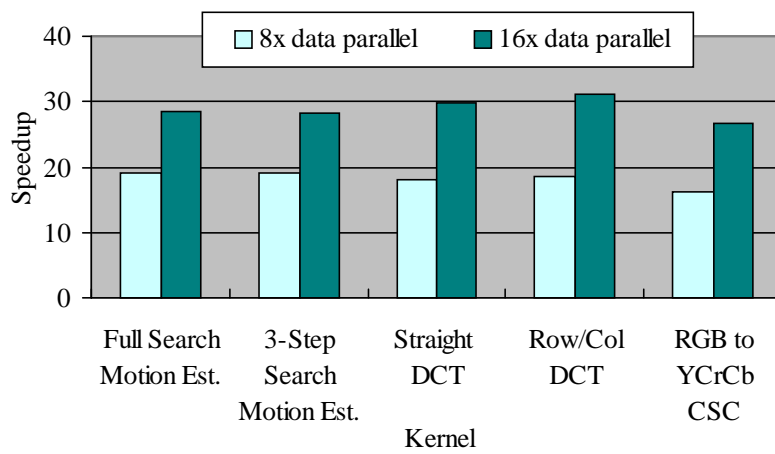
12

## *SIMD Parallelism Across Clusters*



13

## *Speedup from SIMD Parallelism Across Clusters*



14

## *Parallel Compiler and Architecture Methods*

---

- **Determine independence of loop iterations**
  - Static parallelization by a parallel compiler
    - memory disambiguation may not be able to determine parallelism
    - must conservatively assume loop iterations are not parallel
  - Run-time parallelization enables optimistic parallelization
    - non-speculative -> *inspector/executor* approach
    - speculative -> dynamic memory conflict checking
- **Speculative run-time methods**
  - LRPD Test (Rauchwerger)
  - Multiscalar Project (Franklin, Sohi)
  - Thread-Level Data Speculation (TLDS) (Mowry)
  - Thread-Level Speculation (TLS) (Olukotun, Lam)

[LRauchwerger95] "The LRPD Test: Speculative Run-Time Parallelization of Loops with Privatization and Reduction Parallelism," PLDI, 1995.  
[MFranklin93] "The Multiscalar Architecture," Ph.D. Thesis, University of Wisconsin at Madison, 1993.  
[JGregory98] "The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization," HPCA-4, 1998.  
[JOplinger97] "Software and Hardware for Exploiting Speculative Parallelism with a Multiprocessor," Technical Report CSL-TR-97-715, Stanford University, 1997.

15

## *Speculative Broadcast Loop (SBL) Execution*

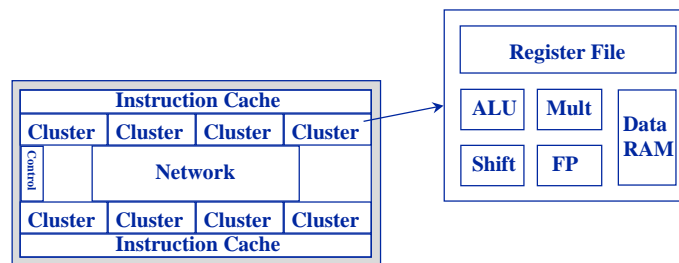
---

- **Speculative SIMD parallelism method for data parallelism**
  - Simplified version of Multiscalar, TLDS, TLS approaches
  - Supports fully and partially parallel loops
  - Does not support parallelism across function boundaries
- **SBL method ideal for multi-cluster architecture**
  - Single instruction control stream may be broadcast for SIMD execution
  - Cluster independence corresponds well with loop independence
  - Processing regularity of multimedia data parallelism seems ideal for SIMD parallelism
  - Control synchronization is implicit in single control stream
    - only need synchronization to ensure memory correctness
  - Separate clusters provides automatic scalar privatization

16

## Wide Issue => Clustered Architecture

- **Global register file too slow for wide issue**
- **Separate datapath into disjoint clusters:**
  - functional units (2-4 issue slots)
  - local register register file
  - local memories/caches
- **Low latency network between clusters**



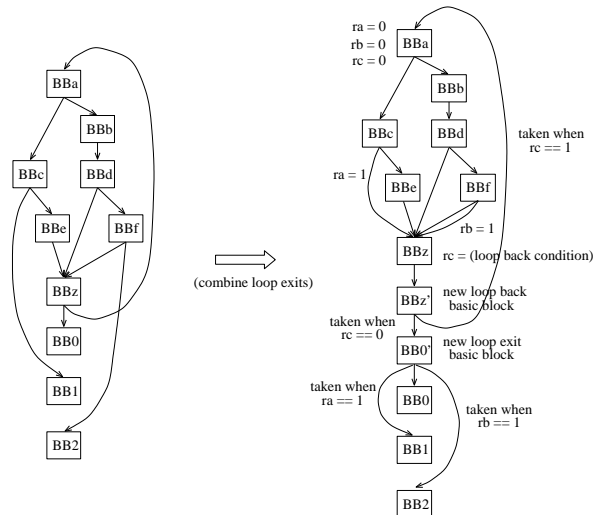
17

## Implementing SBL Execution

- **Major aspects of SBL execution**
  - Loop broadcast and multi-level loop scheduling for SIMD parallelism across a multi-cluster architecture
  - Hardware support for large-scale speculative parallel execution
- **Steps to implement SBL execution**
  - Find potentially parallel loops using profiling and register dependence analysis
  - Select loops for broadcast and schedule them to eliminate all unnecessary branches (Multi-Level If-Conversion)
  - Provide speculative hardware for support large speculative state
- **Multi-Level If-Conversion**
  - Inner loops: combine all control paths into single control path
  - Nested loops: recursively if-convert nested loop regions

18

## If-Conversion of Inner Loops



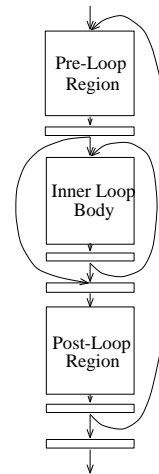
19

## If-Conversion of Nested Loop Regions

- Divide nested loop into inner loops, preloop regions, and post loop regions
- If-Conversion is similar for inner loops, preloop regions, and post loop regions
- Recursively apply nested loop if-conversion until no remaining nested loops

```

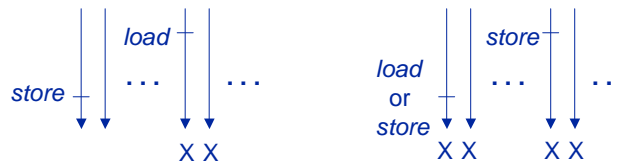
combine_loop_exits (main_loop) {
  if (main_loop has child loops) {
    for each (child_loop) {
      combine_loop_exits (child_loop);
      combine_pre_loop_exits (child_loop);
    }
    combine_post_loop_exits (child_loop);
  }
  else
    combine_inner_loop_exits (child_loop);
}
    
```



20

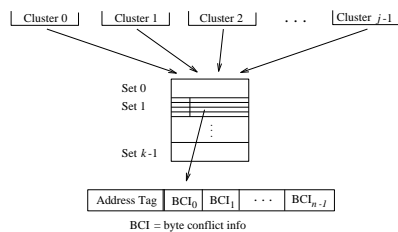
## Speculative Hardware Support for SBL Execution

- **SBL method must support speculation for**
  - Memory Independence Speculation
  - Control Flow Speculation
  - Loop Continuance Speculation
- **For large speculative state, use L1 data cache**
  - Also need checkpoints and a restorable register file
- **Loop Memory Conflict (LMC) cache monitors all memory accesses**
- **Memory conflict checking:**
  - Compares memory accesses for separate iterations
  - Clusters have left-to-right ordering
  - Store-load conflict
    - cancel iterations beyond load
  - Load-store/store-store conflict
    - cancel iterations beyond first load or store



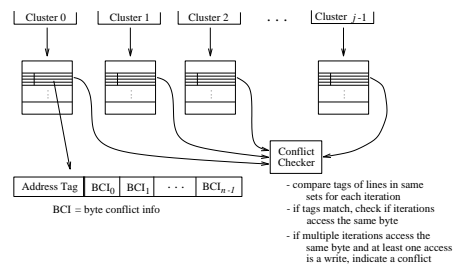
21

## Loop Memory Conflict (LMC) Cache



- global LMC cache

- distributed LMC cache



22

---

## *Evaluation Environment*

23

---

## *MediaBench Benchmark Suite*

- **Developed at UCLA**
  - MediaBench II currently under development

[CLee97] "MediaBench: A Tool for Evaluating and Synthesizing Multimedia Communication Systems," MICRO-30, 1997.
- **Excellent combination of applications**
  - video: MPEG-2
  - audio: ADPCM coder
  - graphics: Mesa
  - image: JPEG, EPIC, Ghostscript
  - security: PGP, Pegwit
  - speech: GSM, G.721, Rasta
- **Augmented for greater representation of future multimedia**
  - MPEG-4 object-oriented video
  - H.263 very-low bitrate video

24

## *IMPACT Compilation and Simulation Environment*

---

- **Aggressive ILP research compiler**
  - speculation (via the *superblock* optimization)
  - predication (via the *hyperblock* optimization)
  - loop unrolling
  - BUT, does not support parallel compiler optimizations
- **Architecture-independent evaluation**
  - large, generic instruction set
  - retargetable back-end
- **Performance analysis tools**
  - profiling
  - simulation for superscalar & VLIW architectures
  - extended simulation tools for simulation of:
    - multi-cluster architectures
    - SIMD multi-processing
- **Initial results assume perfect caches and classical compilation**
  - further results also indicate very good memory performance

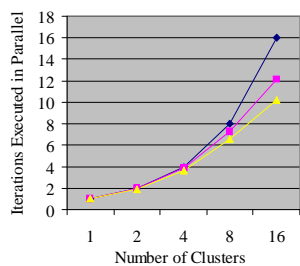
25

---

## *Results*

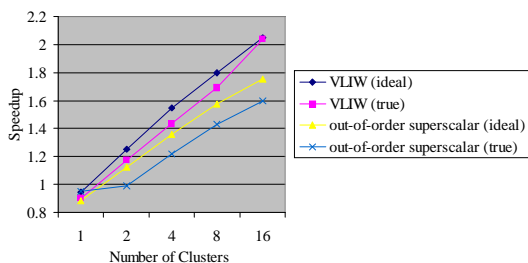
26

## Performance of SBL Execution



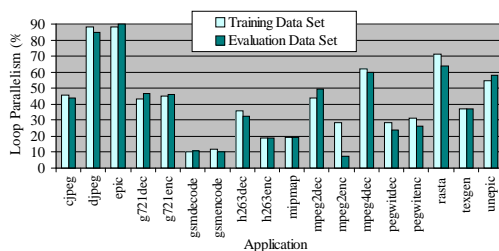
- speedup of SBL execution on parallel loops in MediaBench

- speedup of SBL method for full applications



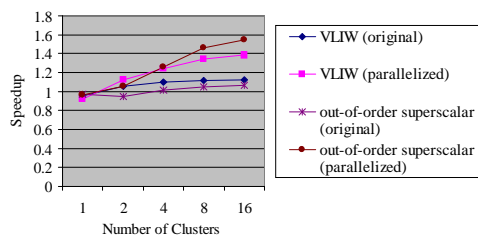
27

## Improving Performance of SBL Execution



- parallelism found by SBL execution

- speedup from manually applying parallel compiler optimizations



28

## *Conclusions and Future Work*

---

- **Conclusions**

- Multi-level if-conversion effective at minimizing control deviations in SIMD parallelism
- Achieved performance within 80-90% of ideal parallelism on parallel loops (for up to 8x processors)
- SBL misspeculation typically less than 5%
- Achieved ~2x speedup overall
- Parallel compiler optimizations promise further parallelism

- **Future Research**

- Use Parallel Compiler to Extract Data Parallelism
  - available parallelism for static SIMD parallelism
  - available parallelism for SBL execution method
  - available parallelism for static multiprocessor parallelism
- Evaluate SBL Method on Single-Chip Multiprocessor
  - available parallelism for SBL execution on multiprocessor
- Examine hybrid of SBL and LRPD test
  - perform conflict checking only on memory accesses with indeterminate data dependences
- Extend Multi-Level If-Conversion to Subword Parallelism