Issues in the Design of SWAR Programming Models

Randall J. Fisher
School of Electrical and Computer Engineering
Purdue University

rfisher@ecn.purdue.edu

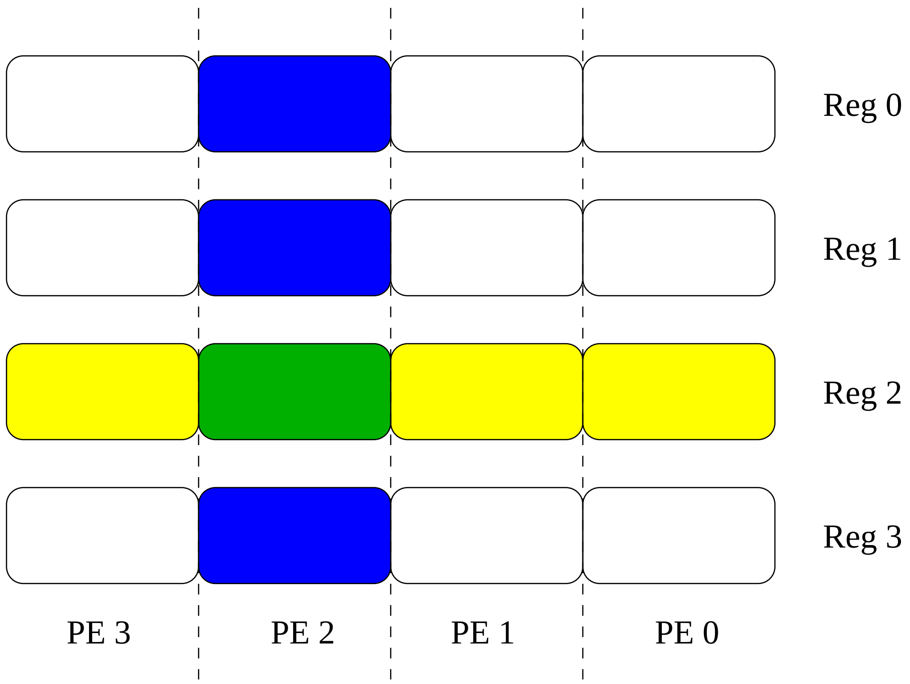
http://albrecht.ecn.purdue.edu/~rfisher/

SIMD Review

- Single Instruction stream, Multiple Data streams
- One Control Unit (CU)
- N Processing Elements (PEs)
- Single instruction read by CU, executed on all PEs concurrently
- Speedup equal to *parallelism width* N

SWAR is "SIMD Within A Register"

- CPU registers *partitioned* into *fields*.
- N register fields act as SIMD PEs
- Single instruction applied to all fields simultaneously
- Ideally, speedup equal to *parallelism width* N



SWAR Operation Types

- Polymorphic : Logical AND

	31	23	15	7 0
A:	7 A	F 3	C A	3 E
B:	3 3	6 7	A 6	E 7
A&B:	3 2	6 3	8 2	2 6

- Partitioned : Ideal Parallel Add

	31	23	15	7 0
A:	128	46	178	255
B:	49	43	135	7
A+B:	177	89	57	6

SWAR Operation Types cont'd

- Partitioned : Parallel Add with Spacers

31 23 15 7 0

A:	**46**	**?**	**178**	**?**	**255**
B:	**43**	**?**	**135**	**?**	**7**
A&~S	**46**	**0**	**178**	**0**	**255**
B&~S:	**43**	**0**	**135**	**0**	**7**
(A&~S)+(B&~S):	**89**	**1**	**57**	**1**	**6**
		Spacer		*Spacer*	

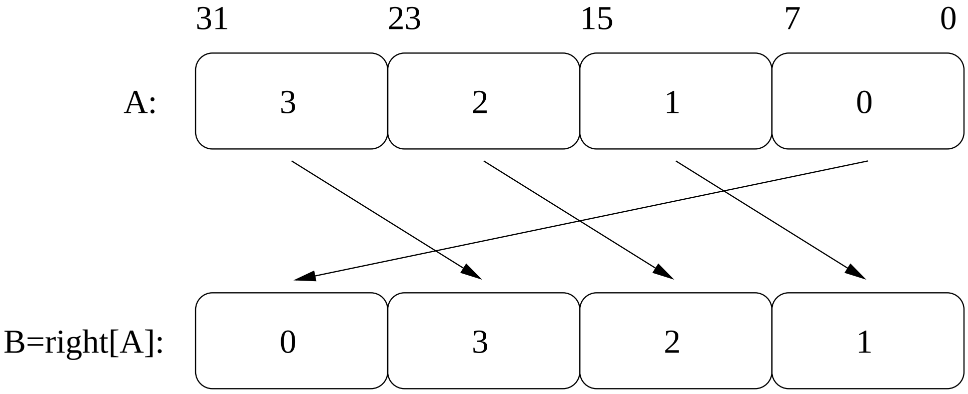
SWAR Operation Types cont'd

- Partitioned : Parallel Add with Virtual Spacers

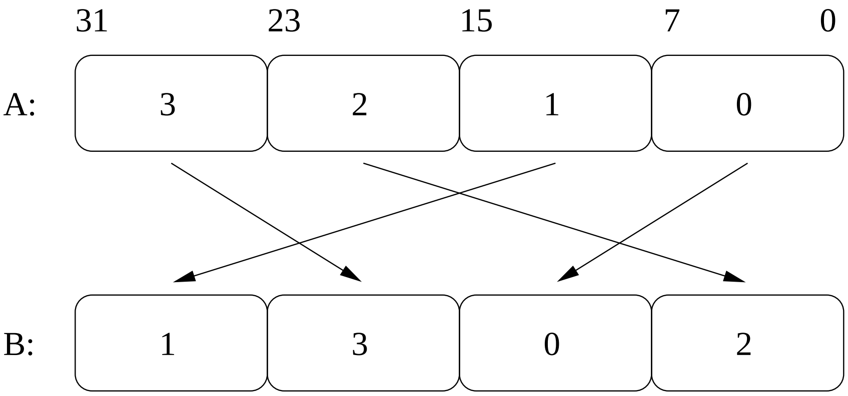
	31		23		15		7	0
A:		**128**		**46**		**178**		**255**
B:		**49**		**43**		**135**		**7**
C=A&~S:	**0**	**0**	**0**	**46**	**0**	**50**	**0**	**127**
D=B&~S:	**0**	**49**	**0**	**43**	**0**	**7**	**0**	**7**
E=C+D:	**0**	**49**	**0**	**89**	**0**	**57**	**1**	**6**
F=A^B:	**1**	**49**	**0**	**5**	**0**	**53**	**1**	**120**
G=F&S:	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
H=E^G:	**1**	**49**	**0**	**89**	**0**	**57**	**0**	**6**
H:		**177**		**89**		**57**		**6**

SWAR Operation Types cont'd

- Communication & Type Conversion :
  - right [ ] w/wrap



  - permutation



SWAR Operation Types cont'd

- Recurrence : reduceAdd ( )

<table>
<tr><th></th><th>31</th><th>23</th><th>15</th><th>7</th><th>0</th></tr>
<tr><td>A:</td><td>4</td><td>3</td><td>2</td><td colspan="2">1</td></tr>
<tr><td>B=A&mask:</td><td colspan="2">3</td><td colspan="3">1</td></tr>
<tr><td>C=A>>8:</td><td>0</td><td>4</td><td>3</td><td colspan="2">2</td></tr>
<tr><td>D=C&mask:</td><td colspan="2">4</td><td colspan="3">2</td></tr>
<tr><td>E=B+D:</td><td colspan="2">7</td><td colspan="3">3</td></tr>
<tr><td>F=E>>16:</td><td colspan="2">0</td><td colspan="3">7</td></tr>
<tr><td>G=E+F:</td><td colspan="2">7</td><td colspan="3">10</td></tr>
<tr><td>G&mask:</td><td colspan="5">10</td></tr>
</table>

Overview of Target Architectures

- Ideal SWAR-capable Registers
- Unenhanced SWAR-capable Registers
- Multimedia Enhanced Registers
- Multimedia Enhanced Registers with Logical Partitioning

Target: Ideal SWAR Registers

- Partitioning inherent to instructions
- All field sizes directly supported
- Field sizes limited only by required precision or magnitude
- Emulation and Isolation costs not an issue

Target : Unenhanced Registers

- Must be logically partitionable - polymorphics

- Non-polymorphics require extra work:
  - Emulation in software
  - Field isolation maintenance
  - May be more expensive than parallelism gained

- Field sizes limited only by required precision or magnitude

- Example: Normal integer registers

Target: Multimedia Extensions

- Hardware partitioning inherent to parallel instructions

- No emulation or isolation required for supported ops/sizes

- Inconsistent support:
  - Supported operations differ between families
  - Supported Field sizes differ between functions
  - May be more expensive to convert between supported operations than parallelism gained

- Examples: Intel MMX, Sun VIS, DEC MVI, AMD/Cyrix MMFP

Comparison of Multimedia Extension Sets

Target: Multimedia Extensions with Logical Partitioning

- Hardware partitioning inherent to supported instructions/field sizes

- Emulation and isolation required for unsupported instructions/field sizes

- May be too expensive to convert between supported operations

- May be too expensive to emulate unsupported instructions

Portability

- Which architectures to target?
  - More targets increases the odds of code survival
  - Fewer targets are easier to implement and optimize

- Which operations and field sizes to support?
  - Would like to exploit of each target's assets
  - Portability requires operations supported for one target be supported for all

Target Applications

- Application Specific vs General Purpose programming
  - Targeting specialized algorithms allows tuning
  - Targeting general-purpose algorithms avoids preclusion of new applications

- Which data types are required?
  - Integer support in standard integer registers and multimedia extensions
  - These could also support character types
  - Floating point support in newer multimedia extensions only

Integration/Abstraction Level

- Inline assembly:
  - Tedious, non-portable

- Functions operating on a parallel type:
  - Portable in languages such as C
  - Portable as linkable libraries

- Specially defined operators:
  - New languages
  - Languages which allow operator overloading

- Parallel class in object-oriented languages:
  - Portable in languages such as Java
  - May be difficult to port to non-OOLs

HLL Constructs

- Conditionals require *enable masking*
  - EM Generation from parallel conditional values
  - Application of EM to operations

- Loops require aggregate conditionals
  - *while* loops execute while condition holds for any field

- Other constructs
  - *router[]* may require disabled PEs to evaluate an expression for an enabled PE

Example: Enable Masking

- where (C) A = B;

PE 3	PE 2	PE 1	PE 0	
27	0	45	89	C
11...11	0	11...11	11...11	MASK
1	2	3	4	A
4567	9278	1238	4783	B
4567	2	1238	4783	A - After where()

Data Layout

- How to describe?
  - Explicit partitioning - a[4x16];
  - By field size - a[4];
  - As type or type attribute - int16 a[4];

- How visible should layout be?
  - Hidden layout allows larger field sizes to be used
  - Visible layout may allow element manipulation

- What about multi-register length vectors?

- What communications patterns should be supported?
  - generalized point-to-point is not easy
  - shifts allow some regular patterns such as right[] and left[]

Summary

- Microprocessor manufacturers have not developed portable high-level SWAR models

- A good model will have to address several issues:
  - Portability between targets
  - Future expansion
  - Generality vs. Exploitation
  - HLL Constructs applied to fields
  - Data Layout/Partitioning
  - Inter-field Communication

- More information at :
  http://shay.ecn.purdue.edu/˜swar