# Compiling for
# for
# SIMD Within A Register

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

rfisher@ecn.purdue.edu

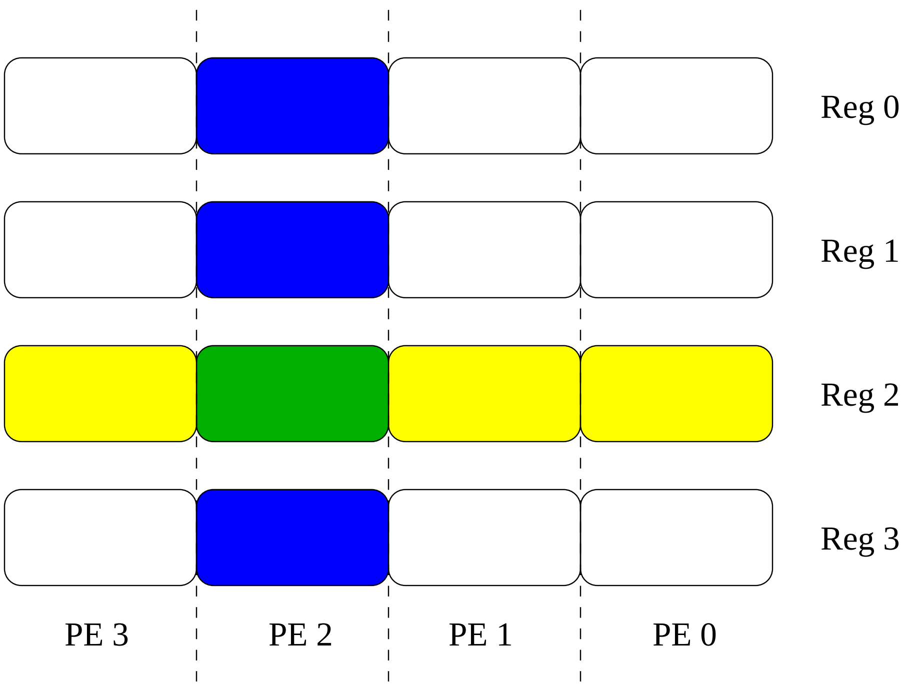
http://albrecht.ecn.purdue.edu/~rfisher/

hankd@ecn.purdue.edu

http://dynamo.ecn.purdue.edu/~hankd/

# 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 and independently
- Ideally, speedup equal to *parallelism width* N, which is a *function of field size*



# Possible SWAR Targets

Ideal target

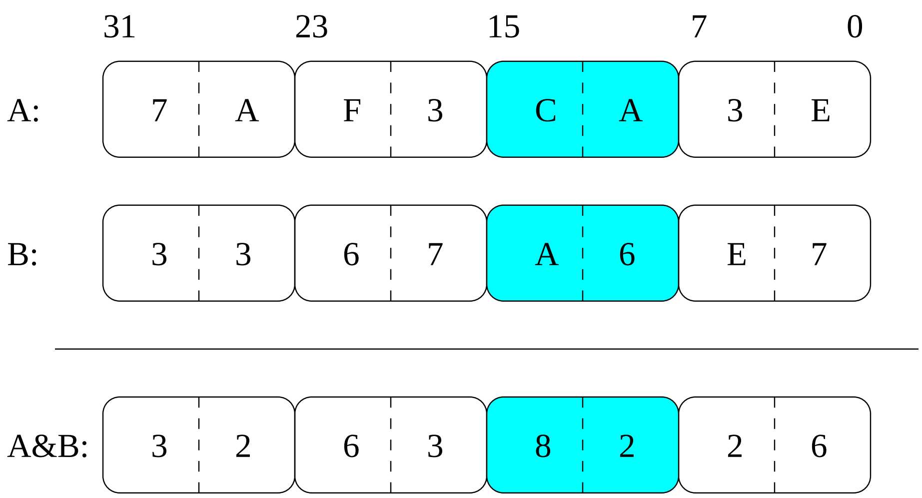
- arbitrary field partitioning for each op
- regular, full instruction set Current targets
- multimedia enhancements in current CPUs:
  AMD MMX + 3DNow!,
  Cyrix MMX + Extended MMX,
  DEC MVI,
  HP MAX,
  Intel MMX,
  Motorola AltiVec,
  SGI MDMX,
  Sun VIS
- Sparse, irregular coverage of ops and field sizes
- Techniques to simulate/avoid the missing ops/field sizes

# Comparison of Multimedia Extension Sets

| | DEC Alpha MVI | HP PA-RISC MAX | SGI MIPS MDMX | Intel, AMD, & Cyrix MMX | Sun Sparc V9 VIS |
|---|---|---|---|---|---|
| **# MM Registers** | 32 | 31 | 32 (1 Acc.) | 8 (+ Mem.) | 32 |
| **# Bits/ MM register** | 64 | 32 or 64 | 64 (192 Acc.) | 64 | 64 |
| **Which registers?** | Integer | Integer | Float | Float | Float |
| **Modular Add & Sub** | | X | X | X | X |
| **Saturation Add & Sub** | | X | X | X | |
| **Sum of Abs. Diffs** | X | | | | |
| **Modular Mul** | | | X | X | X |
| **Saturation Mul** | | | X | | |
| **Mul by Sign** (-,0,+) | | | X | | |
| **Mul/Add** | | | | X | |
| **Average** | | X | | | |
| **Shift Left/Right** | | X | X | X | |
| **Signed Shift Right** | | X | X | X | |
| **Shift and Add** | | X | | | |
| **Scale** | | | X | | |
| **Maximum/Minimum** | X | | X | | |

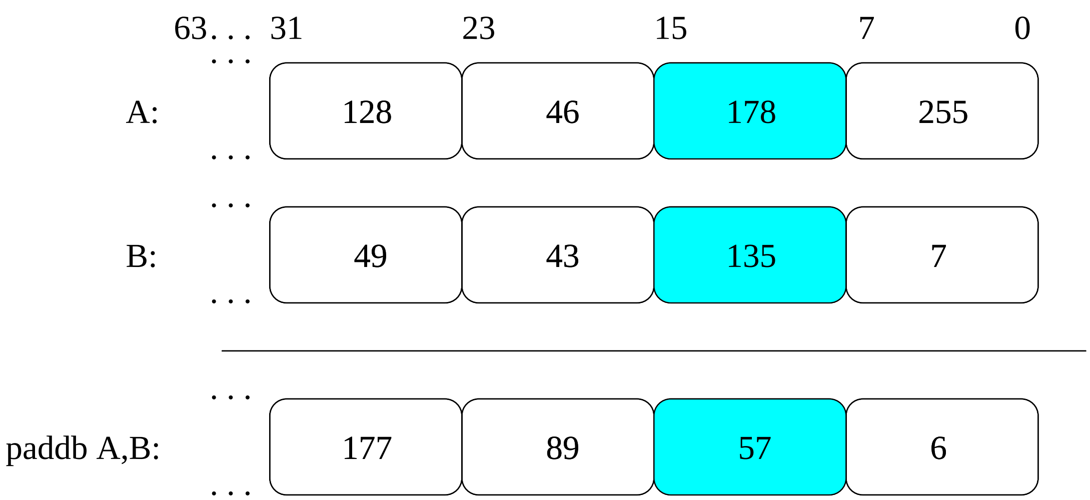
# Polymorphic Operations

- Operations independent of field sizes
- 1-bit (a.k.a. bitwise) operations, ANY test, etc.
- Example: Bitwise AND



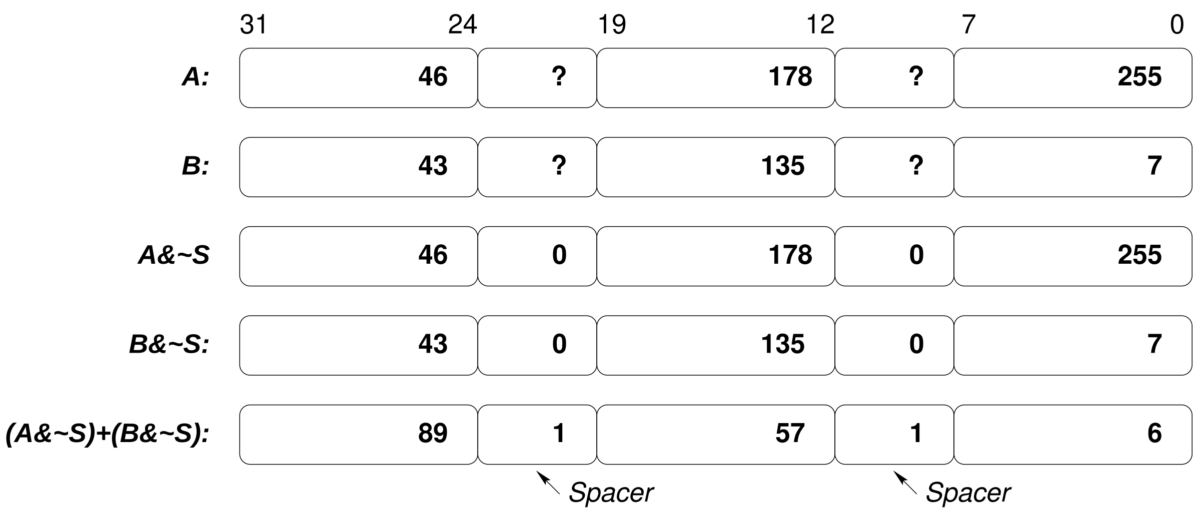
# Hardware-Partitioned Operations

- Instructions encode partitioning into fields
- Field operations applied concurrently and independently
- Carries/borrows blocked in hardware ->
  no interfield interference
- Example: paddb on MMX



# Partitioning Using Spacer Bits

- Used for unsupported field sizes
- Fields separated by non-data spacer bits
- Carries/borrows affect spacers -> no interfield interference
- Spacers normalized for each op
- Example: 8-bit field add



# Virtual Spacer Bits

- Polymorphic masking used to create virtual spacers between dense-packed fields
- Example: Dense 8-bit field add

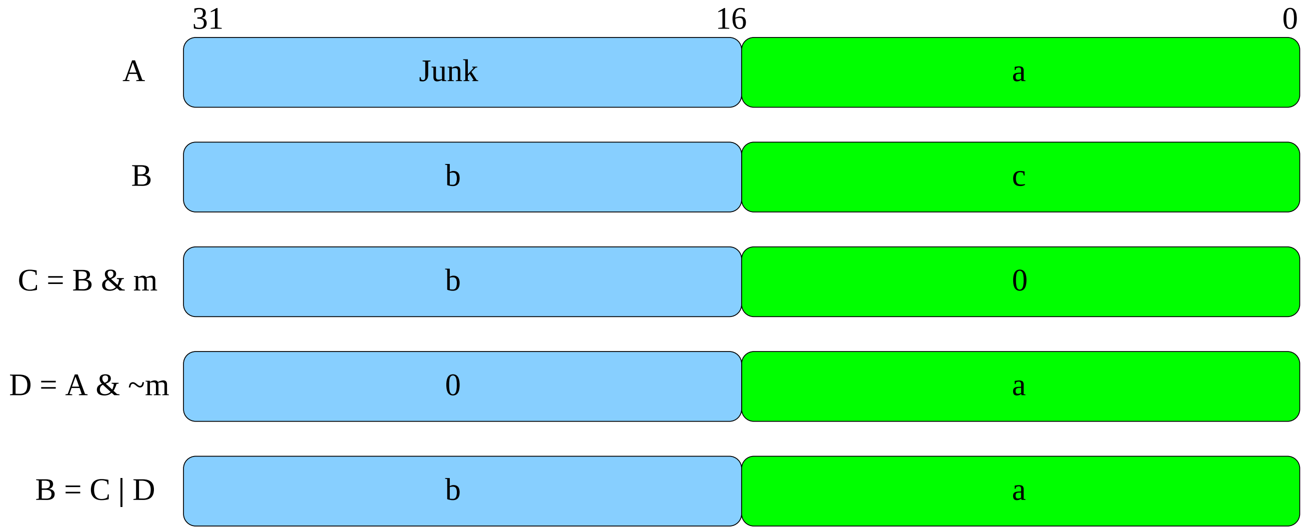
| | 31 | | 23 | | 15 | | 7 | 0 |
|---|---|---|---|---|---|---|---|---|
| ***A:*** | | **128** | | **46** | | **178** | | **255** |
| ***B:*** | | **49** | | **43** | | **135** | | **7** |
| ***C=A&~S:*** | **0** | **0** | **0** | **46** | **0** | **50** | **0** | **127** |
| ***D=B&~S:*** | **0** | **49** | **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** |

# Compiler Target

- Virtualized width and precision for portability
- Use only efficient field sizes
  - built-in sizes (e.g., 16 bits)
  - sizes where parallelism outweighs overhead in emulation (e.g., 2 bits)
- Support external interface to C/Fortran layout data, functions
- Cannot afford to be sloppy about local code quality (lest overhead exceed speedup)

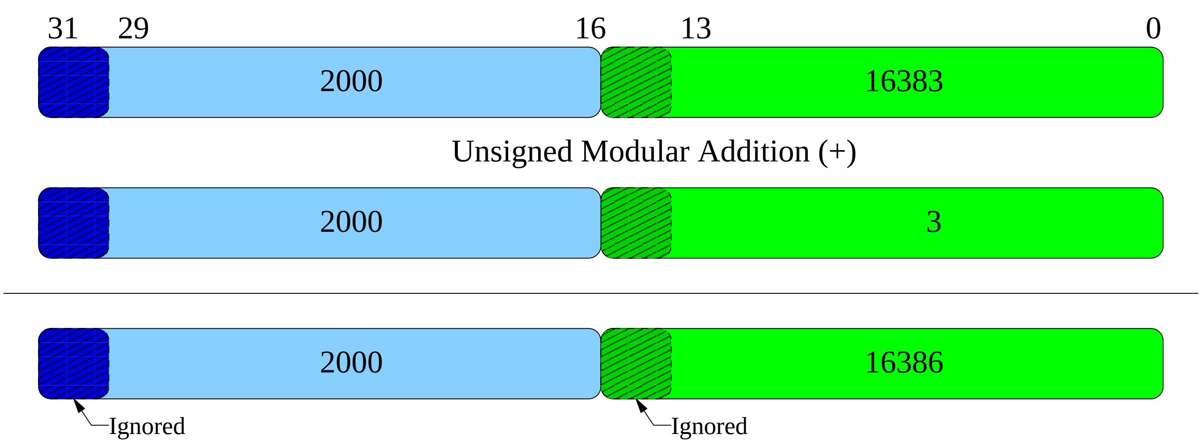
# Specified Width Isn't Supported

- Longer vectors are *fragmented* into word-length chunks
- Extra fields in a fragment are not treated *specially* unless the effect would be visible:
  - store results to external format
  - reduction op
- Example: store one 16-bit value a into location B



# Specified field size isn't efficient

- Promote to smallest efficient size > spec.
- Apply corrections where excess precision is objectionable, e.g., for saturation types and storing results
- Often, no loss of parallelism because fields do not span memory access (word) boudaries
- Example: 16-bit fields used for 14-bit fields



# Optimization of Spacer Manipulation

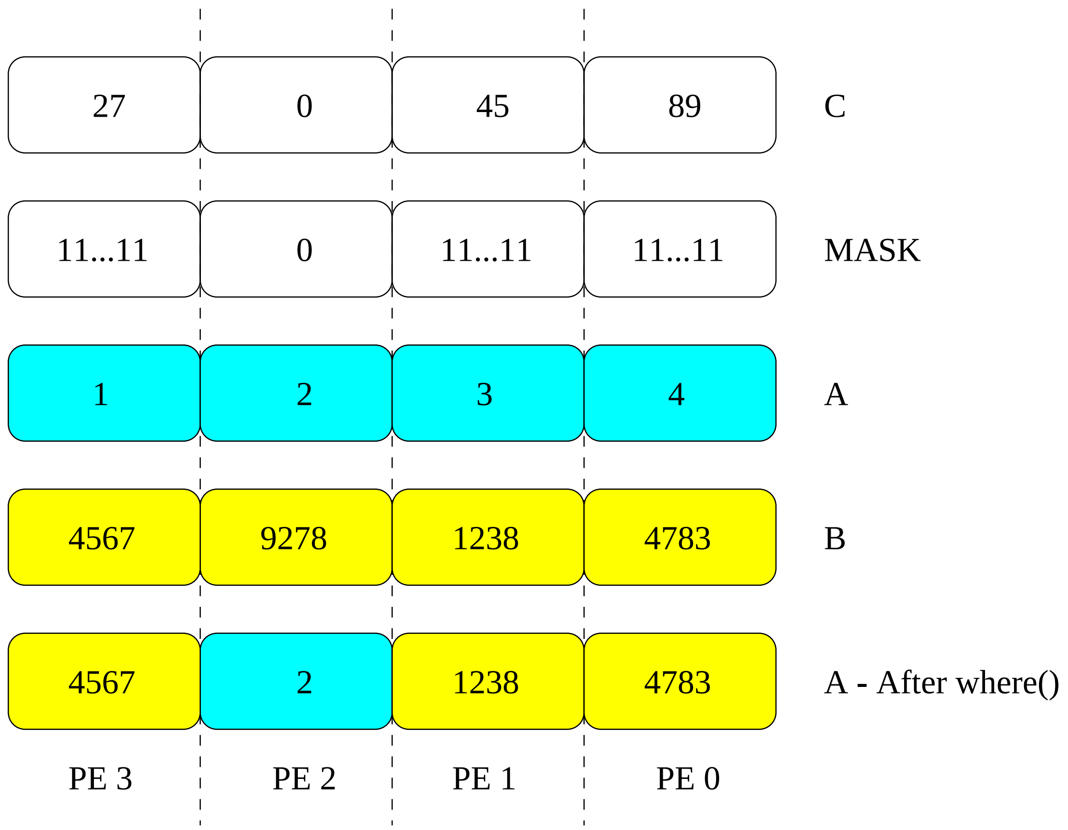
- Tracking of spacer values allows unnecessary normalization to be eliminated
- Example: 8-bit add using spacers

| | 31 24 | | 19 12 | | 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 |
| *((A&~S)+(B&~S))&~s:* | 89 | 0 | 57 | 0 | 6 |
| | | *Spacer* | | *Spacer* | |

| | 31 24 | | 19 12 | | 7 0 |
|---|---|---|---|---|---|
| *A:* | 46 | 0 | 178 | 0 | 255 |
| *B:* | 43 | 0 | 135 | 0 | 7 |
| *A+B:* | 89 | 1 | 57 | 1 | 6 |
| *(A+B)&~s:* | 89 | 0 | 57 | 0 | 6 |
| | | *Spacer* | | *Spacer* | |

# Enable Masking Optimizations

- Conditional execution requires masking of undesired field computations on emulated targets
- Tracking mask values allows unnecessary masking operations to be eliminated. If fields are:
  - all enabled -> no masking, all compute
  - all disabled -> no masking, original values used
  - otherwise -> the usual SIMD masking is required
- Example: where ( C ) A = B;



## SWARC Language and Compiler

- C like language, but:
  - 1st class arrays with optional :*precision*, and saturation e.g., int:7[100]x;
  - new ops for min, max, avg, shift, rotate, etc.
  - C*-like reductions, e.g., +=x
  - parallel masking, e.g. if (x>5) x=6;
  - mix with C code for I/O, etc.
- Scc compiler (very alpha, will be PD)
  - built using PCCTS, etc.
  - internals build vector form, optimize, fragment, optimize, and then schedule/reg. alloc. using detailed timing model
  - output is C, with inline asm for MMX, etc.
  - Scc works like gcc, and can invoke gcc to make executables

SWARC program to add two arrays of 14 char each:

```
void f(char[14] a, char:8[14] b) { a += b; }
```

Note that a has C language layout, requiring masking at store.

Scc Generated C Header:

```
/ * compiled by Scc alpha-test v980728 for generic IA32 MMX target */
extern void f(char *a,
mmx_t *b);
```

## Sccc Generated C Source Code:

```
/ * compiled by Scc alpha-test v980728 for generic IA32 MMX target */
void f(char *a,
mmx_t *b)
{
    extern mmx_t mmx_cpool[];
    register mmx_t *_cpool = &(mmx_cpool[0]);

    {
    movq_m2r(*(((mmx_t *) a) + 0), mm0);
    movq_m2r(*(((mmx_t *) a) + 1), mm1);
    paddb_m2r(*(((mmx_t *) b) + 0), mm0);
    movq_r2r(mm1, mm2);
    paddb_m2r(*(((mmx_t *) b) + 1), mm1);
    movq_r2m(mm0, *(((mmx_t *) a) + 0));
    pand_m2r(*(_cpool + 2), mm1);
    pand_m2r(*(_cpool + 3), mm2);
    por_r2r(mm2, mm1);
    movq_r2m(mm1, *(((mmx_t *) a) + 1));
    }

_return:
    emms();
}


/ * MMX constant pool */
mmx_t mmx_cpool[] = {
/ *    0 */  0x0000000000000000LL,
/ *    1 */  0xffffffffffffffffLL,
/ *    2 */  0x0000ffffffffffffLL,
/ *    3 */  0xffff000000000000LL
};
```

# gcc Generated Assembly Code:

```
	.file	"Scout.c"
	.version	"01.01"
gcc2_compiled.:
.text
	.align 4
.globl f
	.type	 f,@function
f:
	pushl %ebp
	movl %esp,%ebp
	movl 8(%ebp),%eax
	movl 12(%ebp),%edx
	movl $mmx_cpool,%ecx
#APP
	movq (%eax), %mm0
	movq 8(%eax), %mm1
	paddb (%edx), %mm0
	movq %mm1, %mm2
	paddb 8(%edx), %mm1
	movq %mm0, (%eax)
	pand 16(%ecx), %mm1
	pand 24(%ecx), %mm2
	por %mm2, %mm1
	movq %mm1, 8(%eax)
	emms
#NO_APP
	leave
	ret
.Lfe1:
	.size	 f,.Lfe1-f
.globl mmx_cpool
.data
	.align 4
	.type	 mmx_cpool,@object
mmx_cpool:
```

```
.long 0
.long 0
.long -1
.long -1
.long -1
.long 65535
.long 0
.long -65536
.size    mmx_cpool,32
.ident  "GCC: (GNU) 2.7.2.3"
```

Using various flavors of MMX, timing model predicts speedup of 2.44x over word-oriented C code for this example.

## Conclusions

- SIMD isn't dead!
- SWAR is a commercially important class of parallel targets
- SWAR can yield speedup, even from a HLL source
- SWARC as an example portable HLL for SWAR
- Scc as a research compiler, PD tool
- More SWAR compiler research needed
- More information at:
  http://shay.ecn.purdue.edu/~swar