Chapter 3

Subword parallelism

(This set contains adaptations of ten slides from MK Publishers´ originals and 25 new, complementary slides by Luiz Santos)

Streaming SIMD Extension 2 (SSE2)

- Adds 8 × 128-bit registers
  - Extended to 16 registers in AMD64/EM64T
- Can be used for multiple FP operands
  - 2 × 64-bit double precision (*double* in C)
  - 4 × 32-bit single precision (*float* in C)
  - Instructions operate on them simultaneously
    - <u>S</u>ingle-<u>I</u>nstruction <u>M</u>ultiple-<u>D</u>ata

Example: Matrix Multiplication

- C = C + A × B (DGEMM)
  - Double precision General Matrix Multiply
- Hypothesis:
  - All 32 × 32 matrices, 64-bit double-precision elements
- C code:

```
void mm (double c[][],
            double a[][], double b[][]) {
  int i, j, k;
  for (i = 0; i! = 32; i = i + 1)
    for (j = 0; j! = 32; j = j + 1)
      for (k = 0; k! = 32; k = k + 1)
        c[i][j] = c[i][j]
                     + a[i][k] * b[k][j];
}
```

Matrix: vectorial representation

- For higher performance:
  - Single-dimensional representation of a matrix
  - Column-major transformation

n

$$\begin{array}{ccc} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{array} \quad n$$

Address arithmetic

c_{ij} **= C[i+j*n]**
i, j in [0,n-1]

c_{ij} **= C[i+j*3]**
i, j in [0,2]

C[0]	c_{00}	n
C[1]	c_{10}	
C[2]	c_{20}	
C[3]	c_{01}	n
C[4]	c_{11}	
C[5]	c_{21}	
C[6]	c_{02}	n
C[7]	c_{12}	
C[8]	c_{22}	

Example: c_{12} **= C[1+2*3] = C[7]**

Alternative representation

- Unoptimized code:

```
void dgemm (int n, double* A, double* B, double* C)
{
 for (int i = 0; i < n; ++i)
   for (int j = 0; j < n; ++j)
   {
    double cij = C[i+j*n]; /* cij = C[i][j] */
    for(int k = 0; k < n; k++ )
     cij += A[i+k*n] * B[k+j*n]; /* cij += A[i][k]*B[k][j] */
    C[i+j*n] = cij; /* C[i][j] = cij */
   }
}
```

Alternative representation

- x86 assembly code: (generated with gcc, 2014 version)

```
1. vmovsd (%r10),%xmm0  # Load 1 element of C into %xmm0
2. mov %rsi,%rcx        # register %rcx = %rsi
3. xor %eax,%eax        # register %eax = 0
4. vmovsd (%rcx),%xmm1  # Load 1 element of B into %xmm1
5. add %r9,%rcx         # register %rcx = %rcx + %r9
6. vmulsd (%r8,%rax,8),%xmm1,%xmm1 # Multiply %xmm1,
   element of A
7. add $0x1,%rax        # register %rax = %rax + 1
8. cmp %eax,%edi        # compare %eax to %edi
9. vaddsd %xmm1,%xmm0,%xmm0 # Add %xmm1, %xmm0
10. jg 30 <dgemm+0x30>  # jump if %eax > %edi
11. add $0x1,%r11d      # register %r11 = %r11 + 1
12. vmovsd %xmm0,(%r10) # Store %xmm0 into C element
```

(Inner loop only, i.e. it corresponds to lines 6 to 9 from source code)

Alternative representation

- x86 assembly code: (generated with gcc, 2014 version)

```
1. vmovsd (%r10),%xmm0  # Load 1 element of C into %xmm0
2. mov %rsi,%rcx        # register %rcx = %rsi
3. xor %eax,%eax        # register %eax = 0
4. vmovsd (%rcx),%xmm1  # Load 1 element of B into %xmm1
5. add %r9,%rcx         # register %rcx = %rcx + %r9
6. vmulsd (%r8,%rax,8),%xmm1,%xmm1 # Multiply %xmm1,
   element of A
7. add $0x1,%rax        # register %rax = %rax + 1
8. cmp %eax,%edi        # compare %eax to %edi
9. vaddsd %xmm1,%xmm0,%xmm0 # Add %xmm1, %xmm0
10. jg 30 <dgemm+0x30>  # jump if %eax > %edi
11. add $0x1,%r11d      # register %r11 = %r11 + 1
12. vmovsd %xmm0,(%r10) # Store %xmm0 into C element
```

(Inner loop only, i.e. it corresponds to lines 6 to 9 from source code)

Alternative representation

- x86 assembly code: (generated with gcc, 2014 version)

```
1. vmovsd (%r10),%xmm0  # Load 1 element of C into %xmm0
2. mov %rsi,%rcx        # register %rcx = %rsi
3. xor %eax,%eax        # register %eax = 0
4. vmovsd (%rcx),%xmm1  # Load 1 element of B into %xmm1
5. add %r9,%rcx         # register %rcx = %rcx + %r9
6. vmulsd (%r8,%rax,8),%xmm1,%xmm1 # Multiply %xmm1,
   element of A
7. add $0x1,%rax        # register %rax = %rax + 1
8. cmp %eax,%edi        # compare %eax to %edi
9. vaddsd %xmm1,%xmm0,%xmm0 # Add %xmm1, %xmm0
10. jg 30 <dgemm+0x30>  # jump if %eax > %edi
11. add $0x1,%r11d      # register %r11 = %r11 + 1
12. vmovsd %xmm0,(%r10) # Store %xmm0 into C element
```

(Inner loop only, i.e. it corresponds to lines 6 to 9 from source code)

Alternative representation

- x86 assembly code: (generated with gcc, 2014 version)

```
1. vmovsd (%r10),%xmm0  # Load 1 element of C into %xmm0
2. mov %rsi,%rcx        # register %rcx = %rsi
3. xor %eax,%eax        # register %eax = 0
4. vmovsd (%rcx),%xmm1  # Load 1 element of B into %xmm1
5. add %r9,%rcx         # register %rcx = %rcx + %r9
6. vmulsd (%r8,%rax,8),%xmm1,%xmm1 # Multiply %xmm1,
   element of A
7. add $0x1,%rax        # register %rax = %rax + 1
8. cmp %eax,%edi        # compare %eax to %edi
9. vaddsd %xmm1,%xmm0,%xmm0 # Add %xmm1, %xmm0
10. jg 30 <dgemm+0x30>  # jump if %eax > %edi
11. add $0x1,%r11d      # register %r11 = %r11 + 1
12. vmovsd %xmm0,(%r10) # Store %xmm0 into C element
```

(Inner loop only, i.e. it corresponds to lines 6 to 9 from source code)

Alternative representation

- x86 assembly code: (generated with gcc, 2014 version)

```
1. vmovsd (%r10),%xmm0   # Load 1 element of C into %xmm0
2. mov %rsi,%rcx  Cij (right)  # register %rcx = %rsi
3. xor %eax,%eax         # register %eax = 0
4. vmovsd (%rcx),%xmm1   # Load 1 element of B into %xmm1
5. add %r9,%rcx          # register %rcx = %rcx + %r9
6. vmulsd (%r8,%rax,8),%xmm1,%xmm1 # Multiply %xmm1,
   element of A aik          bkj   product
7. add $0x1,%rax         # register %rax = %rax + 1
8. cmp %eax,%edi         # compare %eax to %edi
9. vaddsd %xmm1,%xmm0,%xmm0 # Add %xmm1, %xmm0 sum-of-products
10. jg 30 <dgemm+0x30>   # jump if %eax > %edi   accumulator
11. add $0x1,%r11d       # register %r11 = %r11 + 1
12. vmovsd %xmm0,(%r10)  # Store %xmm0 into C element
            Cij (left)
```

(Inner loop only, i.e. it corresponds to lines 6 to 9 from source code)

How to fully exploit AVX?

- Can be used for multiple FP operands
  - **4 ×** 64-bit **double** precision (*double* in C)
  - 8 × 32-bit single precision (*float* in C)
- How to exploit vector operations?
  - Cannot compute scalar times scalar
  - Classic (scalar) matrix traversal inadequate...

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}	a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}	a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 0 **k = 0**

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

$\mathbf{b_{00}}$	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
$\mathbf{b_{10}}$	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
$\mathbf{b_{20}}$	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
$\mathbf{b_{30}}$	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
$\mathbf{b_{40}}$	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
$\mathbf{b_{50}}$	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3   j = 0   **k = 1**

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}	a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}	a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3   j = 0   **k = 2**

Vectorial traversal

$$\begin{array}{cccccc} c_{00} & c_{01} & c_{02} & c_{03} & c_{04} & c_{05} \\ c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{20} & c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{30} & c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{40} & c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{50} & c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{array} \qquad \begin{array}{cccccc} \mathbf{a_{00}} & \mathbf{a_{01}} & \mathbf{a_{02}} & \mathbf{a_{03}} & \mathbf{a_{04}} & \mathbf{a_{05}} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array} \qquad \begin{array}{cccccc} \mathbf{b_{00}} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ \mathbf{b_{10}} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ \mathbf{b_{20}} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ \mathbf{b_{30}} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ \mathbf{b_{40}} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ \mathbf{b_{50}} & b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{array}$$

i = 0,1,2,3   j = 0   **k = 3**

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 0 **k = 4**

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 0 **k = 5**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 1 **k = 0**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}		$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$		b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}		a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}		b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}		a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}		b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}		a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}		b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}		a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}		b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}		a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}		b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 1 **k = 1**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 1 **k = 2**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 1 **k = 3**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 0,1,2,3 j = 1 **k = 4**

Vectorial traversal

c_{00}	$\mathbf{c_{01}}$	c_{02}	c_{03}	c_{04}	c_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}

$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

$i = 0,1,2,3$ $j = 1$ $\mathbf{k = 5}$

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}	a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	b_{00}	b_{01}	b_{02}	b_{03}	b_{04}	b_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}	a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}

$i = 0,1,2,3$ $j = 31$ **$k = 31$**

Vectorial traversal

c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}	$\mathbf{a_{00}}$	$\mathbf{a_{01}}$	$\mathbf{a_{02}}$	$\mathbf{a_{03}}$	$\mathbf{a_{04}}$	$\mathbf{a_{05}}$	b_{00}	$\mathbf{b_{01}}$	b_{02}	b_{03}	b_{04}	b_{05}
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	b_{10}	$\mathbf{b_{11}}$	b_{12}	b_{13}	b_{14}	b_{15}
c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	b_{20}	$\mathbf{b_{21}}$	b_{22}	b_{23}	b_{24}	b_{25}
c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	b_{30}	$\mathbf{b_{31}}$	b_{32}	b_{33}	b_{34}	b_{35}
c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	b_{40}	$\mathbf{b_{41}}$	b_{42}	b_{43}	b_{44}	b_{45}
c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}	a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	b_{50}	$\mathbf{b_{51}}$	b_{52}	b_{53}	b_{54}	b_{55}

i = 4,5,6,7 j = 0 **k = 0**

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```c
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
              _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
              _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized C code: (exploiting compiler intrinsics)

```
#include <x86intrin.h>
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int i = 0; i < n; i+=4 )
  for ( int j = 0; j < n; j++ ) {
   __m256d c0 = _mm256_load_pd(C+i+j*n); /* c0 = C[i][j] */
   for( int k = 0; k < n; k++ )
    c0 = _mm256_add_pd(c0, /* c0 += A[i][k]*B[k][j] */
             _mm256_mul_pd(_mm256_load_pd(A+i+k*n),
             _mm256_broadcast_sd(B+k+j*n)));
   _mm256_store_pd(C+i+j*n, c0); /* C[i][j] = c0 */
  }
}
```

Exploiting subword parallelism

- Optimized x86 assembly code:

```
vmovapd (%r11),%ymm0           # Load 4 elements of C into %ymm0
mov %rbx,%rcx                  # register %rcx = %rbx
xor %eax,%eax                  # register %eax = 0
vbroadcastsd (%rax,%r8,1),%ymm1 # Make 4 copies of B element
add $0x8,%rax                  # register %rax = %rax + 8
vmulpd (%rcx),%ymm1,%ymm1      # Parallel mul %ymm1,4 A elements
add %r9,%rcx                   # register %rcx = %rcx + %r9
cmp %r10,%rax                  # compare %r10 to %rax
vaddpd %ymm1,%ymm0,%ymm0       # Parallel add %ymm1, %ymm0
jne 50 <dgemm+0x50>            # jump if not %r10 != %rax
add $0x1,%esi                  # register % esi = % esi + 1
vmovapd %ymm0,(%r11)           # Store %ymm0 into 4 C elements
```

[Generated with gcc (2014 version) when using C Intrinsics to induce full AVX exploitation]

(Inner loop only, i.e. it corresponds to lines 6 to 11 from source code)

Impact of subword parallelism

- Experiment: 32 x 32 matrices
  - 2.6GHz Intel Core i7 (Sandy Bridge)
  - Using a single core
- Unoptimized DGEMM
  - 1.7 GigaFLOPS
- Optimized DGEMM
  - 6.4 GigaFLOPS
- Speed up: 3.85 times as fast!