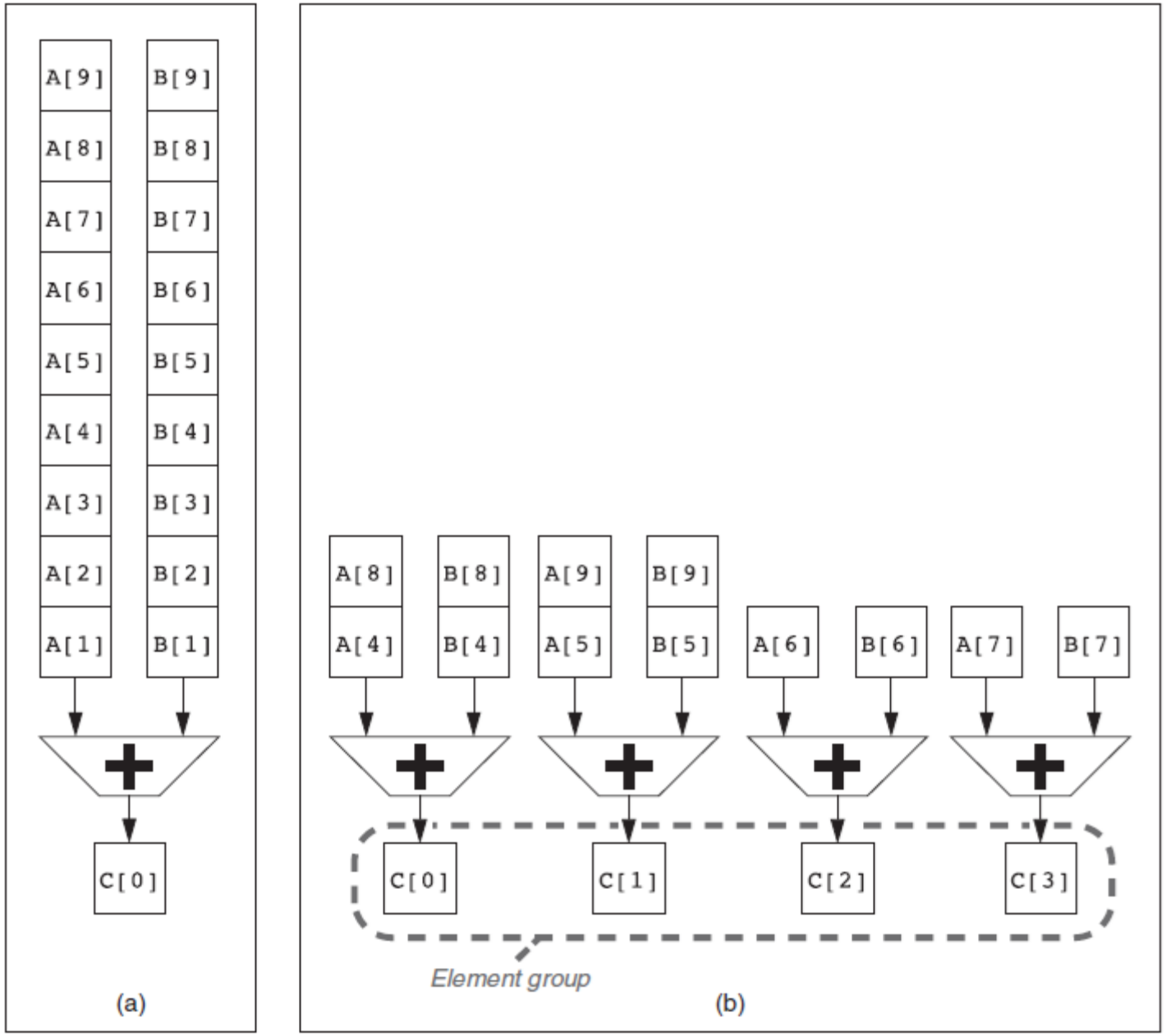
Further Optimizations

- How can a vector processor execute a single vector faster than one element per clock cycle? Multiple elements per clock cycle improve performance.
- How does a vector processor handle programs where the vector lengths are not the same as the length of the vector register (64 for VMIPS)? Since most application vectors don't match the architecture vector length, we need an efficient solution to this common case.
- What happens when there is an IF statement inside the code to be vectorized? More code can vectorize if we can efficiently handle conditional statements.
- What does a vector processor need from the memory system? Without sufficient memory bandwidth, vector execution can be futile.
- How does a vector processor handle multiple dimensional matrices? This popular data structure must vectorize for vector architectures to do well.
- How does a vector processor handle sparse matrices? This popular data structure must vectorize also.
- How do you program a vector computer? Architectural innovations that are a mismatch to compiler technology may not get widespread use.

Multiple Lanes

- Each lane processes a subset of vector elements in parallel
- Four lanes reduce execution time by 4× compared to a single lane
- Requires minimal increase in control complexity
- Allows trading off area, clock rate, voltage, and energy without sacrificing peak performance



Vector-Length Registers

```
for (i=0; i <n; i=i+1)
  Y[i] = a * X[i] + Y[i];
```

- The value of n may be unknown at compile-time and not a multiple of the architectural vector length
- Vector-length registers (VLR) solve the problem of vector lengths not matching the architectural vector length

```
low = 0;
VL = (n % MVL);
for (j = 0; j <= (n/MVL); j=j+1) {
  for (i = low; i < (low+VL); i=i+1)
    Y[i] = a * X[i] + Y[i] ;
  low = low + VL;
  VL = MVL;
}
```

- Strip mining
  - Generation of code such that each vector operation is done for a size less than or equal to the maximum vector length (MVL)

Vector Mask Registers

```
for (i = 0; i < 64; i=i+1)
  if (X[i] != 0)
    X[i] = X[i] – Y[i];
```

Vector Mask Enabled

- This loop cannot normally be vectorized because of the conditional execution of the body
- If the inner loop could be run for the iterations for which X[i] ≠ 0, then the subtraction could be vectorized.

```
LV        V1,Rx        ;load vector X into V1
LV        V2,Ry        ;load vector Y
L.D       F0,#0        ;load FP zero into F0
SNEVS.D   V1,F0        ;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D   V1,V1,V2     ;subtract under vector mask
SV        V1,Rx        ;store the result in X
```

Memory System for Vector Processors

1

Memory Banks

Multiple independent memory banks supply bandwidth for vector loads/stores

- Support multiple simultaneous accesses
- Allow non-sequential memory access patterns
- Support multiple processors sharing memory

2

Bank Conflicts

Occur when multiple accesses target the same bank

- Can significantly reduce effective memory bandwidth
- Worst case: stride is a multiple of number of banks
- Best case: unit stride with no conflicts

Example: Cray T932 required 1344 memory banks to support 32 processors with 6 references per processor!

Stride: Handling Multidimensional Arrays

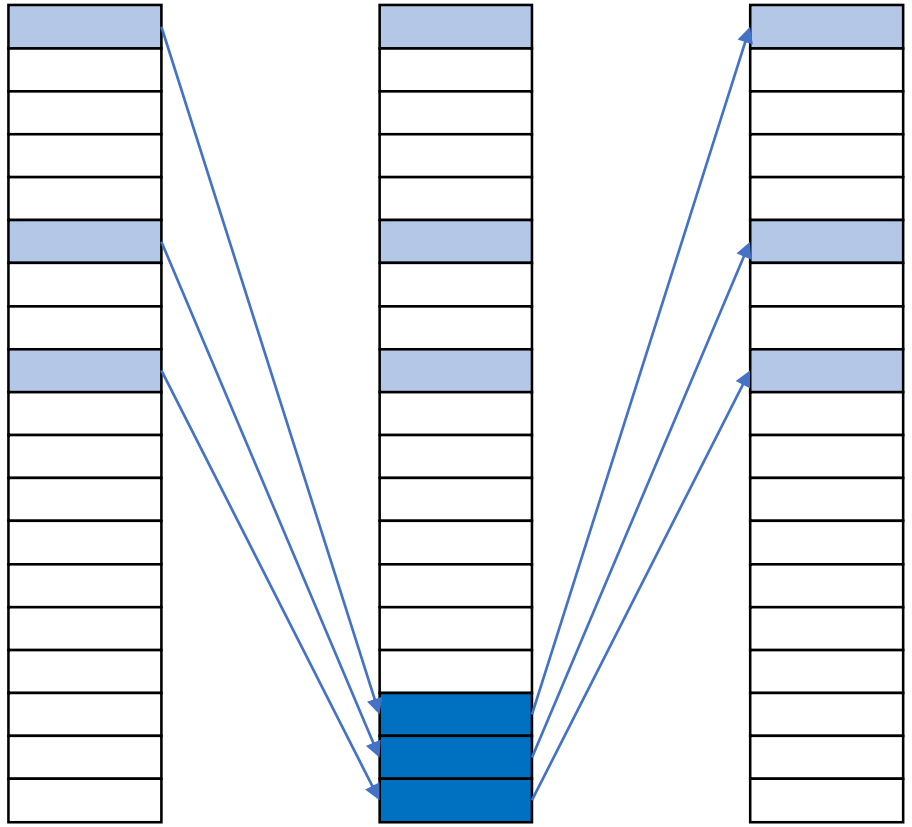
```
for (i = 0; i < 100; i=i+1)
  for (j = 0; j < 100; j=j+1) {
    A[i][j] = 0.0;
    for (k = 0; k < 100; k=k+1)
      A[i][j] = A[i][j] + B[i][k] * D[k][j];
  }
```

- In row-major order (as in C):
  - B has stride of 1 (adjacent elements)
  - D has stride of 100 (elements separated by row size)
- Vector processors support non-unit strides:
  - LVWS (load vector with stride)
  - SVWS (store vector with stride)

Gather-Scatter: Handling Sparse Matrices

```
for (i = 0; i < n; i=i+1)
  A[K[i]] = A[K[i]] + C[M[i]];
```

```
LV      Vk, Rk       ;load K
LVI     Va, (Ra+Vk)  ;load A[K[]]
LV      Vm, Rm       ;load M
LVI     Vc, (Rc+Vm)  ;load C[M[]]
ADDVV.D Va, Va, Vc   ;add them
SVI     (Ra+Vk), Va  ;store A[K[]]
```



Programming Vector Architectures

- Compilers can identify vectorizable code at compile time
- Programmers receive feedback on why code didn't vectorize
- Programmers can provide hints when dependencies are safe to ignore
- Clear performance model helps optimize code

Benchmark name	Operations executed in vector mode, compiler-optimized	Operations executed in vector mode, with programmer aid	Speedup from hint optimization
BDNA	96.1%	97.2%	1.52
MG3D	95.1%	94.5%	1.00
FLO52	91.5%	88.7%	N/A
ARC3D	91.1%	92.0%	1.01
SPEC77	90.3%	90.4%	1.07
MDG	87.7%	94.2%	1.49
TRFD	69.8%	73.7%	1.67
DYFESM	68.8%	65.6%	N/A
ADM	42.9%	59.6%	3.60
OCEAN	42.8%	91.2%	3.92
TRACK	14.4%	54.6%	2.52
SPICE	11.5%	79.9%	4.06
QCD	4.2%	75.1%	2.15