Data Locality

- Application data space is often too large to all fit into cache
- Key to minimizing memory access costs is data locality (re-use of data in registers and cache)
- Types of Data Locality:
  - Register Locality: Multiple refs to data element in a register
  - Cache-Temporal Locality: Multiple refs to data element in cache (without access to main memory)
  - Cache-Spatial Locality: Ref to data with address close to another recently accessed data element
- Attempt (in order of preference) to:
  - Keep all data in registers
  - Order algorithm to do all ops on a data element before it is overwritten in cache
  - Order algorithm so that successive operations are on physically contiguous data
Single Processor Performance Enhancement

Two fundamental issues:

- Adequate op-level parallelism (to keep super-scalar, pipelined processor units busy)
- Minimize memory-access costs (about an order of magnitude higher than clock cycle)

Three useful techniques:

- Loop Permutation
- Loop Unrolling
- Loop Blocking (tiling)
Access Stride and Spatial Locality

- Access stride: Separation between successively accessed memory locations
- Unit access stride maximizes spatial locality (only one miss per line)
- 2-D arrays have different linearized representations in Fortran and C
- Fortran’s column-major rep favors column-wise access of data

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
<tr>
<td>m</td>
<td>n</td>
<td>o</td>
<td>p</td>
</tr>
</tbody>
</table>
```

Row Major Order (C)

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>o</th>
<th>p</th>
</tr>
</thead>
</table>
```

Column-major Order (Fortran)
Matrix-Vector Multiply: Dot-Product Form

\[
\begin{align*}
do i &= 1, n \\
do j &= 1, n \\
y(i) &= y(i) + A(i, j) \times x(j) \\
end do \\
end do
\end{align*}
\]

Access Stride for Arrays

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>n</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Matrix-Vector Multiply: SAXPY Form

\[
\begin{align*}
&\text{do } j = 1,n \\
&\quad \text{do } i = 1,n \\
&\quad \quad y(i) = y(i) + A(i,j)\times x(j) \\
&\quad \text{end do} \\
&\text{end do}
\end{align*}
\]

Access Stride for Arrays

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>n</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Loop Permutation

- Changing the order of nested loops changes the strides of array access; choose one that maximizes spatial locality

C Matrix-Matrix Multiply
   do i = 1,n
      do j = 1,n
         do k = 1,n
            C(i,j) = C(i,j) + A(i,k)*B(k,j)
         end do
      end do
   end do

<table>
<thead>
<tr>
<th>Reference</th>
<th>ijk</th>
<th>jik</th>
<th>ikj</th>
<th>kij</th>
<th>jki</th>
<th>kji</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(i,j)</td>
<td>0</td>
<td>0</td>
<td>n</td>
<td>n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A(i,k)</td>
<td>n</td>
<td>n</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B(k,j)</td>
<td>1</td>
<td>1</td>
<td>n</td>
<td>n</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Access Stride for Arrays (Fortran: Column-Major)

<table>
<thead>
<tr>
<th>ijk</th>
<th>jik</th>
<th>ikj</th>
<th>kij</th>
<th>jki</th>
<th>kji</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>36</td>
<td>10</td>
<td>10</td>
<td>450</td>
<td>500</td>
</tr>
</tbody>
</table>

Performance (MFLOPS) on oscbw Pentium 4
Loop Unrolling

- Reduce number of iterations of loop but add statement(s) to loop body to do work of missing iterations
- Increase amount of op-level parallelism in loop body
- Outer-loop unrolling changes the order of access of memory elements and could reduce number of memory accesses and cache misses

```plaintext
do i = 1,2n
   do j = 1,m
      Loop-Body(i,j)
   end do
end do
```

```plaintext
do i = 1,2n,2
   do j = 1,m
      Loop-Body(i,j)
      Loop-Body(i+1,j)
   end do
end do
```
Example: Outer Loop Unrolling

C Assumes n is a multiple of 4
C 4-outer unrolled Dot-Product form of MV
   do i = 1,n,4
      do j = 1,n
         y(i) = y(i) + a(i,j)*x(j)
         y(i+1) = y(i+1) + a(i+1,j)*x(j)
         y(i+2) = y(i+2) + a(i+2,j)*x(j)
         y(i+3) = y(i+3) + a(i+3,j)*x(j)
      end do
   end do
C 4-outer unrolled SAXPY form of MV
   do j = 1,n,4
      do i = 1,n
         y(i) = y(i) + a(i,j)*x(j) + a(i,j+1)*x(j+1)
         &       + a(i,j+2)*x(j+2)
         &       + a(i,j+3)*x(j+3)
      end do
   end do