Software optimizations through loop interchange

for (j = 0; j < n; j = j+1)
for (i = 0; i < n; i = i+1)
C += x[i][j];

Row-wise storage
Block size = 4
Cache size < 4n
Miss rate = 100%

Take advantage of spacial locality

for (i = 0; i < n; i = i+1)
for (j = 0; j < n; j = j+1)
C += x[i][j];

Miss rate = 25%

Software optimization through blocking (partitioning)

For simplicity, assume that the cache block size = 1 word

for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
{r = 0;
 for (k = 0; k < n; k++)
   r = r + A[i][k]*B[k][j];
 C[i][j] += r; }

Data used when i = 0, j = 0, ..., n-1

Data used when i = 1, j = 0, ..., n-1

• Assume that the cache can fit at least one row of A (perfect reuse)
• B will “thrash” if cache size < n² (all of B does not fit in cache)
• Every element of B will be used only once when brought to cache
Software optimization through blocking (partitioning)

Partition the matrices into submatrices of size\( psize \times psize \):

```c
for (si = 0; si < n; si =+ psize)
    for (sj = 0; sj < n; sj =+ psize)
        for (sk = 0; sk < n; sk =+ psize)
            for (i=si ; i < si+psize ; i++)
                for (j=sj ; j < sj+psize ; j++)
                {
                    r = 0;
                    for (k=sk; k < sk+psize; k++)
                        r = r + A[i][k]*B[k][j];
                    C[i][j] = C[i][j] + r;
                }
```

If cache size > 2 \( n \times psize \), then
- A will be perfectly reused
- Each element of B will be reused \( "psize" \) times (reduce miss rate)

Dependable memory hierarchy (sec. 5.5)

- Fault: failure of a component
- Error: manifestation of a fault
- Faults may or may not lead to system failure

- Reliability measure: mean time to failure (MTTF)
- Repair efficiency: mean time to repair (MTTR)
- Mean time between failures

\[
MTBF = MTTF + MTTR
\]

- Availability = \( \frac{MTTF}{MTTF + MTTR} \)

- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair
Error detection and correction Code

- Hamming distance: Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection.
  - Example: even parity code

\[
\begin{align*}
\text{000} & \rightarrow \text{000} \\
\text{001} & \rightarrow \text{001} \\
\text{010} & \rightarrow \text{010} \\
\text{011} & \rightarrow \text{011} \\
\text{100} & \rightarrow \text{100} \\
\text{101} & \rightarrow \text{101} \\
\text{110} & \rightarrow \text{110} \\
\text{111} & \rightarrow \text{111}
\end{align*}
\]

- Eight of the sixteen 4-bit code words are invalid (distance 2 code)
- Any single bit flip in a valid code will produce an invalid code
  - Hence, single error detection --- but cannot correct the error

Minimum distance ≥ 3 provides single error correction (SEC)

Minimum distance ≥ 4 provides single error correction (SEC) and double error detection (DEC)

2-bit → 5-bit encoding

2-bit → 6-bit encoding
Hamming (Single error correcting) code

- To calculate Hamming code:
  - Number the bits from 1 to 12
  - All bit positions that are a power of 2 are parity bits, the others are data bits
  - Each parity bit is set so that a certain group of data bits have even parity.

<table>
<thead>
<tr>
<th>Bit position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoded data bits</td>
<td>p1</td>
<td>p2</td>
<td>d3</td>
<td>p4</td>
<td>d5</td>
<td>d6</td>
<td>d7</td>
<td>d8</td>
<td>p9</td>
<td>p10</td>
<td>p11</td>
<td>p12</td>
</tr>
<tr>
<td>Parity bit coverset</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Example:

Data bits 10100011 are encoded as 01100100011

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>1</th>
<th>p4</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>p8</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

s1 = 0 (parity is correct)
s2 = 1 (parity is not correct)
s3 = 0 (parity is correct)
s4 = 1 (parity is not correct)

- If all syndrome bits are zeroes, then there is no error
- Otherwise, the syndrome bits indicate the position of the bit in error
- In our example s4, s3, s2, s1 = 1010 = 10 → bit 10 is the wrong bit

Not magic!!
There is a theory behind that
Hamming SEC/DEC Code

- Hamming code cannot detect two errors (distance < 4)
- Add an additional parity bit for the whole word ($p_n$)
- Make Hamming distance = 4
- Decoding:
  - Let $H = SEC$ parity bits
    - $H = 0$ and parity is correct $\Rightarrow$ no error
    - $H > 0$ and parity is not correct $\Rightarrow$ single correctable error
    - $H > 0$ and parity is correct $\Rightarrow$ double uncorrectable error
    - $H = 0$ and parity is not correct $\Rightarrow$ error in the parity bit

- Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits