

ENCE360 Computer Systems



- Architecture -

Performance
Optimisation

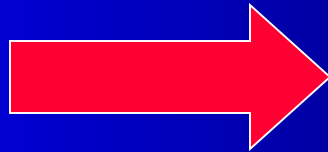
Performance Optimisation

- Data structure reorganisation
- Data alignment
- Strip mining
- Loop blocking
- Code using Intel SIMD instructions
(Single Instruction Multiple Data)

Data structure reorganisation

- Maximise cache hits by keeping data accesses contiguous:

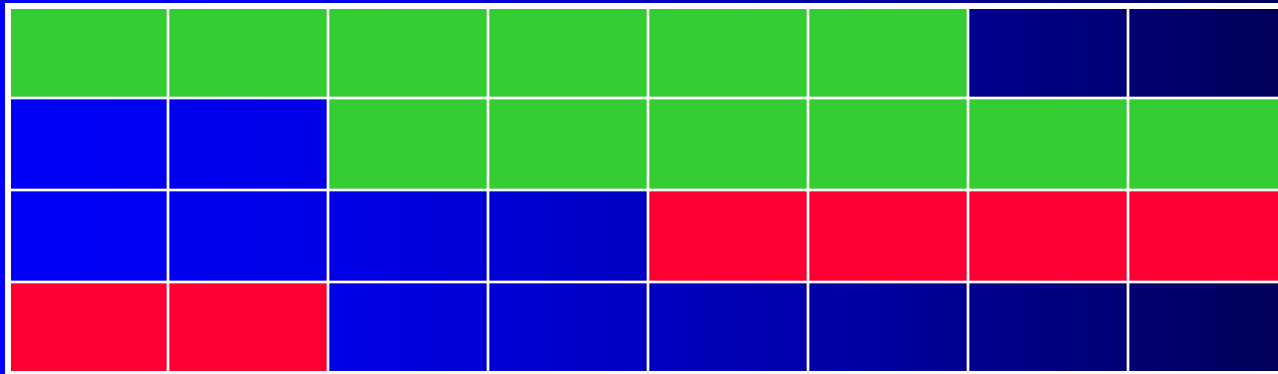
```
typedef struct{  
    float x,y,z;  
    int a,b;  
} Vertex;  
Vertex Vertices[NumVs];
```



```
typedef struct{  
    float x[NumVs];  
    float y[NumVs];  
    float z[NumVs];  
    int a[NumVs];  
    int b[NumVs];  
} VerticesList;  
VerticesList Vertices;
```

Data Alignment

- Try to keep data within 1 line of cache:



Organise data to fit on a word boundary or cache line to enable data to be retrieved with the minimum number of accesses \Rightarrow maximises cache hit rate.

Strip Mining

- Combine loops and iterate over cache size:

```
typedef struct _VERTEX {
float x, y, z, nx, ny, nz, u, v;
} Vertex_rec;

main() {
    Vertex_rec v[Num];
    for (i=0; i<Num; i++) {
        transform(v[i]);
    }
    for (i=0; i<Num; i++) {
        lighting(v[i]);
    }
}
```

Strip mining 2

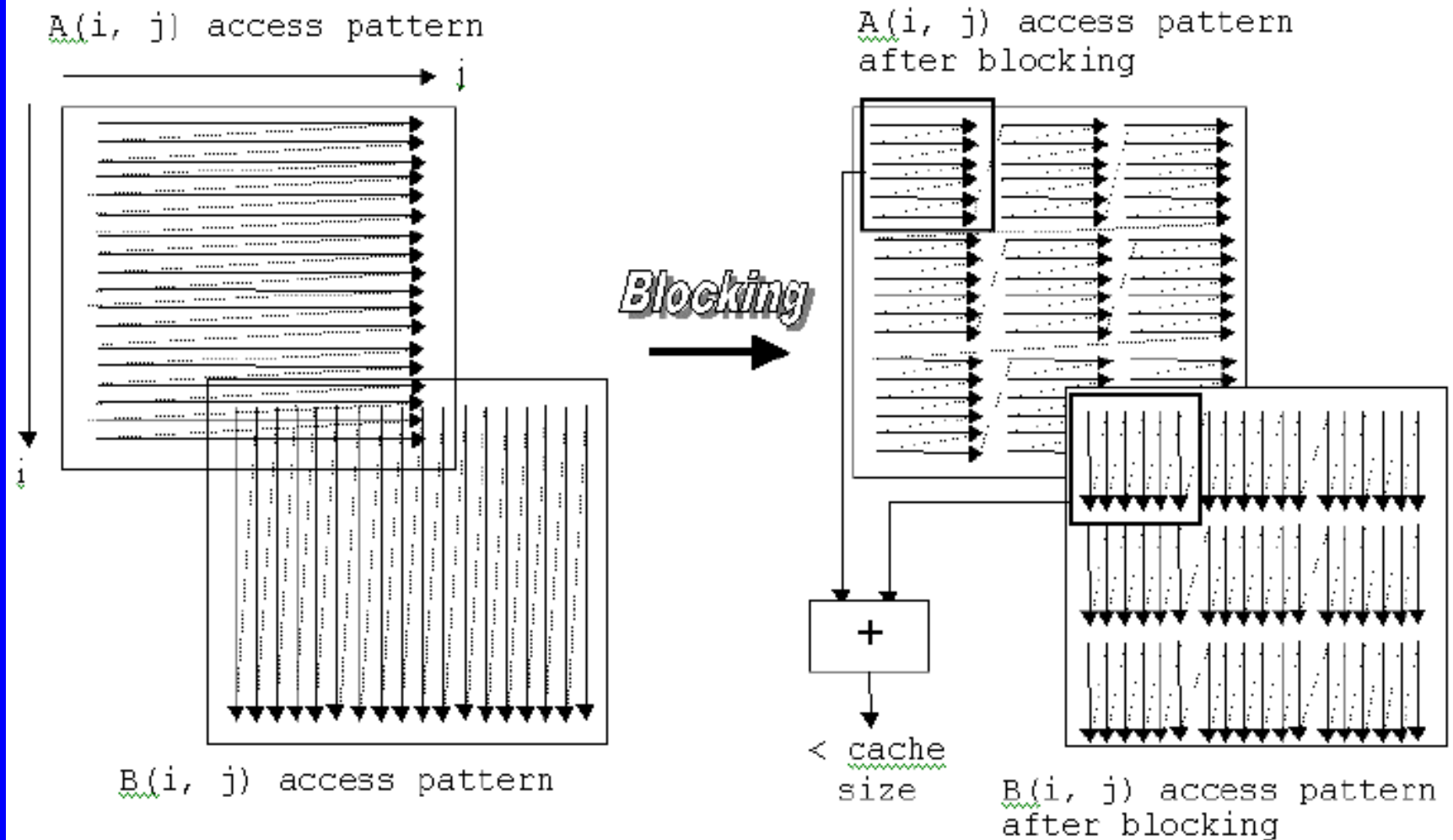
- Combine loops and iterate over cache size:

```
main()
{
    Vertex_rec v[Num];
    for (i=0; i < Num; i+=strip_size) {
        for (j=i; j < min(Num, i+strip_size); j++) {
            transform(v[j]);
        }
        for (j=i; j < min(Num, i+strip_size); j++) {
            lighting(v[j]);
        }
    }
}
```

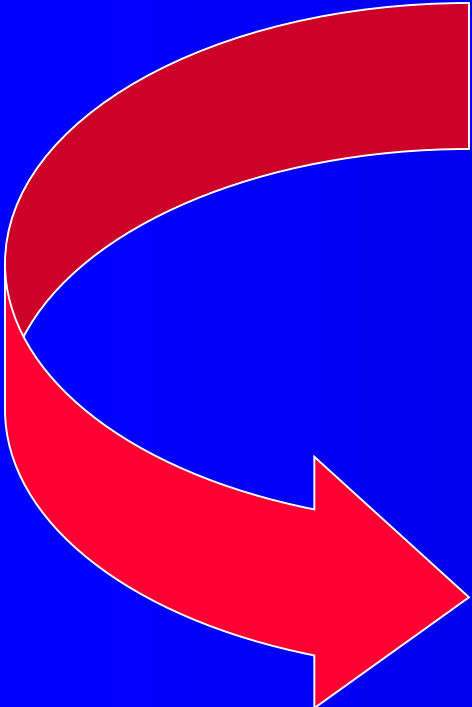
Loop Blocking

- Large, non-contiguous memory accesses are BAD
- E.g. two large arrays A and B, add $A(x,y)$ to $B(y,x)$
- Solution: reduce the “span” of accessed memory

Loop Blocking (2)



Loop Blocking (3)



```
float A[MAX, MAX], B[MAX, MAX]
...
for (i=0; i< MAX; i++) {
    for (j=0; j< MAX; j++) {
        A[i,j] = A[i,j] + B[j, i];
    }
}
```

```
float A[MAX, MAX], B[MAX, MAX];
for (i=0; i< MAX; i+=block_size) {
    for (j=0; j< MAX; j+=block_size) {
        for (ii=i; ii<i+block_size; ii++) {
            for (jj=j; jj<j+block_size; jj++) {
                A[ii,jj] = A[ii,jj] + B[jj, ii];
            }
        }
    }
}
```

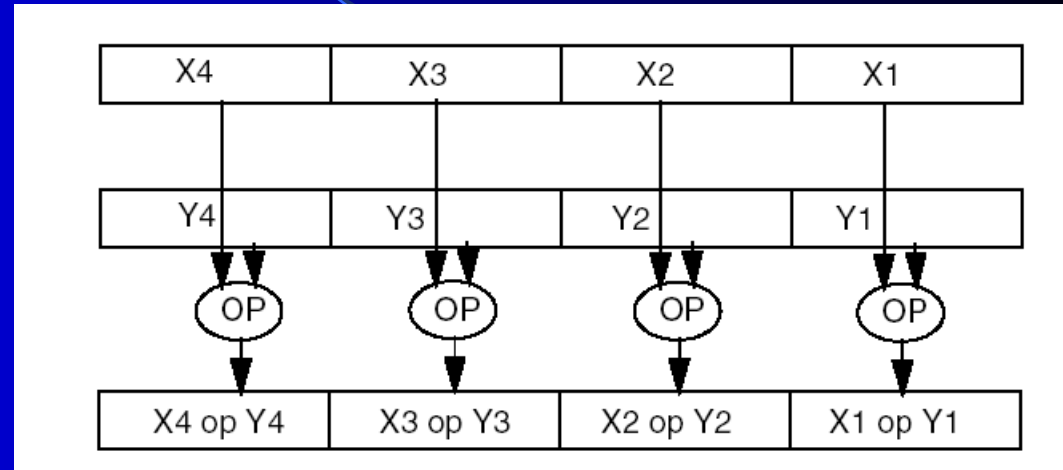
Loop unrolling

- Unroll small loops until branching is less than 10% of execution time
- Unroll hot (inner) loops to 16 or less iterations
- Reduces the number of conditional branch/jump instructions
- Maximises the use of BTB
- Reduces delay caused by branch/jump instructions (less 'delay slots').

Intel Pentium Overview

SSE: Streaming SIMD Extensions

- Extra set of 256 bit MMX registers



- SSE uses added instructions to perform (“streamed”) **Single Instruction, Multiple Data (SIMD)** direct from memory (a supercomputer technique employed to achieve data level parallelism)
(Added instructions to control memory, cache store)

Intel IPP = Integrated Performance Primitives

SIMD = **S**ingle **I**nstruction, **M**ultiple **D**ata (256bit registers)
is used by **SSE** = **S**teaming **SIMD** **E**xtensions (in silicon).

SSE is used by **IPP** (Intel software libraries) for:

- Vector/Matrix Mathematics
- Signal Processing
- Computer Vision
- Speech Recognition
- Data Compression
- Video, Audio Decode/Encode
- Cryptography
- Ray Tracing/Rendering
- String Processing

Counterparts:

Sun: mediaLib for Solaris, **Apple**: vDSP, vImage etc. for Mac OS, **AMD**: AMD Performance Library (APL)

Code Using SSE

- **YOU** have to do it (or your compiler)
- Use if you have code fragments that:
 - are computationally intensive
 - are executed often enough to have an impact on performance
 - require integer computations with little data-dependent control flow
 - require floating-point computations
 - **require help in using the cache hierarchy efficiently.**

A collection of other technologies that increase parallelism

- “hyperpipelined” for headroom to 10GHz
- High performance bus architecture
- Rapid execution engine (for small instructions)
- Speculative execution
- Superscalar
- Hardware register renaming
- Hardware prefetching
- Hyperthreading

The real world

- Tuning OS and applications is a complex art, with lots of conflicting compromises to consider
- Use software tuning tools to find problems and optimise parallelism

Example Exam Questions

Explain what each of the following means and describe how they help speed performance in modern processors:

- a) Loop unrolling
- b) Data structure reorganisation
- c) Data alignment
- d) Strip mining
- e) Loop blocking

Solution (a) Loop Unrolling

Copy loop content multiple times to reduce number of loops.

For example, instead of a small loop repeated many times, ‘unroll’ it by repeating the loop code inside – then need less iterations of this larger ‘unrolled loop’. This reduces the number of conditional branch/jump instructions – maximises the use of BTB – reduces delay caused by branch/jump instructions (less ‘delay slots’).

Solution (b)

Data structure reorganisation

Restructures data which is accessed together so that it is stored contiguously. This enables the cache to be filled more efficiently and maximises the cache hit rate.

For example, change an array of structures (where data items are separated by the size of the structure) into a structure of arrays:

```
struct x{  
    int x;  
    int y;  
}
```

```
x array[10];
```

```
struct x{  
    int x[10];  
    int y[10];  
}
```

```
x array;
```

Solution (c) Data alignment

Data alignment is organising data to fit on a word boundary or cache line.

This enables data to be retrieved with the minimum number of accesses – maximises cache hit rate.

Solution (d) Strip mining

Instead of multiple loops through one huge block of data, strip mining breaks one huge array into strips or sections.

For example:

```
for (i=0; i<Num; i++)  
    transform(v[i]);  
for (i=0; i<Num; i++)  
    lighting(v[i]);
```

Instead, Combine loops and iterate over cache size:

```
for (i=0; i < Num; i+=strip_size){  
    for (j=i; j < min(Num,i+strip_size); j++)  
        transform(v[j]);  
    for (j=i; j < min(Num,i+strip_size); j++)  
        lighting(v[j]);  
}
```

This saves loading the same page from memory into the cache multiple times – helps to maximise the cache hit rate.

Solution (e) Loop blocking

Instead of one loop through multiple huge blocks of data, loop blocking breaks loops into accessing smaller sections of these huge blocks of data. Working on only a small part of both huge arrays at the same time can prevent excessive cache fetching of either array – maximising the cache hit rate to significantly increase speed.

For example:

```
for (i=0; i<MAX; i++)  
  for (j=0; j<MAX; j++)  
    A[i,j] = A[i,j] + B[j,i];
```

Instead both bits fit in cache:

```
for (i=0; i<MAX; i+=block_size)  
  for (j=0; j<MAX; j+=block_size)  
    for (ii=i; ii<i+block_size; ii++)  
      for (jj=j; jj<j+block_size; jj++)  
        A[ii,jj] = A[ii,jj] + B[jj,ii];
```

