Optimisation

Performance Tuning

Constant Elimination

```
do i=1,n
  a(i) = 2*b*c(i)
  enddo
```

What is wrong with this loop?

Compilers can move simple instances of constant computations outside the loop. Others need to be done manually.

Merging Expensive Operations

Eg.

division

modulo

sqrt

transcendental functions

compute $V(r) = \frac{1}{r^{12}} - \frac{1}{r^6}$

```
r3inv = 1/(r*r*r)
V = r3inv * r3inv - r3inv
```

not

```
V = 1/r ** 12 - 1/r ** 6
```

Special case functions

Replace special case functions with faster algorithms

eg.

`x * x` is faster than `x**2` ≡ `exp(2*log(x`

`sqrt(x)` is faster than `x**.5` ≡ `exp(.5*log`

`iand(x,63)` is faster than `mod(x,64)`

`x & 63` is faster than `x % 64`

In C++, pow(double,int) may be more efficient than the standard pow(double,double).

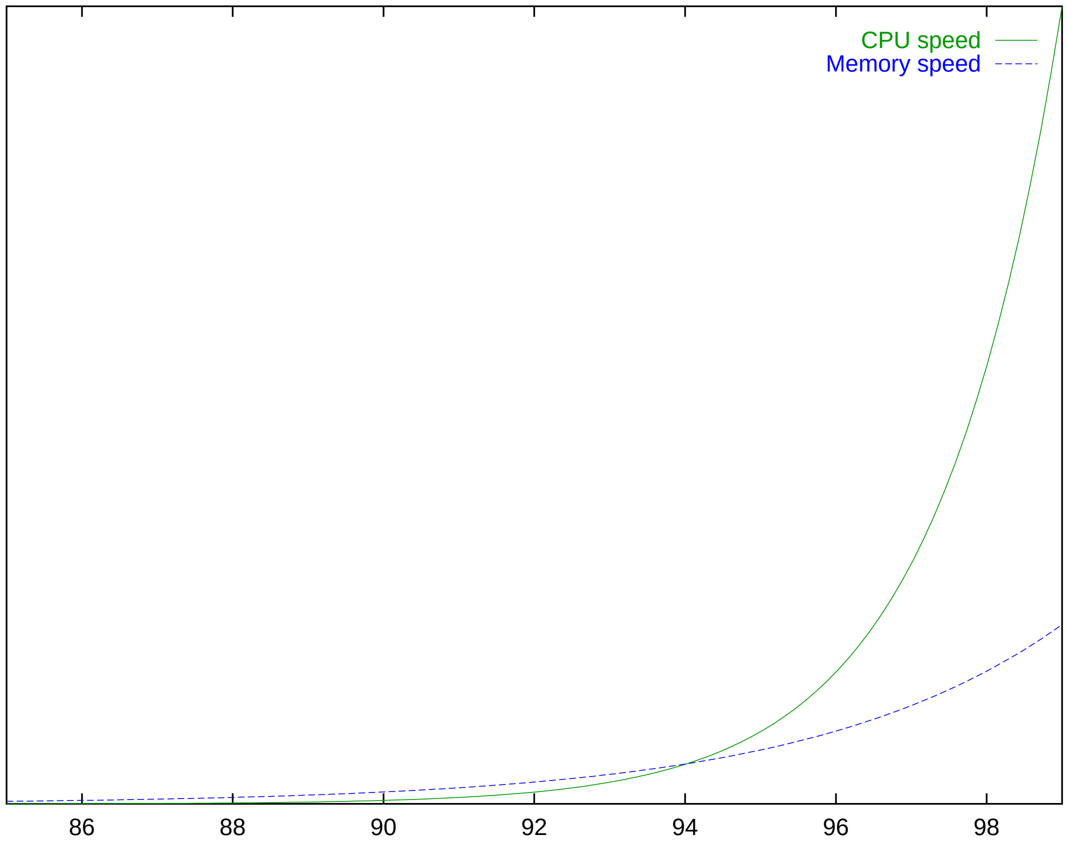
Fortran compilers should be able to recognise x**i as a special case.

Optimised Libraries

Don’t reinvent the wheel!

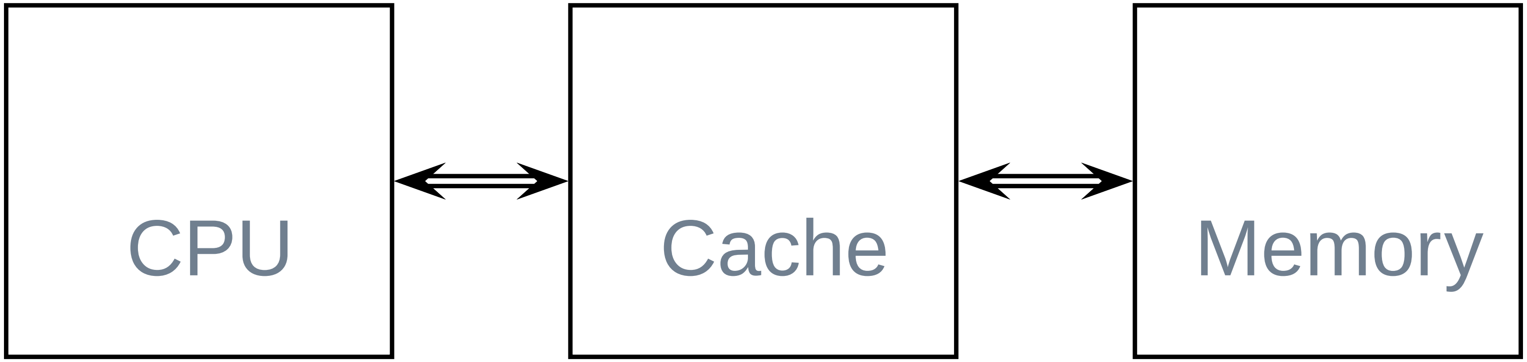
Well optimised libraries include BLAS, LAPACK and FFTW.

The Cache



DRAM is much cheaper than SRAM, but it is also much slower. Therefore place a small SRAM cache near processor.

Cache. . .



Vector CPUs usually use SRAM for all memory, and bank it to b. . . ery.

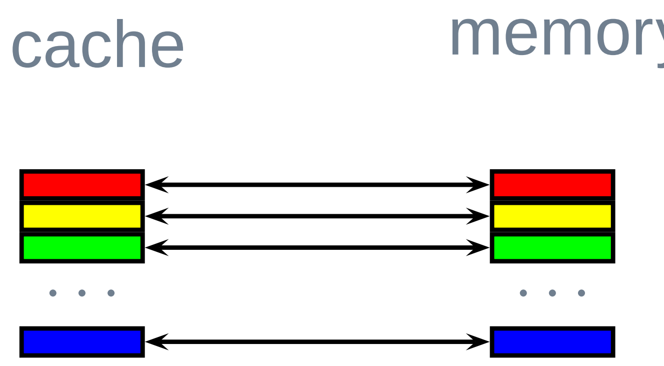
Memory-Cache mapping

The cache is partitioned up into chunks of size c called *cache-lines*. In the simplest caching scheme, every memory location x is mapped to a specific cache line l , along the lines of:

$$l = (x \bmod s)/c$$

where s is the size of the cache.

A status register records if the cache line has been written to (*dirty*) and so needs to be flushed back to main memory before that line can be reused for another part of memory.



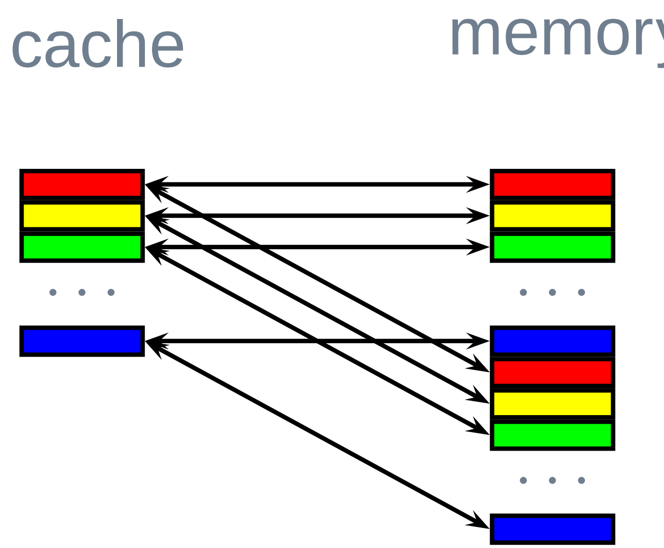
Memory-Cache mapping

The cache is partitioned up into chunks of size c called *cache-lines*. In the simplest caching scheme, every memory location x is mapped to a specific cache line l , along the lines of:

$$l = (x \bmod s)/c$$

where s is the size of the cache.

A status register records if the cache line has been written to (*dirty*) and so needs to be flushed back to main memory before that line can be reused for another part of memory.



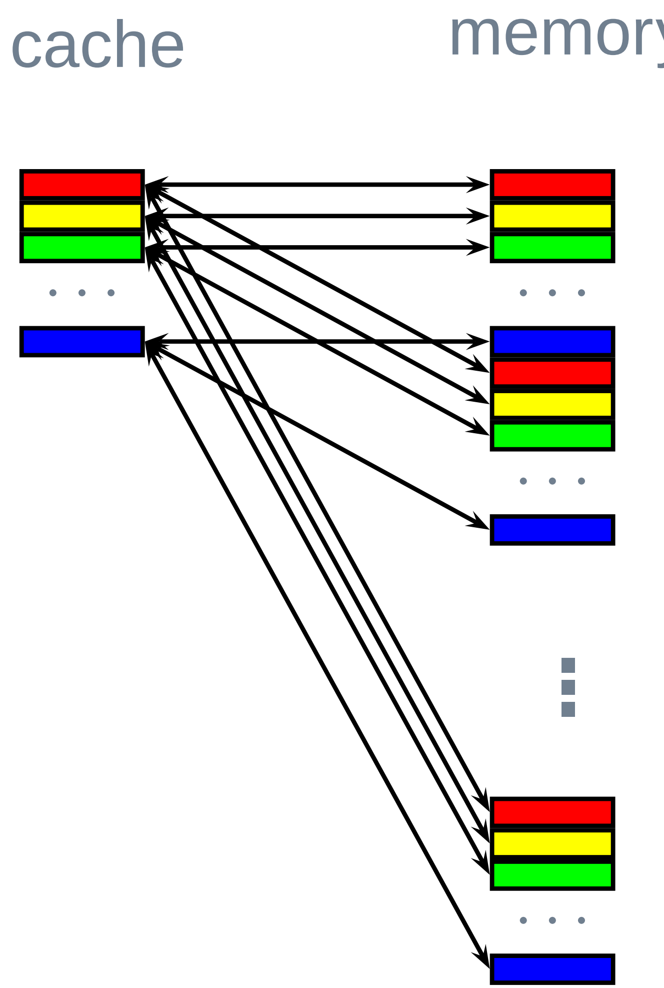
Memory-Cache mapping

The cache is partitioned up into chunks of size c called *cache-lines*. In the simplest caching scheme, every memory location x is mapped to a specific cache line l , along the lines of:

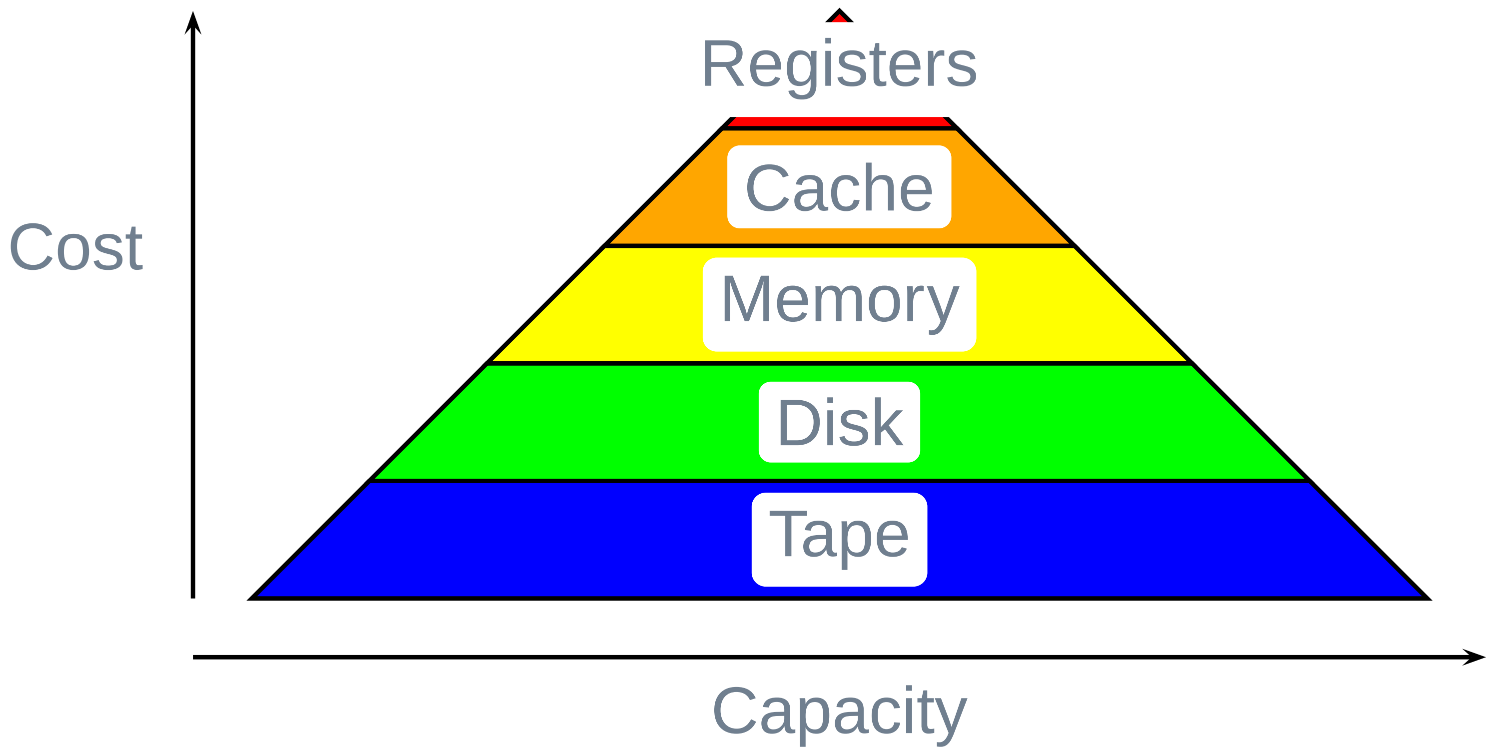
$$l = (x \bmod s)/c$$

where s is the size of the cache.

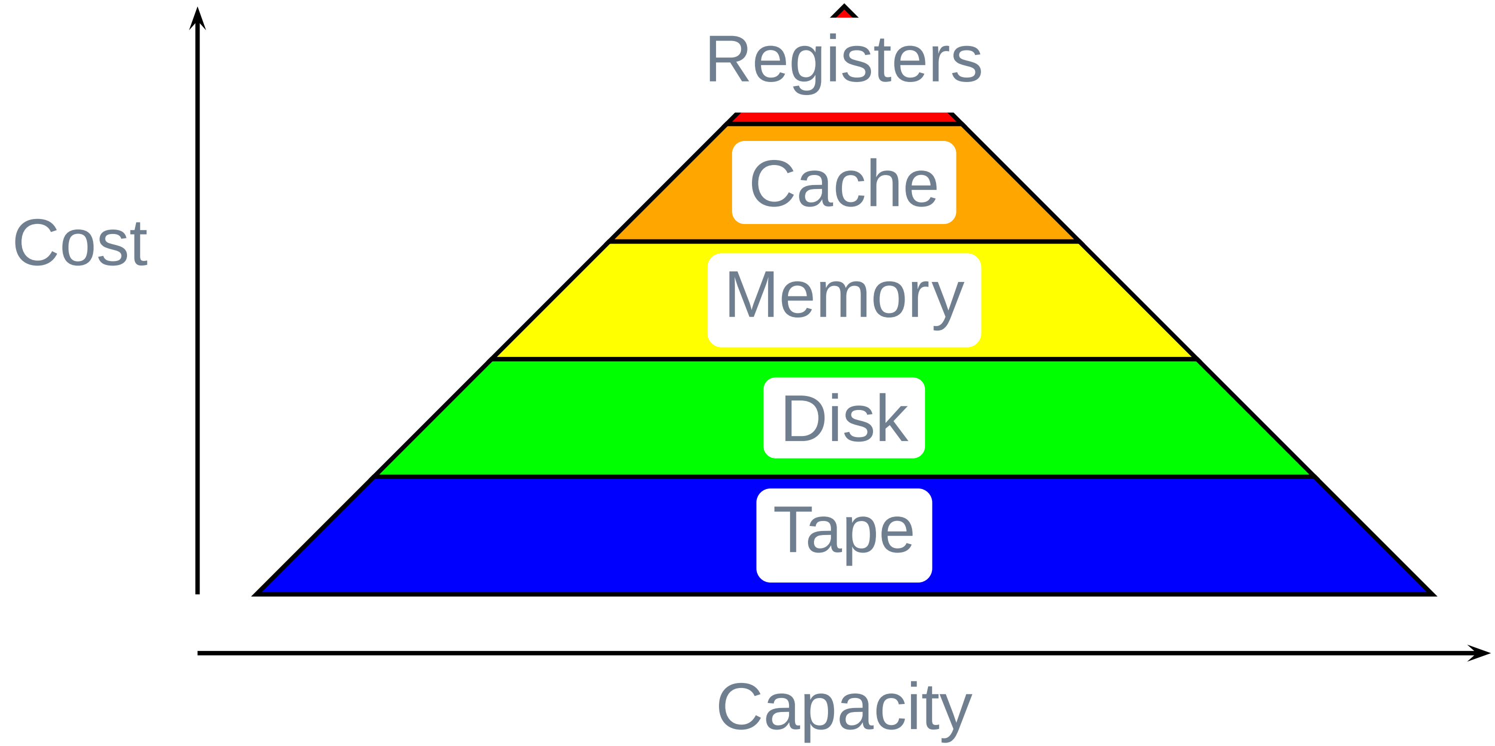
A status register records if the cache line has been written to (*dirty*) and so needs to be flushed back to main memory before that line can be reused for another part of memory.



Memory Heirarchy



Memory Heirarchy



- Arrange data locally (eg use stride 1 if possible)
- Avoid strides that are multiples of cache line size/page size (typically a power of two)

Array Padding

```
integer::parameter n=1024*1024
real*8 a(n),b(n),c(n)
do i=1,n
   a(i)=b(i)+c(i)
```

- These arrays are 8MB in size. Napier has a 2MB cache.
- Each array overlaps in cache a(i) has the same cache location as b(i) and c(i). The cache line will be flushed 3 times each iteration!

Array Padding...

```
integer::parameter n=1024*1024
real*8 a(n),space1(16),b(n)
real*8 space2(16),c(n)
common /foo/a,space1,b,space2,c
do i=1,n
   a(i)=b(i)+c(i)
```

- This ensures a(i) is on a different cache line to b(i) and c(i)
- Similarly it may be sensible to add an extra row to a higher dimensional array: `real*8 a(129,128)` rather than `real*8 a(128,128)`

Prefetching

- The latency involved in a cache miss can be hidden by issuing a load instruction several instructions ahead of the data actually being needed.
- The optimiser will usually take care of this for you
- This can be simulated in your source code, but its difficult to arrange this without interference from the optimiser.

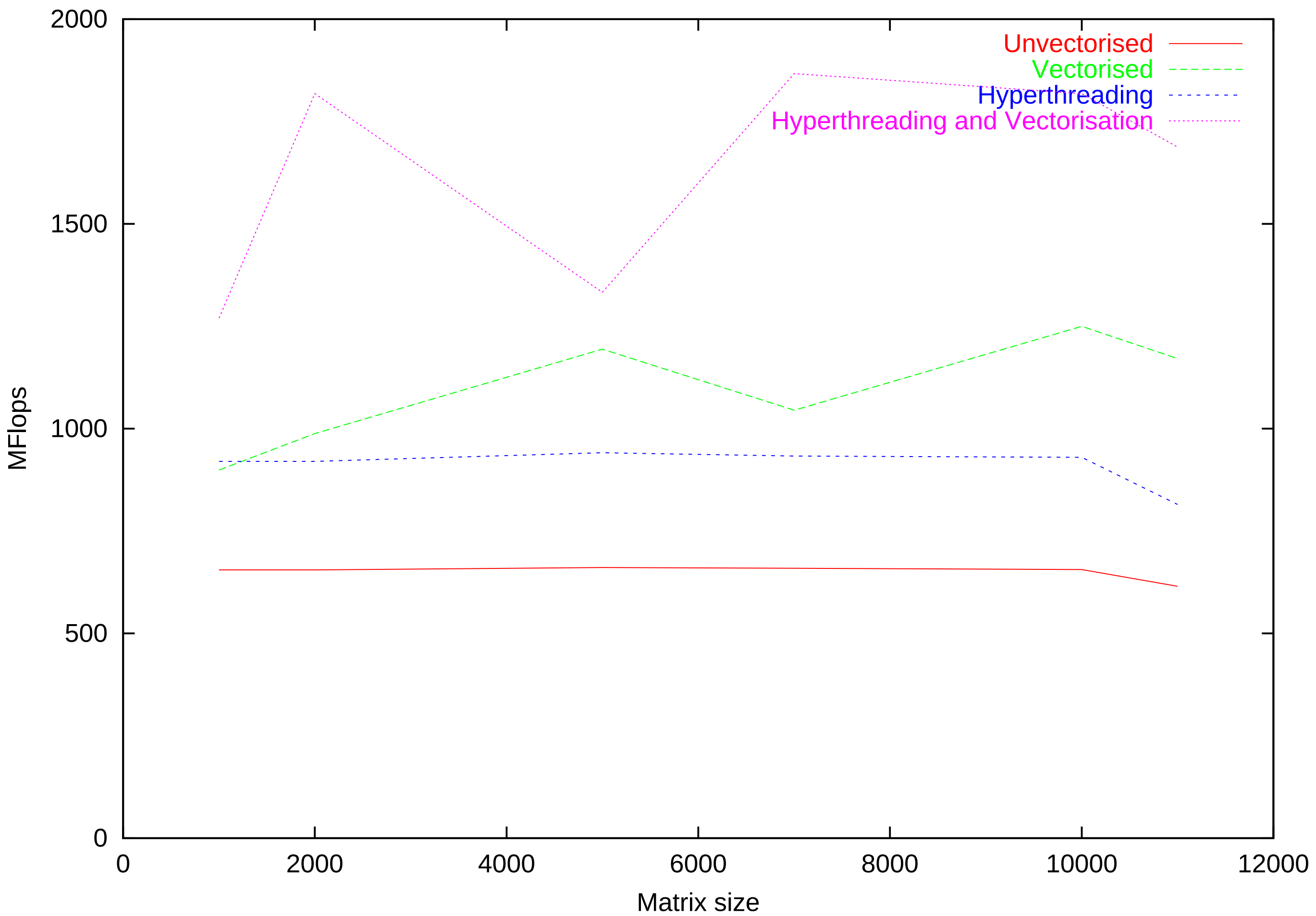
Hyperthreading

- Technology invented by Tera corporation, and bought by Intel.
- Implemented in latest Pentium IV CPUs
- When a thread stalls due to a cache miss, CPU switches to another thread.
- Compile program with -openmp or -parallel, and run on 2 threads per CPU

Hyperthreading example

Single precision Matmul compiled with

`ifc -O3 -tpp7 -unroll -openmp -vec -axW -xW`



Its a bit more complicated...

- Modern CPUs have multiple cache levels (L1, L2, etc.).
- Addresses used at machine language level are *virtual*. Virtual addresses are mapped to physical address by the *virtual memory manager*. Mapped addresses are cached in the *Translation Lookaside Buffer* (TLB).
- The effect of the TLB is like a large cache

Inlining

Subroutine & Function calls degrade performance

- Call and return instructions add overheads
- Pushing arguments onto stack and setting stackframe add overheads
- Breaks software pipelines
- Inhibits parallelisation (ameliorated with PURE)

Small functions/subroutines should be inlined

Inlining...

- Compilers usually do inlining at highest optimisation level
- C++ has inline keyword
- Fortran has internal functions (possibly inlined)
- C preprocessor macros can be used in simple cases
- Worst case scenario — you can always manually inline code
- Inlining trades speed for code size — unless coding for embedded applications, code size is rarely a problem.

Loop unrolling

Loop overheads: 3 clock cycles per iteration

- Increment index `i=i+1`
- Test `i<n`
- Branch `if` *false* `then exit`

Consider axpy

`z(i)=a*x(i)+y(i)`

3 load/stores, 1 fused add-multiply: Loop overheads dominate!

Unrolling (depth 4)

```
do i=1,n,4
  z(i)=a*x(i)+y(i)
  z(i+1)=a*x(i+1)+y(i+1)
  z(i+2)=a*x(i+2)+y(i+2)
  z(i+3)=a*x(i+3)+y(i+3)
  enddo
```

- 12 load/stores, 4 fused add-multiplies, 3 cycles of loop overhead. Loop overhead no longer dominates!
- But need 13 registers instead of 4. Unrolling too much leads to *register spill*.
- Unrolling typically performed at -O3.

Temporary Copies

Consider a 5 point stencil

$$\Delta x_{ij} = \kappa(x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1} - 4X_{ij})$$

```
delx=kappa*(eoshift(x,shift=-1,dim=1)+...-4*x)
```

This creates 4 temporary arrays to hold shifted data. Lots of copying!

Temporary copies...

Instead:

```
forall (i=2:n-1,j=2:n-1)
  delx(i,j)=kappa(i,j)*(x(i-1,j)+...-4*x(i,j))
```

- A clever compiler may be able to optimise the eoshift code, but don't bet on it!
- In C++, *expression templates* can help

Conclusions

- Advanced Programming doesn’t always help performance

Conclusions

- Advanced Programming doesn't always help performance
- but, usually helps code readability

Conclusions

- Advanced Programming doesn't always help performance
- but, usually helps code readability
- 10% of code consume 90% of CPU time

Conclusions

- Advanced Programming doesn't always help performance
- but, usually helps code readability
- 10% of code consume 90% of CPU time
- *Premature optimisation is the root of all evil*
  Donald Knuth