1

Optimización serie

Objectives

2

Metodología de optimización código serie:

0.– Algoritmica

1.– Obtener los resultados correctos

2.– Emplear código ya optimizado: librerías

3.– Averiguar dónde hay que optimizar

4.– Dejar al compilador hacer su trabajo

5.– Utilizar la cache optimamente

6.– Algunos trucos

Algoritmica

* Elegir el método más eficiente: el de menor orden de complejidad

$$O(1) < O(\log_2(n)) < O(n) < O(n*\log_2(n)) < O(n^2) < \ldots$$

* Vale más la pena pensar un minuto que compilar y ejecutar 10 veces (y además somos más sociales...)

* nos ahorraremos mucho tiempo y esfuerzo con un buen algoritmo

* si pretendemos paralelizar, ahora es un buen momento para pensar un método que nos lo permita.

O(1)

```
Total = Sum1 + Sum2
```

O(n)

```
Do I=1,N
    Total = Total + Sum(I)
End Do
```

O(n²)

```
Do I=1,N
   Do J=1,N
      Total = Total + Sum(I,J)
   End Do
End Do
```

Ejemplo

5

Calcular el orden de complejidad del código siguiente:

```
Do I=1,N
   Do J=I,N
      Total = Total + Sum(I,J)
   End Do
End Do
```


Ejemplo

6

```
Do I=1,N
   Do J=I,N
      Total = Total + Sum(I,J)
   End Do
End Do
```

Cuando I=1, J=1,2,..N => N iteraciones
Cuando I=2, J=2,3,..N => N–1 iteraciones
Cuando I=3, J=3,4,..N => N–2 iteraciones
...
Cuando I=N–2, J=N–2,N–1,N => 3 iteraciones
Cuando I=N–1, J=N–1,N => 2 iteraciones
Cuando I=N, J=N => 1 iteracion

Número total de iteraciones: $= \Sigma\, i \;\; (i = [1..N])$

$$O(N^2 / 2)$$

$\text{Total} = 1 + 2 + 3 + \dots + (N-2) + (N-1) + N$
$\text{Total} = (N+1) + ((N-1) + 2) + ((N-2) + 3) + \dots$
$\text{Total} = (N+1) + (N+1) + (N+1) + \dots$
$\text{Total} = (N+1) * N/2$
$\text{Total} = (N^2 + N) / 2$

$\text{Total} = N^2 / 2 + N / 2$

1.– Obtener los resultados correctos

7

Temas de Porting:

- 64–bit operating system
  - –32, –n32, –64
  - –mips2, –mips3, –mips4
- librerias de otros fabricantes
- Standards violations
  - –static –trapuv (0xFFFA5A5A)
- diferencias de Roundoff (redondeo)
- Precisión de los MADD
- Errores de programación

2.– Emplear código ya optimizado: librerías

8

- libfastm
  - sin, cos, tan, exp, log, pow
  - `f77|cc -o program <objects> -lfastm [-lm]`

`-> man libfastm(3M)`

- Vector instrinsics
  - acos, atan, asin, sin, cos, tan, exp, log, sqrt
  - `cc -o program <objects> -lm`
- Silicon complib libraries
- SiliconCray SCSL libraries
- Silicon MPT libraries
- Otras librerias

Librerías Complib

9

– complib: librerias matemáticas. Incluyen:

SGIMATH
- BLAS
- EISPACK
- LAPACK
- LINPACK
- FFT
- SCILIB

SLATEC

– version serie y paralela, optimizada para diferentes ABI / ISA's

SGIMATH

10

– BLAS level 1, 2 y 3 (Basic Linear Algebra Subprograms)

– FFT (Fast Fourier Transforms) 1D, 2D y 3D

– LAPACK (linear numerical algebra)

– LINPACK (linear equations& linear least squares)

– EISPACK (eigenvectors & eigenvalues 9 dif types of matrices)

– SOLVERS: pcg, direct, symetric & special iterative solvers

– SCIPORT : portable FORTRAN emulation
(by M.J. McBride and S.H. Lamson)
of CRAY SCILIB library

SGIMATH – BLAS

11

The **BLAS** library, level 1, 2 and 3 and machine constants.

Blas1 Library

* vector–vector (unoptimized)

* prefixes

"s"	single precision
"d"	double precision
"c"	single precision complex
"z"	double precision complex

Blas1

12

Blas1 Functions

* DOT		
	(ddot,sdot,zdotu,cdotu)	dot = transpose(X).Y
	(zdotc,cdotc)	dot = transpose(conjg(X)).Y
* AXPY		
	(daxpy,saxpy,caxpy,zaxpy)	Y = Y + a.X
* ROT		
	(drotg,srotg)	calc cos and sin of rotation using vectors
	(drot,srot)	apply rotation to vectors
* COPY		
	(dcopy,scopy,ccopy,zcopy)	Y = X
* SWAP		
	(dswap,sswap,cswap,zswap)	tmp = X; X = Y; Y = tmp
* NRM2 (Euclidean norm)		
	(dnrm2,snrm2)	nrm2 = sqrt(sum(X(i)**2))
	(dznrm2,scnrm2)	nrm2 = sqrt(sum(conjg(X(i))*X(i)))
* ASUM (L1 norm)		
	(dasum,sasum)	asum = sum(abs(X(i)))
	(dzasum,scasum)	asum = sum(abs(real(X(i))) + abs(imag(X(i))))
* SCAL		
	(dscal,sscal,cscal,zscal)	X = a.X
	(zdscal,csscal)	X = cmplx(a).X
* AMAX		
	(idamax,isamax)	imax = max(X(i))
	(izamax,icamax)	imax = max(abs(real(X(i))) + abs(imag(X(i))))

Blas2

1 3

* vector–matrix (optimized)

* prefixes

"s"	single precision
"d"	double precision
"c"	single precision complex
"z"	double precision complex

* suffixes

"mv"	matrix–vector multiply
"r"	rank one update to a matrix
"r2"	rank two update to a matrix
"sv"	solve triangular matrix

* matrix formats

"ge"	m x n 'general'
"gb"	'general' banded
"sy"	symmetric
"sp"	symmetric packed
"sb"	symmetric banded
"he"	hermitian
"hp"	hermitian packed
"hb"	hermitian banded
"tr"	U/L triangular
"tp"	U/L triangular packed
"tb"	U/L triangular banded

Blas2

1 4

* Matrix–Vector

(dgemv,sgemv,zgemv,cgemv)		y = alpha.A.x + beta.y
(dgbmv,sgbmv,zgbmv,cgbmv)	or	y = alpha.A'.x + beta.y
	or	y = alpha.conjg(A').x + beta.y
(dspmv,sspmv,zhpmv,chpmv)		y = alpha.A.x + beta.y
(dsymv,ssymv,zhemv,chemv)		
(dsbmv,ssbmv,zhbmv,chbmv)		
(dtrmv,strmv,ztrmv,ctrmv)		y = A.x
(dtbmv,stbmv,ztbmv,ctbmv)	or	y = A'.x
(dtpmv,stpmv,ztpmv,ctpmv)	or	y = conjg(A').x

* Rank 1 Update

(dger,sger,zgeru,cgeru)	A = alpha.x.y' + A
(zgerc,cgerc)	A = alpha.x.conjg(y') + A
(dsyr,ssyr,dspr,sspr)	A = alpha.x.x' + A
(zhpr,chpr,zher,cher)	A = alpha.x.conjg(x') + A

* Rank 2 Update

(dsyr2,ssyr2,dspr2,sspr2)	A = alpha.x.y' + alpha.y.x' + A
(zher2,cher2,zhpr2,chpr2)	A = alpha.x.conjg(y') + conjg(alpha).y.conjg(x') + A

* Solve

(dtrsv,strsv,ztrsv,ctrsv)		A.x = b
(dtbsv,stbsv,ztbsv,ctbsv)	or	A'.x = b
(dtpsv,dtpsv,ztpsv,ctpsv)	or	conjg(A').x = b

Blas3

15

* matrix–matrix (optimized–parallelized)

* prefixes

"s"	single precision
"d"	double precision
"c"	single precision complex
"z"	double precision complex

* suffixes

"mm"	matrix–vector multiply
"rk"	rank one update to a matrix
"r2k"	rank two update to a matrix
"sm"	solve triangular matrix

Blas3

16

* Matrix–Multiply

(dgemm,sgemm,zgemm,cgemm)		C = alpha.A.B + beta.C (A=A or A' or conjg(A'))
(dsymm,ssymm,zsymm,csymm)		C = alpha.A.B + beta.C
(zhermm,chermm)	or	C = alpha.B.A + beta.C
(dtrmm,strmm,ztrmm,ctrmm)		B = alpha.A.B
	or	B = alpha.B.A
		(A=A or A' or conjg(A')) and (B=B or B' or conjg(B'))

* Rank 1 Update

(dsyrk,ssyrk,zsyrk,csyrk)		C = alpha.A.A' + beta.C
	or	C = alpha.A'.A + beta.C
(zherk,cherk)		C = alpha.A.conjg(A') + beta.C
	or	C = alpha.conjg(A').A + beta.C

* Rank 2 Update

(dsyr2k,ssyr2k,zsyr2k,csyr2k)		C = alpha.A.B' + alpha.B.A' + beta.C
	or	C = alpha.A'.B + alpha.B'.A + beta.C
(zher2k,cher2k)		C = alpha.A.conjg(B') + conjg(alpha).B.conjg(A') + beta.C
	or	C = alpha.conjg(A').B + conjg(alpha).conjg(B').A + beta.C

* Solve

(dtrsm,strsm,ztrsm,ctrsm)		A.X = alpha.B
	or	A'.X = alpha.B
		(A=A or A' or conjg(A'))

SGIMATH

The EISPACK library, a collection of Fortran subroutines that compute the eigenvalues and eigenvectors of nine classes of matrices. The package can determine the eigensystems of complex general, complex Hermitian, real general, real symmetric, real symmetric band, real symmetric tridiagonal, special real tridiagonal, generalized real, and generalized real symmetric matrices. In addition, there are two routines which use the singular value decomposition to solve certain least squares problems.

The LAPACK library, for the most common problems in numerical linear algebra: linear equations, linear least squares problems, eigenvalue problems, and singular value problems. It has been designed to be efficient on a wide range of modern high–performance computers.

SGIMATH

The LINPACK library, for linear equations and linear least squares problems, linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least squares problems.

The SCILIB library, a portable FORTRAN emulation (by M.J. McBride and S.H. Lamson) of CRAY SCILIB, a library of scientific applications subprograms developed by Cray Research, Inc.

Albert Trill v2.0
Copyright: SiliconGraphics
7.–Optimización serie
Supercomputing Team

SGIMATH INFO

19

– para información hacer:

man sgimath –> listado de las rutinas

y para cada una:

man <rutina> –> información específica de la rutina

–otros:

man LIBCONV
man blas
man LAPACK
man solvers – man PSLDLT
man FFT
man conv (convolutions)

SGIMATH LAPACK & LINPACK INFO

20

Documentation for LAPACK and LINPACK is available by writing:

SIAM Department BKLP93
P.O. Box 7260
Philadelphia, Pennsylvania 19101

Anderson E., et. al. SIAM 1992 "LAPACK Users Guide", $19.50
Dongarra J., et. al. SIAM 1979 "LINPACK Users Guide", $19.50

SGIMATH Link

21

utilizar el flag "–l" con:

–lcomplib.sgimath

–lcomplib.slatec

y para multiprocesador paralelo:

–lcomplib.sgimath_mp

–lcomplib.slatec_mp

SLATEC

22

SLATEC is the acronym for the Sandia, Los Alamos, Air Force Weapons Laboratory Technical Exchange Committee.

on–line documentation: /usr/sbin/SLADOC

Incluye 1400 rutinas de public domain distribuido por:

Energy Science and Technology Software Center
P.O. Box 1020
Oak Ridge, TN 37831

Telephone 615–576–2606
E–mail estsc%a1.adonis.mrouter@zeus.osti.gov

SLATEC

23

- error analysis
- elementary and special functions: FNLIB, FUNPACK, AMOSLIB
- elementary vector operations: BLAS
- solutions of systems of linear equations: LINPACK
- eigenanalysis: EISPACK
- QR, SVD and LLS: LINPACK
- interpolation: PCHIP
- optimizations and solution of nonlinear equations: MINPACK
- quadrature: QUADPACK
- ordinary differential equations and partial differential eq.: FISHPACK
- fast Fourier transforms: FFTPACK
- approximations: pseudorandom number generation
- sorting; machine constants, diagnostics, and error handling

Librerias SCSL

24

SGI Cray Scientific Library (SCSL)

Es la nueva librería matemática fruto de la reunión de la SCILIB de Cray y la COMPLIB de SGI.

Está a partie de Irix 6.4 y está optimizada para Origin/Octane. De momento la complib se mantiene, pero tenderá a desaparecer en un futuro próximo.

Para más info: man INTRO_LIBSCSL

Librerias SCSL

25

* Signal processing routines (man INTRO_FFT(3S))
  - Fast Fourier Transform (FFT) routines
* Vector–vector linear algebra subprograms (man INTRO_BLAS1(3S))
  - Level 1 Basic Linear Algebra Subprograms (Level 1 BLAS)
* Matrix–vector linear algebra subprograms (man INTRO_BLAS2(3S))
  - Level 2 Basic Linear Algebra Subprograms (Level 2 BLAS)
* Matrix–matrix linear algebra subprograms (man INTRO_BLAS3(3S))
  - Level 3 Basic Linear Algebra Subprograms (Level 3 BLAS)
* LAPACK routines

Librerias SCSL

26

Para linkar:

–lscs

y para la versión paralela:

–lscs_mp

El API para las FFT es el de Cray, no el anterior de las complib.

Hay más rutinas BLAS (de la SCLIB) de las que había en la Complib

http://hpc.cray.com/products/software/developer/scsl.html

MPT

27

– Message Passing Toolkit (versión actual 1.1)

- MPI 3.0
- XMPI 2.1
- PVM 1.2 (3.3.9)
- XPVM 1.1.1
- SMA 3.0

–SHMEM: (SMA 3.0) Shared Mem lib de Cray portada a IRIX 6.4 sólo para 1 proc.

MPT

28

MPI 3.0

- Optimized for Origin NUMA archtitecture
- Shared memory and/or Sockets
- Shared Memory, Parallel HIPPI bypass, Serial HIPPI bypass
- Shared memory, HIPPI bypass and Sockets
- Support for N64(mips3, mips4) and N32(mips3, mips4) DSO versions, no support for O32 version
- Support new visualization tool called XMPI from public domain and discontinue support for MPIview product
- Will be released as part of Message Passing Toolkit (MPT)

PVM 1.2

- PVM released from merged source base of SGI/CRAY
- Fix All outstanding Bugs/RFE's on PVM 1.1
- XPVM will continue to be supported
- Will be released as part of Message Passing Toolkit (MPT)

Otras librerias

29

- NAG Mark 16 : disponibles para IRIX 6.x

para mas info ver: http://www.nag.co.uk

- Sparse solvers: neuvas rutinas disponibles (consultar)
  - direct solvers for sparse symmetric systems
  - direct solvers for sparse unsimmetric with nearly sym. struct.
  - parallel iterative solver for positive definite systems using conjugate gradient or conjugate residual methods
  - unsimmetric systems using BiCGSTAB or conjugate gradient square methods

Librerias shared – non_shared

30

Shared: Dynamic Shared Objects, DSO's (*.so)

- comparten el segmento de código (no datos) de todos los usuarios
- ahorran memoria (1 sóla copia)
- algo más lentas de ejecución puesto que se crea un nuevo segmento de datos al iniciarse el programa

Non_shared: static objects (*.a)

- más rapidas de ejcución
- mayor ocupación de memoria
- no todas las librerías existen en este formato
- no se instalan por defecto

3.– Averiguar dónde hay que optimizar

31

– hacer un profile (procedure level & line level [–heavy] if necesary)

PC sampling

BasicBlock counting

Counting experiments

– empezar por los módulos que más consumen

–optimizar manualmente sólo donde se pueden obtener beneficios significativos

4.– Dejar al compilador hacer su trabajo

32

- El SWP no se activa por defecto: usar –O3
- Vectorizable loops buenos candidatos para SWP
- SWP puede ser más lento para pocas iteraciones
- la performance puede estar reñida con la precisión de los cálculos
- algunos algoritmos de optimización (SWP p.e.) son heurísticos:
  - puede aumentar mucho el tiempo de compilación
  - los schedules no son únicos
  - pequeñas diferencias pueden modificar significativamente los resultados
- el tener el codigo troceado permite aplicar diferentes optimizaciones via flags de compilación. Si ponemos todo el código en un sólo fichero esto no es posible.

Optimización de bucles

3 3

Aqui hay una multiplicación de matrices (triple nested loop).

```
do i=1,m
   do j=1,n
      do k=1,p
         c(i,j)=c(i,j)+a(i,k)* b(k,j)
      end do
   end do
end do
```

Intentemos optimizarlo

loop optimization

3 4

```
do i=1,m
   do j=1,n
      do k=1,p
         c(i,j)=c(i,j)+a(i,k)* b(k,j)
      end do
   end do
end do
```

Loop Constant

Loop Reversal

```
do j=1,n
   do k=1,p
      t=b(k,j)
      do i=1,m
         c(i,j)=c(i,j)+a(i,k)*t
      end do
   end do
end do
```

loop optimization

35

```
  do j=1,n
     do k=1,p
        t=b(k,j)
        do i=1,m
           c(i,j)=c(i,j)+a(i,k)*t
        end do
     end do
  end do
```

Inner Loop Unrolling

Cleanup Loop

```
do j=1,n
   do k=1,p
      t=b(k,j)
      do i=1,(m-4)+1,4
         c(i+0,j)=c(i+0,j)+a(i+0,k)*t
         c(i+1,j)=c(i+1,j)+a(i+1,k)*t
         c(i+2,j)=c(i+2,j)+a(i+2,k)*t
         c(i+3,j)=c(i+3,j)+a(i+3,k)*t
      end do
      do i=i,m
         c(i,j)=c(i,j)+a(i,k)*t
      end do
   end do
end do
```

Reduce el sobrecoste del loop y a veces mejora la reutilización de datos

loop optimization

36

```
  do j=1,n
     do k=1,p
        t=b(k,j)
        do i=1,m
           c(i,j)=c(i,j)+a(i,k)*t
        end do
     end do
  end do
```

Middle Loop Unrolling

Cleanup Loop

```
do j=1,n
   do k=1,(p-4)+1,4
      t0=b(k+0,j)
      t1=b(k+1,j)
      t2=b(k+2,j)
      t3=b(k+3,j)
      do i=1,m
         c(i,j)=c(i,j)+a(i,k+0)*t0
$                     +a(i,k+1)*t1
$                     +a(i,k+2)*t2
$                     +a(i,k+3)*t3
      end do
   end do
   do k=k,p
      t0=b(k,j)
      do i=1,m
         c(i,j)=c(i,j)+a(i,k)*t0
      end do
   end do
end do
```

La mejora es debida a la reducción del número de "LOAD" "STORE" en c(i,j), y a la mejor localidad secuencial de referencias de b(k,j)

Albert Trill v2.0
Copyright: SiliconGraphics
7.–Optimización serie

Supercomputing Team

loop optimization

37

```
do j=1,n
  do k=1,p
     t=b(k,j)
     do i=1,m
        c(i,j)=c(i,j)+a(i,k)*t
     end do
  end do
end do
```

Outer Loop Unrolling

```
do j=1,(n-4)+1,4
   do k=1,p
      t0=b(k,j+0)
      t1=b(k,j+1)
      t2=b(k,j+2)
      t3=b(k,j+3)
      do i=1,m
         c(i,j+0)=c(i,j+0)+a(i,k)*t0
         c(i,j+1)=c(i,j+1)+a(i,k)*t1
         c(i,j+2)=c(i,j+2)+a(i,k)*t2
         c(i,j+3)=c(i,j+3)+a(i,k)*t3
      end do
   end do
end do
```

Cleanup Loop

```
do j=j,n
   do k=1,p
      t0=b(k,j)
      do i=1,m
         c(i,j)=c(i,j)+a(i,k)*t0
      end do
   end do
end do
```

La mejora radica en el mejor uso de a(i,k) en el bucle

loop optimization

38

```
do j=1,n
  do k=1,p
     t=b(k,j)
     do i=1,m
        c(i,j)=c(i,j)+a(i,k)*t
     end do
  end do
end do
```

Loop Blocking

a(i,k) es reusado por cada valor de "j"

```
do j1=1,n,jblk
   j2=j1+jblk-1
   if (j2.gt.n) j2=n
   do k1=1,p,kblk
      k2=k1+kblk-1
      if (k2.gt.p) k2=p
      do i1=1,m,iblk
         i2=i1+iblk-1
         if (i2.gt.m) i2=m
         do j=j1,j2
            do k=k1,k2
               t=b(k,j)
               do i=i1,i2
                  c(i,j)=c(i,j)+a(i,k)*t
               end do
            end do
         end do
      end do
   end do
end do
```

código Original

Esta modificación implementa localidad espacial de referencia (elimina los fallos de cache).

loop optimization

39

Tiempo de ejecución de las siguientes versiones de multiplicación de matrices:

- original code
- loop reversal and loop constant
- loop reversal and inner loop unrolling
- loop reversal and middle loop unrolling
- loop reversal and outer loop unrolling
- loop reverval and middle and outer loop unrolling
- loop reversal and loop blocking
- loop reversal and middle and outer unrolling with blocking
- original code compiled with –O3

En un Power Challenge R8000/75MHz:

87.0	* original code
59.4	* loop reversal and loop constant
62.1	* loop reversal and inner loop unrolling
44.3	* loop reversal and middle loop unrolling
35.5	* loop reversal and outer loop unrolling
27.4	* loop reverval and middle and outer loop unrolling
36.7	* loop reversal and loop blocking
17.1	* loop reversal and middle and outer unrolling with blocking
20.2	* original code compiled with –O3

otro ejemplo

40

```
do i=1,1000
   do j=1,1000
      a(j,i) = float(i*j)
   end do
end do


do k=1,100
   c = 1.0/float(k)
   do i=1,1000
      do j=1,1000
         b(j,i) = a(j,i) * c
      end do
   end do
end do
```

En un Power Challenge R8000/75MHz:

original:	46.8	
optimized:	27.9	
original:	28.0	(–OPT:IEEE_arithmetic=3,roundoff=3)

Recommended performance flags para CrayOrigin2000

4 1

f77 –n32 –mips4 –Ofast=ip27 –OPT:IEEE_arithmetic=3 –lfastm –lm

–O3
–IPA
–OPT:roundoff=3
–OPT:alias=typed
ip27

–TARG:platform=ip27 (cache size, etc...)

–r10000 (instruction scheduling)

–TARG:proc=r10000 (libraries to link with)

5.–Utilizar la cache optimamente

4 2

- ***locality of reference***
- **correcto acceso a vectores y matrices**
- **técnicas de *blocking***

Localidad de Referencia

* **Espacial**

Tendencia de un proceso a referenciar una porción del espacio virtual de direcciones cercano a la última referencia

* **Temporal**

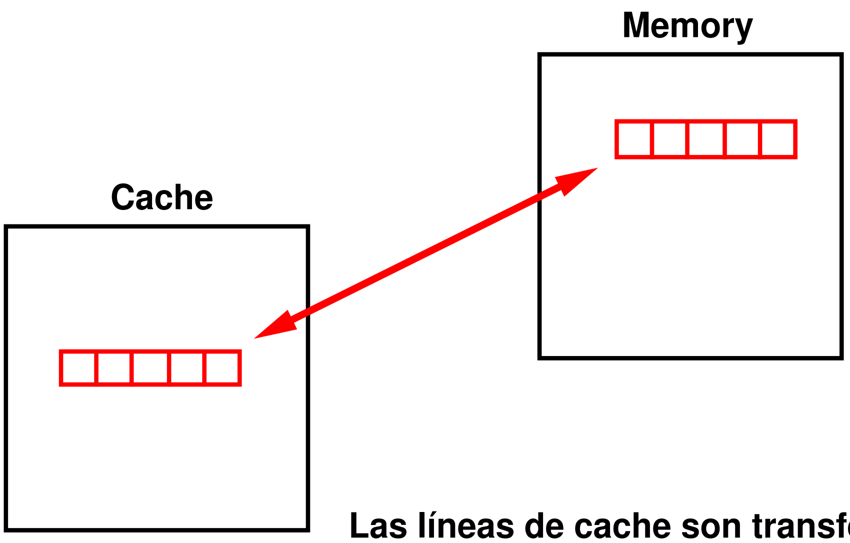
Tendencia de un proceso a referenciar en el futuro elemtos referenciados en un pasado reciente.

Maximizar el uso de la cache a través de:

* **Estructuras adyacentes (Indexing & Pointers?)**
* **Loop Blocking**
* **Loop Reversal**
* **Loop Unrolling**
* **Index Reversal**

Localidad de referencia

45



Las líneas de cache son transferidas en trozos (chuncks) de direcciones de memoria consecutivas (128 Bytes/línea).

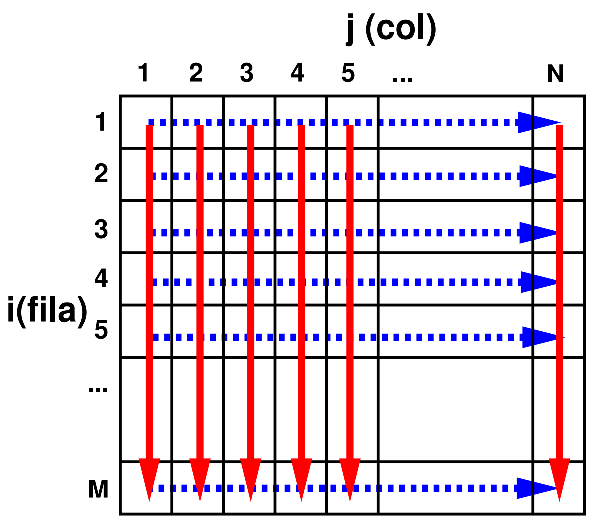
Ejemplo de localidad de referencia en Fortran

46

```
Do i=1,N
    Do j=1,M
        A( i, j ) =  A( i, j ) + B( i , j )
    End Do
End Do
```

A[fila,col]

Fortran –> almac. por columnas
C –> almac. por filas



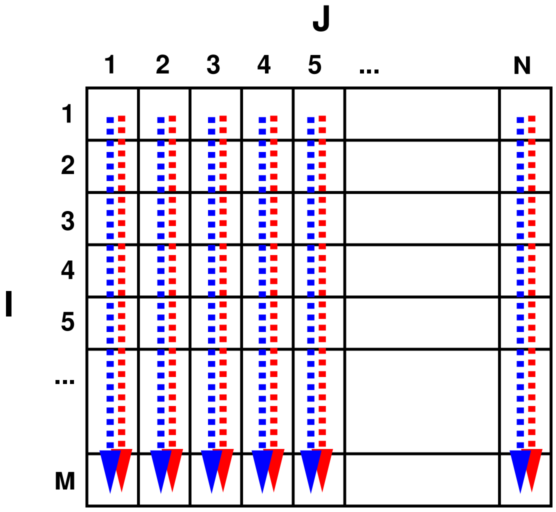
orden de almacenamiento

orden de acceso

Ejemplo de localidad de referencia en Fortran

47

```
Do j=1,M
    Do i=1,N
        A( i, j ) =  A( i, j ) + B( i, j )
    End Do
End Do
```



Localidad espacial de referencia
No hay localidad temporal de referencia !!!

Access order

Storage order

Ejemplo de localidad de referencia en C

48

En C es exactamente al revés:

Mala localidad !

```
for (j=0; j<M; j++)
    for (i=0; i<N; i++)
        C[i][j] = A[i][j] + B[i][j];
```

Loop Reversal

```
for (i=0; i<N; i++)
    for (j=0; j<M; j++)
        C[i][j] = A[i][j] + B[i][j];
```

Index Reversal

```
for (j=0; j<M; j++)
    for (i=0; i<N; i++)
        C[j][i] = A[j][i] + B[j][i];
```

Buena localidad

Localidad de referencia en Fortran/C: resumen

49

Fortran

```
Do j=1,M
   Do i=1,N
      A( i, j ) =  A( i, j ) + B( i, j )
   End Do
End Do
```

Los índices cruzados

C

```
for (i=0; i<N; i++)
      for (j=0; j<M; j++)
            C[i][j] = A[i][j] + B[i][j];
```

Los índices sin cruzar

Ejemplo

50

```
dimension a(300,300,300),b(300,300,300),c(300,300,300)
Do i=1,300
   Do j=1,300
      Do k=1,300
         a(i,j,k) = a(i,j,k) + b(i,j,k) * c(i,j,k)
      End Do
   End Do
End Do
```

1.– Encontrar todas las combinaciones de loop/index reversal
2.– Ejecutar todas las combinaciones.
3.– Usando el caso más lento como base, calcular los respectivos Speedup

En un Power Challenge R8000/75MHz:

i j k	477.0	(S=27.3)
i k j	346.7	(S=19.8)
j i k	146.4	(S=8.4)
j k i	19.3	(S=1.1)
k i j	21.1	(S=1.2)
k j i	17.5	(S=1.0)

Métodos de Indexación

51

Direct (++)

```
Do j=1,M
   Do i=1,N
      a(i,j) = ...
   End Do
End Do
```

Calculated explicit (+)

```
Do j=1,M
   Do i=1,N
      a((i-1)*N+i) = ...
   End Do
End Do
```

Indirect (––)

```
Do j=1,M
   Do i=1,N
      k=k+1
      a(idx1(j)+idx2(i)) = ...
   End Do
End Do
```

Accumulated (–) (Induction/)

```
k=0
Do j=1,M
   Do i=1,N
      k=k+1
      a(k) = ...
   End Do
End Do
```


Blocking

52

Blocking de cache es:

- **Una optimizaciónn que se aplica a data sets que no caben enteros en cache.**
- **Una forma de incrementar la localidad de referencia temporal (reuso de los datos)**
- **Una forma de incrementar la localidad espacial de referencia (explotar las líneas de cache completamente)**

6.– Algunos "trucos"

53

- **Dividir versus Multiplicar**
- **implicit I/O**
- **"factor común"**
- **Inlining para optimizar**

Dividir versus Multiplicar

54

```
subroutine MyRoutine(a,b)
dimension a(1000),b(1000)

do i=1,1000
   a(i) = 0.0
end do

b = 2.0

do i=1,1000
   a(i) = a(i) / b
end do
end
```

```
subroutine MyRoutine(a,b)
dimension a(1000),b(1000)

do i=1,1000
   a(i) = 0.0
end do

b = 1.0 / 2.0

do i=1,1000
   a(i) = a(i) * b
end do
end
```

(Los compilers desde 6.0, con -OPT:IEEE_arithmetic=3,roundoff=2, hacen esta transformación)

Implicit I/O

55

* Add "mmap"
* "read"/"write" desaparecen
* Reduce el Overhead
* todavía se opera sobre el fichero (no es un RAM disk !!!)

```
float f1;
int i,fd;
fd = open("file",O_RDWR,0644);
for (i=1;i!=N;i++) {
        read(fd,&f1,sizeof(f1));
        f1 +=  2.0;
        lseek(fd,-1*sizeof(f1),
               SEEK_CUR);
        write(fd,&f1,sizeof(f1));
}
```

```
float *f1;
int i,fd;
fd = open("file",O_RDWR,0644);
f1 = mmap(0,N*sizeof(*f1),
        PROT_READ|PROT_WRITE,
        MAP_PRIVATE,fd,0);
for (i=1;i!=N;i++)
        f1[i] += 2.0;
```


Factor común

56

```
DO I=1,1000
 DO J=1,3000
       A(I,J) = SIN(PI)* B(I,J) + COS(PI) * B(I,J)
 ENDDO
ENDDO
```

```
SP = SIN(PI)
CP = COS(PI)
DO I=1,1000
 DO J=1,3000
       A(I,J) = SP * B(I,J) + CP * B(I,J)
 ENDDO
ENDDO
```

Inlining


```
REAL*8 A(1000,1000), B(1000,1000), C(1000,1000)
DO I=1,NITER
      CALL CALCULA (A,B,C,ITER)
ENDDO
```

```
REAL*8 A(1000,1000), B(1000,1000), C(1000,1000)
DO I=1,NITER
      DO J=1,1000
         DO K=1,1000
           A(I,J)= B(I,J) * NITER*360.0 + C(I,J)
         ENDDO
      ENDDO
ENDDO
```