NÚCLEOS COMPUTACIONALES Y LIBRERÍAS

Máster de Modelización e Investigación Matemática, Estadística y Computación

Programación científica

Parte III

Curso 2014/2015

Contenido

Contenido

1.1 Introducción

- Una de las primeras aplicaciones de la computación paralela fue la aceleración de todo tipo de algoritmos numéricos, pero especialmente de los algoritmos matriciales numéricos.
- Algunos algoritmos matriciales paralelos pueden considerarse como un auténtico paradigma de programación paralela y su diseño puede esclarecer muchas cuestiones sobre el diseño de algoritmos paralelos.
- El estudio de los algoritmos numéricos paralelos y de las librerías numéricas es especialmente adecuado para un buen conocimiento de la programación paralela.

- La regularidad de las estructuras matriciales las hace especialmente adecuadas para su procesamiento paralelo (particionado, organización, distribución, etc.).
- Los algoritmos matriciales surgen con frecuencia en el campo de la ingeniería y de la ciencia (problemas de gran dimensión o problemas en tiempo real) y deben ser resueltos de manera eficiente.
- Existen numerosas librerías numéricas paralelas especialmente adecuadas/eficaces para la resolución de este tipo de tareas.
- Objetivo: *mantener la estabilidad de los algoritmos desarrollados y disminuir el tiempo de ejecución*.

- Muchos de los problemas numéricos que surgen en ingeniería aparecen al resolver situaciones que están controladas por leyes físicas.
- Su resolución implica la construcción de un modelo matemático previo, que, en muchas ocasiones, suele responder a una ecuación diferencial ordinaria o en derivadas parciales, a un sistema de ecuaciones diferenciales o a un sistema diferencial-algebraico.
- La resolución matemática directa de este tipo de sistemas no suele permitir soluciones prácticas, por lo que resulta necesario buscar una solución numérica.
- Esto requiere algún tipo de discretización que suele desembocar en la resolución de problemas computacionales típicos como la resolución de sistemas de ecuaciones lineales o no lineales, o el cálculo de valores (vectores) propios o singulares.

- En la búsqueda de soluciones de problemas tales como la resolución de sistemas de ecuaciones o el cálculo de valores propios, es necesario utilizar rutinas sencillas que lleven a cabo operaciones elementales.
- Esto ha dado lugar a que las llamadas librerías matriciales se organicen en base a los denominados núcleos computacionales.
- Un paquete de núcleos computacionales está constituido por un conjunto de funciones o subprogramas que resuelven operaciones vectoriales y/o matriciales sencillas (productos escalar, suma de vectores, producto matriz-vector, etc.).
- Una librería sólo contiene rutinas que resuelven problemas más complejos haciendo llamadas a núcleos computacionales cuando se requiere ejecutar operaciones elementales.

- Introduciremos, a continuación, las librerías computacionales más utilizadas y los núcleos computacionales sobre los que se apoyan.
- Nos centraremos en librerías como LAPACK y ScaLAPACK, y en los núcleos computacionales que las sustentan: BLAS, BLACS y PBLAS.
- Existen muchas otras librerías tanto para matrices densas como para matrices dispersas como: SPARSKIT, PETs, PSPASES, SuperLU, PLAPACK, FLAME, etc.

- Conviene destacar otras de reciente creación como StructPack (**http://www.inco2.upv.es/structpack.html**):

  "StructPack was conceived for solving a variety of numerical Linear Algebra problems, such as solving systems of linear equations, least squares problems, eigenvector calculation and singular value decomposition, on different types of matrices such as Toeplitz, Hankel, Vandermonde, or circulant matrices, ... and specific cases as tridiagonal Toeplitz matrices, positive definitive symmetric matrices, etc."

Contenido

1.2 Núcleos computacionales: BLAS

- El núcleo BLAS (*Basic Linear Algebra Subprograms), propuesto en 1973 (BLAS 1) fue ideado en el lenguaje* `Fortran` *77 y cubría únicamente operaciones elementales de complejidad* $O(n)$*, como producto escalar de vectores y suma de vectores con escalado* $y \leftarrow \alpha \cdot x + y$*, siendo* x *e* y *vectores de tamaño* n *y* α *un escalar. (**http://www.netlib.org/blas/**)*

  C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, Basic Linear Algebra Subprograms for FORTRAN usage, ACM Trans. Math. Soft., 5 (1979), pp. 308-323.

- Las ventajas de adoptar este conjunto de subprogramas básico para resolver problemas de álgebra lineal son:
  - Robustez: los subprogramas se diseñan teniendo cuenta la estabilidad numérica de los algorimtos,
  - Portabilidad: el interfaz se diseña de manera que los núcleos puedan ser utilizados por cualquier computador y en un lenguaje tan extendido como el `Fortran`,
  - Legibilidad: el diseño está centrado en hacer visibles las operaciones matemáticas.

- El uso de BLAS 1 no es la mejor forma de obtener una alta eficiencia en códigos de alto nivel, de esta forma surge el BLAS 2 que cubre un conjunto de operaciones de tipo matriz-vector, las cuales aparecen habitualmente en algoritmos matriciales.
- El tipo de operaciones implementadas en BLAS 2 son operaciones de complejidad $O(n^2)$, similares a $y \leftarrow \alpha \cdot A \cdot x + \beta \cdot y$, siendo A una matriz de tamaño $m \times n$, $A \leftarrow \alpha \cdot x \cdot x^T + A$, o $x \leftarrow \alpha \cdot A^{-1} \cdot x$, con A triangular superior.

  J.J. Dongarra, J. Du Croz, S. Hammarling, and R.J. Hanson, *An extended set of FORTRAN Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 14 (1988), pp. 1-17.

- El enfoque basado en optimizar las operaciones tipo matriz-vector no suele dar buenos resultados en computadores que disponen de arquitecturas estructuradas en forma de jerarquía de memorias o en computadores paralelos.
- Para este tipo de arquitecturas es mejor descomponer las matrices por bloques y realizar operaciones de tipo matriz-matriz usando bloques como operadores y optimizando las operaciones de este tipo.
- Esta estrategia permite operar al máximo sobre los bloques cuando estos están en la memoria caché o en la memoria local, evitando movimientos innecesarios de datos.

- Las operaciones se pueden realizar simultáneamente sobre distintos bloques o incluso de dentro de las operaciones de un bloque.
- Estas necesidades se cubren con la aparición de BLAS 3, que tiene una organización por bloques e implementa las operaciones básicas de tipo matriz-matriz tales como $C \leftarrow \alpha \cdot A \cdot B + \beta \cdot C$, $C \leftarrow \alpha \cdot A \cdot A^T + \beta \cdot C$, con A de tamaño $k \times n$, o $B \leftarrow \alpha \cdot T^{-1} \cdot B$, con T triangular.

  J.J. Dongarra, J. Du Croz, I.S. Duff, and S. Hammarling,
  A set of Level 3 Basic Linear Algebra Subprograms,
  ACM Trans. Math. Soft., 16 (1990), pp. 1-17.

- Cada rutina tiene un nombre que especifica la operación, el tipo de matriz involucrada y sus precisiones.
- Algunas de las operaciones más habituales son:
  - DOT: $x^T \cdot y$,
  - AXPY (a x plus y): $\alpha \cdot x + y$,
  - MV (matrix-vector product): $A \cdot x$,
  - SV (matrix-vector solve): $A^{-1} \cdot x$,
  - MM (matrix-matrix product): $A \cdot B$.

- Los tipos de matrices son:
  - GE: general,
  - GB: general tipo banda,
  - SY (SB): simétrica (simétrica banda),
  - HE (HB): hermitiana (hermitiana banda),
  - TR (TB): triangular (triangular banda).

- Cada operación se puede definir con distintas tipos de precisiones:
  - I: entero,
  - S: simple (real),
  - D: doble (real),
  - C: compleja,
  - Z: compleja (doble).
- Así, por ejemplo SAXPY denota "single-precision a x plus y", o SGEMM denota "single-precision general matrix-matrix multiply".

- **Ejemplo**: matrix-matrix multiplication
- SUBROUTINE $C = \alpha \cdot \mathrm{op}(A) \cdot \mathrm{op}(B) + \beta \cdot C$
- SGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
  - S: single precision, GE: general matrix, MM: matrix-matrix
  - TRANSA: the form of $\mathrm{op}(A)$ to be used in the matrix multiplication
    - TRANSA= *N* or *n* $\mathrm{op}(A) = A$, *T* or *t* $\mathrm{op}(A) = A^T$, *C* or *c* $\mathrm{op}(A) = A^H$
  - *M*: the number of rows of $\mathrm{op}(A)$and *C*
  - *N*: the number of columns of $\mathrm{op}(B)$and *C*
  - *K*: the number of columns of $\mathrm{op}(A)$and the number of rows of $\mathrm{op}(B)$

- SUBROUTINE $C = \alpha \cdot \mathrm{op}(A) \cdot \mathrm{op}(B) + \beta \cdot C$
- SGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
  - *ALPHA*, *BETA*: the scalar coefficients *alpha* and *beta*
  - *A*: real array of dimension (LDA, ka), where *ka* is *K* when $TRANSA = N$, and is *M* otherwise
  - *LDA*, *LDB*, *LDC*: the first dimension of *A*, *B* and *C* as declared in the calling program
  - *B*: real array of dimension (LDB, kb), where *kb* is *N* when $TRANSB = N$, and is *K* otherwise
  - *C*: real array of dimension (LDC, n)

1.2.1 Flops versus memoria

- Analizaremos con algo más de detalle la relación entre el uso de BLAS y ciertos parámetros que muestran la eficiencia de la estrategia.

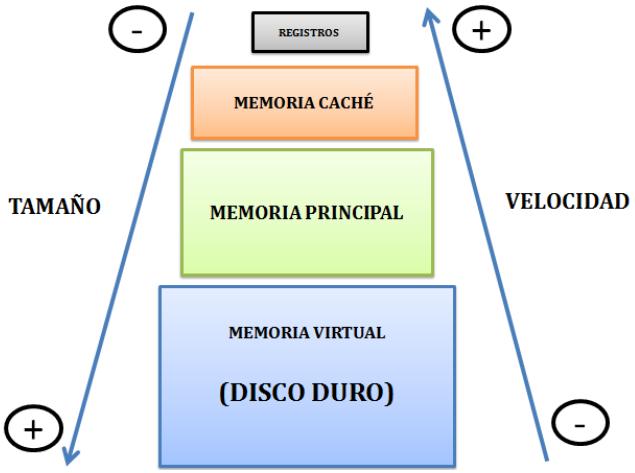

Figura 1: Jerarquía de memoria.

- En la Tabla 1 aparecen las magnitudes asociadas a los parámetros f, m y q para operaciones realizadas en distintos niveles de BLAS:
  - f: número de operaciones en punto flotante,
  - m: número de accesos a la memoria,
  - q: proporción entre el número de flop y el de accesos a memoria. (cantidad de trabajo útil que podemos hacer en comparación con el tiempo de traslado de datos.)

Operation	Definition	f	m	q=f/m
BLAS 1				
`saxpy`	$y \leftarrow \alpha \cdot x + y$	$2n$	$3n+1$	$2/3$
BLAS 2				
`Matrix-vector mult`	$y \leftarrow A \cdot x + y$	$2n^2$	n^2+3n	2
BLAS 3				
`Matrix-matriz mult`	$C \leftarrow A \cdot B + C$	$2n^3$	$4n^2$	$n/2$

Tabla 1: Flops versus memoria.

- Así, por ejemplo, el número de operaciones elementales para calcular $y_i \leftarrow \alpha \cdot x_i + y_i$, con $i = 1, 2, \ldots, n$, es $2n$, ya que en cada paso hacemos 1 suma y 1 producto.
- El número de accesos a memoria (m) necesarios para implementar `saxpy` es $3n+1$,
  - leer n valores de x_i, n valores de y_i y un valor para α desde la memoria "lenta" a la memoria "rápida",
  - escribir a continuación n valores de y_i.
- Finalmente, la proporción entre las magnitudes anteriores, q, resulta ser $2/3$.

- Si denotamos como t_{arith} y t_{mem} al tiempo necesario para realizar una operación en punto flotante y para acceder a la memoria del computador respectivamente, el tiempo de ejecución del algoritmo (siempre que el cálculo y la comunicación no se realicen simultáneamente), puede ser expresado como

$$f \cdot t_{arith} + m \cdot t_{mem} = f \cdot t_{arith} \left(1 + \frac{m}{f} \frac{t_{mem}}{t_{arith}} \right) = f \cdot t_{arith} \left(1 + \frac{1}{q} \frac{t_{mem}}{t_{arith}} \right).$$

- De donde resulta evidente que cuanto mayor sea el valor de q menor será el tiempo de ejecución, situación que se alcanza construyendo algoritmos por bloques y utilizando operaciones de nivel 3 de BLAS.

Contenido

1.3 Librería LAPACK

- LAPACK (*Linear Algebra PACKage*) es una librería escrita inicialmente en `Fortran` 90, aunque puede invocarse también desde programas en lenguaje `C`. (**http://www.netlib.org/lapack/**)
- Está compuesta por subrutinas para resolver: sistemas de ecuaciones, problemas de mínimos cuadrados lineales, problemas de valores propios e de valores singulares. Las factorizaciones matriciales asociadas, *LU*, Cholesky, *QR*, *SVD*, Schur y Schur generalizada, también se proporcionan.
- Es una librería que trabaja con matrices densas y matrices banda (reales y complejas con simple y doble precisión), pero no con matrices dispersas

- Inicialmente surgió como una fusión mejorada de las librerías LINPACK y EISPACK (librerías estándar para la resolución de sistemas de ecuaciones lineales y el cálculo de valores propios).
- Estas librería quedaron obsoletas con la aparición de los computadores vectoriales y paralelos con memoria compartida.
- El diseño de LAPACK se ha planteado considerando una reorganización de los algoritmos a partir de la utilización de operaciones por bloques, especialmente operaciones tipo matriz-matriz.
- Esto le ha permitido ser especialmente eficaz sobre computadoras con jerarquía de memoria, y más aún al permitir una optimización en función de la arquitectura del computador, consiguiendo así portabilidad y eficiencia en un amplio rango de computadores actuales.

- Las rutinas de LAPACK están escritas de forman que utilizan al máximo llamadas a los núcleos computaciones BLAS, fundamentalmente BLAS 3.
- LAPACK contiene tres tipos de rutinas:
  - Rutinas driver: resuelven un problema completo (por ejemplo, el cálculo de los valores propios de una matriz simétrica).
  - Rutinas computacionales: resuelven diferentes tareas computacionales (por ejemplo, factorización QR, LDL^T).
  - Rutinas auxiliares: realizan subtareas de los algoritmos orientados a bloques, puede considerarse como una extensión de BLAS.

- **Ejemplo**: Solve a linear system with a general non-symmetric matrix
- SUBROUTINE $A \cdot x = b$
- SGESV(N,NRHS,A,LDA,IPIV,B,LDB,INFO)
  - S: single precision, GE: general matrix, SV: solve linear system
  - N [in]: The number of linear equations, i.e., the order of the matrix A
  - NRHS [in]: The number of right hand sides, i.e., the number of columns of the matrix B

- SUBROUTINE $A \cdot x = b$
- SGESV(N,NRHS,A,LDA,IPIV,B,LDB,INFO)
  - A(LDA,N) [in, out]: On entry, the $N \times N$ coefficient matrix A. On exit, the factors L and U from the factorization $A = P \cdot L \cdot U$
  - LDA [in]: The leading dimension of the array A
  - IPIV(N) [out]: The pivot indices that define the permutation matrix P
  - B(LDB,RHS) [in, out]: On entry, the right-hand-side matrix B. On exit, if $INFO = 0$, the $N \times NRHS$ solution matrix X
  - LDB [in]: The leading dimension of the array B
  - INFO [out]:
    - $INFO = 0$: successful exit.
    - $INFO = -i$: unsuccessful exit, the $i-$th argument had an illegal value.
    - $INFO = i$: unsuccessful exit, $U(i,i)$ is exactly zero.

Contenido

1.4 Librería ScaLAPACK

- ScaLAPACK (*Scalable Linear Algebra PACKage*) surge en la década de los 90 como una continuación natural de LAPACK para computadores paralelos con memoria distribuida.
(**http://www.netlib.org/scalapack/**)
- Está compuesta por un conjunto de subrutinas de altas prestaciones que resuelven problema de álgebra lineal numérica, fundamentalmente sistemas de ecuaciones lineales, problemas de mínimos cuadrados y problemas de valores propios, siguiendo el modelo de paso de mensajes.
- Este conjunto de subrutinas es un subconjunto de las rutinas que constituyen el LAPACK.

- Los objetivos de ScaLAPACK son los mismos que los de LAPACK, esto es:
  - eficiencia (to run as fast as possible),
  - escalabilidad (as the problem size and number of processors grow),
  - exactitud (including error bounds),
  - portabilidad (across all important parallel machines),
  - flexibilidad (so users can construct new routines from well-designed parts),
  - fácil de usar (by making the interface to LAPACK and ScaLAPACK look as similar as possible).

- ScaLAPACK está escrito siguiendo el modelo SPMD (*Single Program Multiple Data*) y utiliza paso de mensajes explícito para la comunicación entre procesadores.
- Trabaja sobre matrices que están disribuidas entre los procesadores mediante una distribución cíclica por bloques.
- ScaLAPACK trabaja con matrices densas y banda, pero no con matrices dispersas.
- Los tipos numéricos considerados son reales o complejos con simple o doble precisión.

- Está basado en la utilización de núcleos computacionales eficientes de bajo nivel, para lo que fue necesario extender las funcionalidades de BLAS al caso de computadores con memoria distribuida.
- Así se generó PBLAS (*Parallel Basic Linear Algebra*), que es la extensión de BLAS siguiendo el modelo de paso de mensaje.
- Tiene una estructura e interfaz muy similar a BLAS y permite simplificar el desarrollo de software numérico, portable y fiable, sobre máquinas con memoria distribuida.
- Al igual que BLAS también tiene tres niveles (PBLAS 1, PBLAS 2 y PBLAS 3), en función de los algoritmos implementados.

- Para poder diseñar las rutinas de PBLAS fue necesario diseñar un paquete de núcleos computacionales encargados sólo de las comunicaciones y que se denomina BLACS (*Basic Linear Algebra Communications Subroutines*).
- BLACS implementa un conjunto de operaciones básicas de comunicación útiles en álgebra lineal numérica.
- Al estar dedicada a operaciones especificas son rutinas más simples y más eficientes que un paquete de comunicaciones de carácter general.
- El código de comunicaciones está ligado a la máquina física a través del modelo de paso de mensajes y no es necesario cambiarlo al cambiar de plataforma, siendo este uno de sus puntos fuertes.

- Teniendo como objetivo minimizar los movimientos de datos entre los diferentes niveles de jerarquia de memoria, las rutinas de ScaLAPACK se basan en algoritmos organizados sobre matrices y vectores particionados por bloques.
- Al igual que en LAPACK, la librería ScaLAPACK también tiene *rutinas driver*, *computacionales* y *auxiliares*.
- ScaLAPACK está organizado en forma de software jerarquizado que consta de componentes locales y globales. Los componentes locales se llaman desde un único procesador (los datos están en su memoria), mientras que las componentes globales son rutinas paralelas (síncronas) cuyos argumentos utilizan datos de varios procesadores.

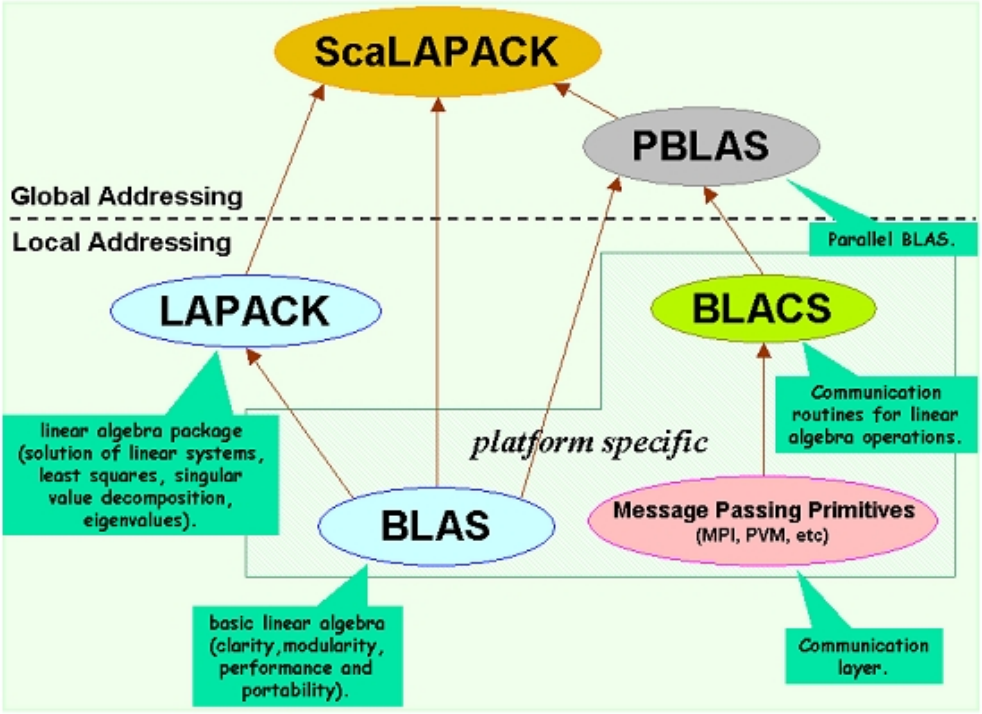

Figura 2: Jerarquía de ScaLAPACK.

Contenido

1.5 Caso de estudio

- We study how to optimize matrix-multiplication ($C = A \cdot B + C$) to minimize the number of memory moves and so optimize its performance.

  J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM (1997)

- We assume that the matrices are stored columnwise, as in Fortran, and we assume that are two levels of memory hierarchy, fast and slow, where the slow memory is large enough to contain three $n \times n$ matrices, A, B and C, but the fast memory contains only M words where $2n < M \ll n^2$.

- This means that the fast memory is large enough to hold two matrix columns or rows but not a whole matrix.

- The simple matrix multiplication algorithm that one might try consist of three nested loops, which we have annotated to indicate the data movement.

Algorithm 1 Unblocked matrix multiplication

```
1:  for i = 1 : n do
2:      {Read row i of A into fast memory}
3:      for j = 1 : n do
4:          {Read C_ij into fast memory}
5:          {Read column j of B into fast memory}
6:          for k = 1 : n do
7:              C_ij = C_ij + A_ik · B_kj
8:          end for
9:          {Write C_ij back to slow memory}
10:     end for
11: end for
```

- The innermost loop is doing a dot product of row i of A and column j of B to compute C_{ij}.
- The two innermost loops (on j and k) as doing a vector-matriz multiplication os the i-th row of A times the matrix B to get i-th row of C.
- This is a hint that we will not perform any better that these BLAS 1 and BLAS 2 operations, since they are within the innermost loops.

- Here is the detailed count of memory references:
  - n^3 for reading B n times (once for each value of i);
  - n^2 for reading A one row at a time and keeping it in fast memory until it is no longer needed;
  - $2n^2$ for reading one entry of C at a time and keeping it in fast memory until it is completely computed, and then moving it back to slow memory.
- This comes to $n^3 + 3n^2$ memory moves.
- Thus $q = (2n^3)/(n^3 + 3n^2) \approx 2$, which is no better than Level 2 BLAS and far from the maximum posible $n/2$.

- If $M \ll n$, so that we cannot keep a full row od A fast memory, q further decrease to 1, since the algorithm reduces to a sequence of inner products, which are Level 1 BLAS.
- For every permutation of three loops on i, j, and k, one gets another algorithm with q about the same.
- It is better to use algorithm blocking, where C is broken into an $N \times N$ block matrices with $n/N \times n/N$ blocks C^{ij}, and A and B are similarly partitioned.

Algorithm 2 Blocked matrix multiplication

```
1: for i = 1 : N do
2:    for j = 1 : N do
3:       {Read C^ij into fast memory}
4:       for k = 1 : N do
5:          {Read A^ik into fast memory}
6:          {Read B^kj into fast memory}
7:          Cij = C^ij + A^ik · B^kj
8:       end for
9:       {Write C^ij back to slow memory}
10:   end for
11: end for
```

- Our memory reference counts is as follows:
  - $2n^2$ for reading and writing each block of C;
  - Nn^2 for reading A N times (reading each $n/N \times n/N$ submatrix A^{ik} N^3 times);
  - Nn^2 for reading B N times (reading each $n/N \times n/N$ submatrix B^{kj} N^3 times).
- This comes to $(N+2)n^2 \approx 2Nn^2$ memory references.
- So we want to choose N as small as possible to minimize the number of memory references.

- N is subject to the constraint $M \geq 3(n/N)^2$, which means that one block each from A, B, and C must fit in fast memory simultaneously.
- This yields $N \approx n\sqrt{3/M}$, and so

$$q \approx \frac{2n^3}{2Nn^2} \approx \sqrt{M/3},$$

  which is much better than the previous algorithm.
- In particular q grows independently of n as M grows, i.e., the algorithm to be fast for any matrix size n and to go faster if the memory size M is increased.

Remarks:

- The previous algorithm is asymptotically optimal, this is, no reorganization of matrix-matrix multiplication (using $2n^3$ arithmetic operations) can have a q larger than $O(\sqrt{M})$.
- This brief analysis ignores a number of practical issues:
  - the optimal blocks size may not be square,
  - the cache memory and register structure of a machine will strongly affect the best shapes of submatrices,
  - there may be special hardware instructions that perform both a multiplication and addition in one cycle, it may also be possible to execute several multiply-add operations simultaneously if they do not interfere.

Contenido

1.6 Bibliografía

- F. Almeida, D. Giménez, J.M. Mantas, A.M. Vidal, *Introducción a la programación paralela*, Paraninfo Cengage Learning, 2008.
- J.W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.
- J. Dongarra, I. Foster, G. Fox, Geoffrey, W. Gropp, K. Kennedy, L. Torczon, A. White (Eds.): *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, 2003.
- A. Grama, G. Karypis, V. Kumar, A. Gupta, *Introduction to Parallel Computing* (2nd ed.), Addison-Wesley, 2003.
- G.H. Golub, C.F. Van Loan, *Matrix computations* (3rd ed.), Johns Hopkins University Press, 1996.
- http://www.netlib.org/blas/
- http://www.netlib.org/lapack/
- http://www.netlib.org/scalapack/