PANORAMICA SULLE LIBRERIE BLAS, LAPACK E ATLAS

1 Introduzione

Gran parte dei problemi del calcolo scientifico ed ingegneristico richiede di risolvere uno o più problemi dell'algebra lineare numerica (ALN) :

- risoluzione di sistemi lineari
- ricerca di autovalori e/o autovettori
- calcolo della SVD (valori e vettori singolari)

Inoltre, la risoluzione di questi problemi è generalmente una percentuale considerevole del costo computazionale totale richiesto per la risoluzione del problema globale; questo costo si traduce spesso in ore o giorni di tempo-macchina impiegato. Dunque, implementare in modo efficiente gli algoritmi che risolvono i problemi dell'algebra lineare numerica è estremamente importante dal punto di vista applicativo ed è anche per questo che trattiamo questo caso più approfonditamente. Gli algoritmi di algebra lineare numerica hanno in comune un insieme relativamente piccolo e stabile di operazioni di base, che svolge la quasi totalità dei calcoli necessari negli algoritmi di ALN. Le operazioni fondamentali in qualsiasi codice numerico sono essenzialmente di tre tipi:

- $w = v_1 \cdot v_2$: prodotto scalare di due vettori (operazioni che lavorano su vettori e producono uno scalare)
- $w = Av$: prodotto di una matrice per un vettore (operazioni tra vettori e matrici o che comunque producono una matrice)
- $B = A_1 A_2$: prodotto di due matrici (operazioni tra matrici).

La loro importanza risiede nel fatto che vengono eseguite un gran numero di volte. È perciò essenziale che siano implementate il più efficientemente possibile. Lo scopo si ottiene in due modi: scrivendo del buon codice e ottimizzando le routine sulla macchina che si ha a disposizione. Questa operazione non può prescindere dalla architettura della macchina. Riducendo la questione ai minimi termini, in un computer si distinguono i seguenti sistemi

- un blocco deputato a immagazzinare le istruzioni, i dati e i risultati (memoria);
- un blocco di calcolo e gestione (CPU);
- un insieme di dispositivi di ingresso e uscita (I/O);
- delle vie di collegamento tra le varie parti (BUS).

Le operazioni floating point richiedono che gli operandi risiedano nei registri della CPU. Poichè invece sono in memoria, è necessario prelevarli e trasportarli nella CPU, solo a questo punto il calcolo può essere eseguito. Quindi, il risultato dell'operazione va nuovamente trasferito in memoria. I trasferimenti degli operandi dalla memoria alla CPU e viceversa richiedono più tempo rispetto a quello necessario per effettuare la operazione vera e propria. Questo tempo è principalmente legato alle caratteristiche della memoria in quanto ogni accesso alla stessa richiede un certo tempo sia in lettura che in scrittura.

L'efficienza di un algoritmo viene valutato in base al tempo di esecuzione, che può essere suddiviso nel tempo impiegato per gli accessi alla memoria e il tempo impiegato per effettuare le operazioni vere e proprie.

$$T_s(N) = \underbrace{N_{mem} \cdot t_{mem}}_{\swarrow \quad \searrow \\ Algoritmo \quad Hardware} + \underbrace{N_{flop} \cdot t_{flop}}_{\swarrow \quad \searrow \\ Algoritmo \quad Hardware}$$

$$T_{mem}(N) + T_{flop}(N)$$

Un software raggiunge prestazioni massime quando

$$T_s(N) \leq T_{flop}(N) \quad \Rightarrow \quad \frac{T_s(N)}{T_{flop}(N)} \leq 1$$

cioè quando il tempo di esecuzione $T_s(N)$ è al più uguale al tempo $T_{flop}(N)$ dell'algoritmo. In generale

$$\frac{T_s(N)}{T_{flop}(N)} = \frac{N_{mem} \cdot t_{mem} + N_{flop} \cdot t_{flop}}{N_{flop} \cdot t_{flop}} = 1 + \frac{N_{mem} \cdot t_{mem}}{N_{flop} \cdot t_{flop}}$$

Per migliorare l'efficienza bisogna ridurre il rapporto $\frac{N_{mem} \cdot t_{mem}}{N_{flop} \cdot t_{flop}}$, quindi ridurre il numero di accessi alla memoria e il tempo di accesso alla memoria. In base a quanto detto, nella esecuzione di un certo numero di operazioni floating point, ad esempio quelle coinvolte nel prodotto di due matrici, è importante movimentare (tra memoria e registri di CPU) il minor numero di dati possibile per effettuare le operazioni richieste. Per ottenere un buon risultato è necessario che l'algoritmo di calcolo e la sua implementazione siano progettati in modo che il numero dei dati movimentati sia di molto inferiore al numero di operazioni effettuate:

$$N_{flop} \gg N_{mem}$$

Un buon indice di efficienza è dato dal fattore q definito come

$$q = \frac{N_{flop}}{N_{mem}} \quad \text{in breve} \quad q = \frac{f}{m}$$

dove f è il numero di operazioni floating point da farsi ed m il numero di dati movimentati ovvero accessi alla memoria lenta. Dunque, assumendo il numero di flops prefissato per un dato compito (ad esempio, la moltiplicazione di due matrici quadrate di dimensione n richiede $2n^3$ flops), quanto più alto è q tanto migliore è l'algoritmo che svolge la suddetta mansione, in quanto a parità di operazioni movimenta meno dati e dunque richiede meno tempo.
Possiamo valutare il parametro q per le operazioni di base di algebra lineare.

• Prodotto scalare-vettore

$$y = \alpha \cdot x$$

con x, y vettori di dimensione n, α scalare. Il numero di trasferimenti per effettuare tale prodotto sono

1. preleva dalla memoria $x \rightarrow n$ accessi

2. preleva dalla memoria $y \rightarrow n$ accessi
3. preleva dalla memoria $\alpha \rightarrow 1$ accesso
4. calcolo $\rightarrow n$ operazioni fl.p.
5. ripone nella memoria $y \rightarrow n$ accessi

da cui segue

$$q = \frac{f}{m} = \frac{n}{3n+1} \approx \frac{1}{3}$$

ovvero il numero degli accessi alla memoria è maggiore del numero delle operazioni floating point, $m > f$.

• Prodotto interno tra vettori

$$\alpha = x^T \cdot y$$

con x, y vettori colonna di dimensione n, α scalare. Il numero di trasferimenti per effettuare tale prodotto sono

1. preleva dalla memoria $x \rightarrow n$ accessi
2. preleva dalla memoria $y \rightarrow n$ accessi
3. calcolo $\rightarrow 2n$ operazioni op.fl.p.
4. ripone nella memoria $\alpha \rightarrow 1$ accesso

da cui segue

$$q = \frac{f}{m} = \frac{2n}{2n+1} \approx 1$$

ovvero il numero degli accessi alla memoria è uguale al numero delle operazioni floating point, $m = f$.

• Prodotto matrice-vettore

$$y = M \cdot x$$

con x, y vettori di dimensione n, M matrice quadrata di dimensione n. Il numero di trasferimenti per effettuare tale prodotto sono

1. preleva dalla memoria $x \rightarrow n$ accessi
2. preleva dalla memoria $y \rightarrow n$ accessi
3. preleva dalla memoria $M \rightarrow n^2$ accessi
4. calcolo $\rightarrow 2n^2$ operazioni op.fl.p.
5. ripone nella memoria $y \rightarrow n$ accessi

da cui segue

$$q = \frac{f}{m} = \frac{2n^2}{n^2 + 3n} \approx 2$$

ovvero il numero degli accessi alla memoria è minore del numero delle operazioni floating point, $m < f$.

• Prodotto matrice-matrice

$$C = A \cdot B$$

con A, B matrici quadrate di dimensione n, C matrice quadrata di dimensione n. Il numero di trasferimenti per effettuare tale prodotto sono

1. preleva dalla memoria $A \rightarrow n^2$ accessi
2. preleva dalla memoria $B \rightarrow n^2$ accessi
3. preleva dalla memoria $C \rightarrow n^2$ accessi
4. calcolo $\rightarrow 2n^3$ operazioni op.fl.p.
5. ripone nella memoria $C \rightarrow n^2$ accessi

da cui segue

$$q = \frac{f}{m} = \frac{2n^3}{4n^2} \approx \frac{n}{2}$$

ovvero il numero degli accessi alla memoria è di molto minore del numero delle operazioni floating point, $m \ll f$.

Dunque il rapporto q dipende dal nucleo computazionale di base, un prodotto tra matrici è più conveniente in termini di efficienza rispetto ad un prodotto scalare, conviene quindi riorganizzare gli algoritmi a blocchi in modo da utilizzare il più possibile il nulceo computazionale matrice-matrice.
Per quasi tutti i problemi dell'algebra lineare numerica sono già stati creati degli insiemi di routine idonei a trattarli nel modo più efficiente possibile; alcune di queste raccolte, o librerie, sono disponibili gratuitamente in rete. Tra queste figurano le Blas, (rivolte alle operazioni fondamentali), le Lapack (idonee a trattare un vasto insieme di problemi, che spaziano dalla risoluzione di sistemi lineari alla ricerca di autovalori e autovettori in senso genearalizzato), le Sparskit (utili in tutti quei casi in cui si hanno problemi di grandi dimensioni che coinvolgono matrici sparse). Nella scrittura di un programma è più conveniente e performante attingere a queste librerie piuttosto che non scrivere in toto del nuovo codice. Al fine di incrementare ulteriormente le prestazioni del codice è utile "calibrare" le librerie al tipo di macchina disponibile in modo da sfruttarne appieno la potenza di calcolo. Questa calibratura è, talvolta, già fatta dal venditore della macchina; piú frequentemente è necessario provvedere in proprio e a questo proposito si può ricorrere a quanto riportato nel progetto Atlas.

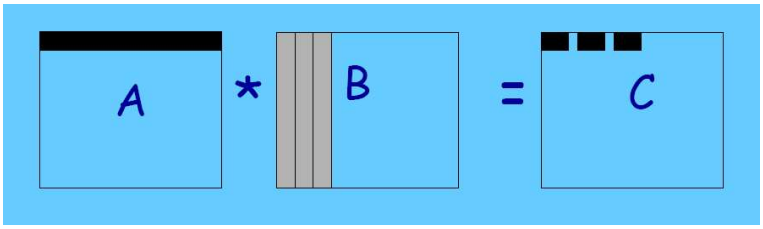
2 BLAS

BLAS è l'acronimo per Basic Linear Algebra Subprogram. Libreria di software matematico per l'esecuzione di operazioni di base del calcolo matriciale che ottimizza gli accessi alla memoria.

L'evoluzione della tecnologia delle CPU e delle memorie è rispecchiata nelle successive modifiche delle BLAS: si parte (1979) dalle BLAS1, concepite per eseguire solo operazioni scalari tra due vettori, si continua (1988) con le BLAS2 dove si eseguono anche operazioni tra matrici e vettori, e si arriva (1989) alle BLAS3 in cui l'operazione fondamentale è il prodotto tra due matrici.
La necessità di rivedere le versioni precedenti è essenziale: infatti, il codice delle varie versioni della BLAS è scritto in modo da sfruttare al meglio le caratteristiche della macchina. Nel momento in cui la tecnologia apporta dei cambiamenti radicali alla architettura, i vecchi meccanismi di ottimizzazione diventano obsoleti o comunque non più in grado di sfruttare le nuove potenzialità.
L'esecuzione del prodotto tra due matrici quadrate A e B di dimensione n su di una moderna macchina dotata di cache utilizzando le idee alla base degli algoritmi BLAS1, BLAS2, BLAS3 permette di apprezzare le tre differenti filosofie. Gli algoritmi presi in considerazione sono descritti in dettaglio nel seguito.

• **Algoritmo 1 (BLAS 1)**

La moltiplicazione $C = A * B$ è realizzata nel modo classico : $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Quest'ultima operazione, che di fatto è il prodotto interno di due vettori, è effettuata mediante una routine BLAS1.



L'algoritmo costruisce la matrice C per righe effettuando il prodotto interno tra due vettori, è perciò il seguente

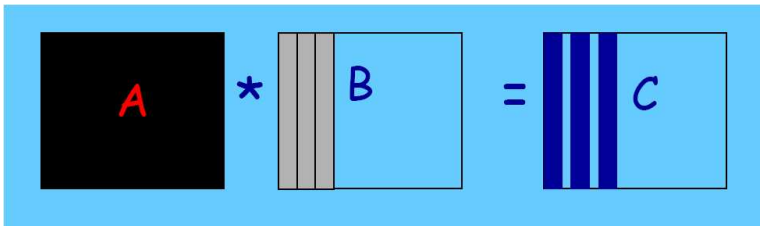
```
for  i=1:n
   for j=1:n
      C(i,j)=A(i,:)* B(:,j)
   end
end
```

• **Algoritmo 2 (BLAS 2)**

L'algoritmo effettua il prodotto delle matrici quadrate A e B basandosi sulla seguente relazione

$$C = A * B = \sum_{k=1}^{n} A(:,k) * B(k,:).$$

Infatti, il posto (i, j) del prodotto $A(:, k) * B(k, :)$ contiene $A(i, k) \cdot B(k, j)$ e perciò $C(i, j) = \sum_{k=1}^{n} A(i, k)B(k, j)$ che è la espressione usuale.
L'utilizzo della precedente relazione permette di mettere a punto una strategia per la esecuzione del prodotto tra due matrici basata sulle BLAS2. L'idea di fondo è quella di partizionare la matrice B in N blocchi contigui, ciascuno formato da n righe e da $p = n/N$ colonne: B_j, $j = 1, \ldots, N$. Così, ad esempio, B_1 contiene le prime p colonne di B, B_2 le colonne dalla $p + 1$ alla $2p$ e cosi via. Lo stesso tipo di suddivisione in N blocchi di colonne contigue è fatto sulla matrice C, il che fornisce i blocchi C_j, $j = 1, \ldots, N$.

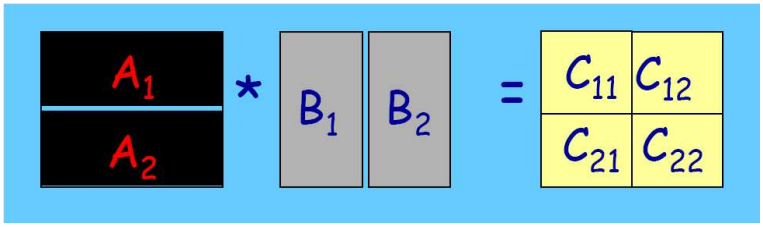


Quindi, si applica la relazione appena scritta al calcolo dei prodotti $C_j = A * B_j$, che sono matrici rettangolari n x p. Per il calcolo del singolo blocco C_j si ha perciò il seguente algoritmo

for j=1:N
    $C_j \leftarrow 0$
    for k=1:n
        $C_j = C_j + A(:, k) * B_j(k, :)$
    end
end

• **Algoritmo 3 (BLAS 3)**

In questo caso si partizionano le matrici A, B e C in N x N blocchi ciascuno dei quali ha dimensione p x p con $p = n/N$. Per esempio $N = 2$



L'algoritmo ha la forma

for i=1:N
  for j=1:N
    $C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$
  end
end

dove, ad esempio, A_{ij} indica il generico blocco p x p di riga i e colonna j.

Se facciamo eseguire questi algoritmi da un calcolatore per il quale la libreria BLAS è stata ottimizzata, otteniamo risultati di questo tipo:

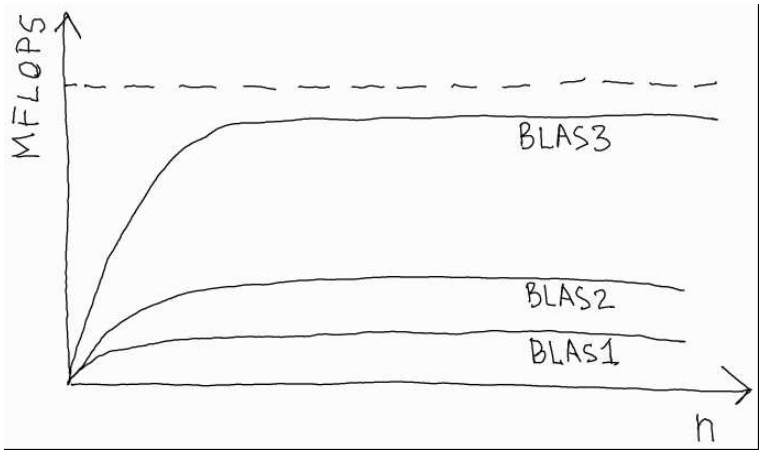


Figura 1: Performance delle BLAS in Megaflops al variare delle dimensioni n.

dunque, formulazioni identiche dal punto di vista matematico possono portare a performance molto differenti dal punto di vista delle prestazioni di calcolo. Notare che, in questo esempio, il numero di operazioni in virgola mobile è sempre uguale a $2 \cdot n^3$ in tutti e tre i casi. Questo risultato ha validità generale: le operazioni BLAS3 sono molto più veloci delle BLAS2 e BLAS1 nelle macchine di calcolo di ultima generazione, e dunque vanno utilizzate il più possibile negli algoritmi di ALN per avere buone prestazioni dal calcolatore.

3 Calcolo delle prestazioni ottenibili dalle BLAS

Vediamo di spiegare perchè le routines BLAS3 sono più performanti ed in quali condizioni. Una caratteristica sempre più frequente dei problemi di ALN che sorgono nelle applicazioni reali è quella di avere matrici di grandi dimensioni, e dunque una criticità nasce dal punto di vista dell'occupazione di memoria.
Lo sviluppo tecnologico delle CPU moderne ha portato a richiedere tempi ridottissimi per l'esecuzione di un'operazione in virgola mobile; non così è stato per la velocità di trasferimento dei dati tra i livelli di memoria: dato che per eseguire un'operazione in virgola mobile

la CPU deve poter caricare gli operandi dalla memoria veloce (cache) nei suoi registri e che questa, per ragioni tecnologiche e di costo, ha dimensioni limitate, la criticità maggiore per le prestazioni sul tempo di calcolo è mantenere impegnata la CPU.
Si è detto che per ottenere un buon risultato è necessario che l'algoritmo di calcolo e la sua implementazione siano progettati in modo che il numero dei dati movimentati sia di molto inferiore al numero di operazioni effettuate:

$$N_{flop} \gg N_{mem}$$

Per le BLAS succede che, detta n la dimensione di vettori e matrici:

	N_{mem}	N_{flop}
BLAS1	$o(n)$	$o(n)$
BLAS2	$o(n^2)$	$o(n^2)$
BLAS3	$o(n^2)$	$o(n^3)$

dunque BLAS3, se implementata correttamente, è potenzialmente adatta ad ottenere le migliori prestazioni.
Cerchiamo ora di valutare $q = \frac{f}{m}$ per ognuno dei tre algoritmi precedenti per il prodotto di matrici. Assumeremo per la memoria un modello semplificato, ovvero supponiamo due soli livelli di memoria ed i dati vengono trasferiti singolarmente dalla memoria lenta, in cui risiedono i dati, a quella veloce, in cui si portano i dati necessari alla CPU mentre nella realtà i trasferimenti tra cache e memoria principale avvengono per 'righe', e tra memoria principale e secondaria avvengono per 'pagine'.
Per tutti gli algoritmi si ha: $f = 2 \cdot n^3$. Supponiamo $m = m_A + m_B + m_C$, cioè scomposto nella somma degli accessi alla matrice A, alla matrice B ed alla matrice C e calcoliamone il valore. Ispezionando l'implementazione dei singoli algoritmi, è possibile determinare il numero di accessi effettuati:

• **BLAS 1**

```
for  i=1:n
    for j=1:n
        C(i,j)=A(i,:)* B(:,j)
    end
end
```

L'algoritmo costruisce la matrice C per righe. Per costruire la riga i-esima di C, è caricata in primo luogo in memoria veloce la riga i-esima di A, $A(i,:)$, il che comporta il trasferimento di n dati. Quindi, per il calcolo di $C(i,j)$ è portata in memoria veloce anche la colonna j-esima di B, il che comporta il trasferimento di altri n dati. A questo punto è effettuato, per mezzo di una routine BLAS 1, il prodotto scalare $C(i,j) = A(i,:) * B(:,j)$ e il risultato è depositato in memoria lenta, il che equivale al trasferimento di un ulteriore dato. Perciò, prescindendo per il momento dal caricamento di $A(i,:)$, il calcolo di $C(i,j)$ comporta il trasferimento di $n+1$ dati. Ne segue che per l'intera riga $C(i,:)$ sono richiesti $n + n(n+1) = n^2 + 2n$ trasferimenti, dove i primi n sono associati al caricamento di $A(i,:)$. Infine, il calcolo della

intera matrice C comporta $n(n^2+2n) = n^3+2n^2$ dati movimentati.
Dunque $m_A = n^2$, $m_B = n \cdot n^2$, $m_C = n^2$, ne risulta un fattore di merito

$$q_1 = \frac{2n^3}{n^3+2n^2} \approx 2,$$

dove la approssimazione vale per valori elevati di n. Ciò significa che vengono effettuate, in media, 2 operazioni floating point per dato trasferito.

• **BLAS 2**

for j=1:N
    $C_j \leftarrow 0$
    for k=1:n
        $C_j = C_j + A(:,k) * B_j(k,:)$
    end
end

Il numero di trasferimenti si può calcolare nel modo seguente. Ad ogni j corrisponde il caricamento in memoria veloce del blocco B_j. Questo comporta il trasferimento di $np = n^2/N$ dati. Inoltre è caricata la matrice C_j composta anch'essa da n^2/N dati tutti nulli. Il ciclo interno, fatto sulla variabile k, richiede ad ogni passo il caricamento in memoria veloce della colonna k-esima di A, ossia n dati. Non è necessario ricaricare B_j e C_j in quanto sono già presenti. L'esecuzione dell'intero ciclo interno comporta, in successione, il caricamento in memoria veloce di tutte le colonne di A per un totale di n^2 trasferimenti. In conclusione, il passo j-esimo richiede $n^2/N + n^2/N + n^2 = 2n^2/N + n^2$ trasferimenti di dati dalla memoria lenta a quella veloce. Una volta concluso il ciclo interno, la matrice $C_j = A * B_j$ viene trasferita in memoria lenta, il che richiede $np = n^2/N$ trasferimenti. Il ciclo in j è eseguito N volte. Poiché ad ogni esecuzione sono trasferiti $3n^2/N + n^2$ dati, il numero totale di dati movimentati risulta $N(3n^2/N + n^2) = 3n^2 + Nn^2$.
Dunque $m_A = N \cdot n^2$, $m_B = n^2$, $m_C = 2 \cdot n^2$, di conseguenza, il fattore q assume l'espressione

$$q_2 = \frac{2n^3}{3n^2+Nn^2} \approx \frac{2n}{N}, \quad N \gg 3.$$

Quindi, q_2 cresce al calare di N, ossia del numero di blocchi in cui è partizionato il calcolo. Per altro verso, al calare di N cresce il numero di dati, associati ad ogni singolo blocco, che devono poter contemporaneamente risiedere in memoria veloce. Dato che la memoria veloce ha una capacità ridotta, c'è un limite inferiore per N al di sotto del quale B_j e C_j non possono risiedere in memoria veloce.
Si osserva, inoltre, la natura parallela dell'algoritmo: ogni singolo blocco C_j può essere calcolato indipendentemente dagli altri blocchi richiedendo i soli dati di ingresso A e B_j. Perciò, disponendo di N processori che lavorano indipendentemente, ciascuno dei quali calcola C_j, il tempo di calcolo totale per $C = A * B$, è ridotto di un fattore N rispetto al caso in cui si abbia a disposizione un solo processore.

• **Algoritmo 3 (BLAS 3)**

In questo caso si partizionano le matrici A, B e C in N x N blocchi ciascuno dei quali ha dimensione p x p con $p = n/N$. L'algoritmo ha la forma

for i=1:N
  for j=1:N
    $C_{ij} = \sum_{k=1}^{N} A_{ik}B_{kj}$
  end
end

dove, ad esempio, A_{ij} indica il generico blocco p x p di riga i e colonna j. Fissati i e j, i dati portati dalla memoria lenta a quella veloce sono relativi ai blocchi A_{ik} e B_{kj}, $k = 1, \ldots, N$. Ciascuno di questi blocchi consta di $p^2 = (n/N)^2$ dati. Dunque il calcolo di C_{ij} richiede il trasferimento in memoria veloce di $N(2n^2/N^2) = 2n^2/N$ dati. Una volta calcolato C_{ij}, lo si deve trasferire in memoria lenta: questo equivale a $p^2 = n^2/N^2$ trasferimenti. Il calcolo del prodotto $C = A * B$ richiede la valutazione di $N \cdot N = N^2$ prodotti del tipo C_{ij}. Poiché in ciascuno dei prodotti C_{ij} sono necessari $2n^2/N + n^2/N^2$ trasferimenti, il numero totale di dati movimentati dall'algoritmo è pari a $N^2(2n^2/N + n^2/N^2) = 2Nn^2 + n^2$.
Dunque $m_A = N \cdot n^2$, $m_B = N \cdot n^2$, $m_C = n^2$, conseguentemente il fattore q risulta

$$q_3 = \frac{2n^3}{2Nn^2 + n^2} \approx \frac{n}{N}, \quad N \gg 1.$$

Anche in questo caso, N non può essere troppo piccolo: infatti un N piccolo equivale ad avere partizionato le matrici in blocchi di dimensione p elevata a cui corrisponde un grande numero di dati (pari a $3p^2$) che devono contemporaneamente risiedere in memoria veloce. Questo aspetto contrasta con la capacità limitata della memoria veloce.

A conclusione, si possono fare alcune considerazioni.

- Se $N = n$ i tre metodi coincidono e corrispondono all'algoritmo 1.
- Più piccolo è N più grande risulta q e più efficiente l'algoritmo.
- Gli algoritmi 2 e 3 richiedono, al fine di ottenere il valore massimo del corrispondente fattore q, il calcolo del valore minimo di N.

Quest'ultimo va fatto con riferimento ad una data capacità di memoria veloce, che è quella in dotazione della macchina. Convenzionalmente, in questo contesto, si indica la capacità della memoria veloce con il numero massimo di dati (in floating point in doppia precisione) che può contenere; sia M questo numero. Nel caso del secondo algoritmo, il numero di dati richiesti in memoria veloce è pari a $np + n + np = 2np + n = 2n^2/N + n$, dove le prime np locazioni sono riservate a C_j, le successive n locazioni alla k-esima colonna di A, le ultime np locazioni al blocco B_j. Perciò, i valori di N sono soggetti al vincolo

$$\frac{2n^2}{N} + n \leq M \quad \Longrightarrow \quad N \geq \frac{2n^2}{M - n} \approx \frac{2n^2}{M}.$$

In base a ciò, il valore massimo del fattore q_2 risulta

$$q_2^{(max)} \approx \frac{2n}{2n^2/M} = \frac{M}{n} \tag{1}$$

Poiché M è fissata, q_2^{max} diminuisce al crescere di n, ossia della dimensione del problema. In altri termini, al crescere di n cala il numero di operazioni floating point per trasferimento; ciò comporta un degrado delle prestazioni del sistema in quanto si spende un tempo sempre maggiore nei trasferimenti dei dati invece che nella esecuzione di operazioni. Si consideri, ora, il terzo algoritmo. In questo caso, la memoria veloce deve essere in grado di ospitare $3(n/N)^2$ dati, essendo $(n/N)^2$ il numero di dati associato a ciascuna delle matrici C_{ij}, A_{ik}, B_{ik}. Ne nasce il vincolo

$$3\frac{n^2}{N^2} \leq M \implies N \geq \sqrt{\frac{3n^2}{M}},$$

per cui il valore massimo del fattore q è

$$q_3^{(max)} \approx \frac{n}{\sqrt{3n^2/M}} = \sqrt{\frac{M}{3}} \tag{2}$$

valore che non dipende più da n. In definitiva, nei sistemi dotati di memoria veloce, l'algoritmo 3 mantiene inalterate le prestazioni al variare della dimensione del problema; inoltre, le prestazioni migliorano all'aumentare della quantità di memoria veloce.

4 LAPACK

LAPACK è l'acronimo per Linear Algebra PACKage. È una libreria dedicata all'algebra lineare numerica. Con riferimento a matrici dense o a banda, permette di trattare i seguenti problemi:

- sistemi lineari (compresa la loro soluzione nel senso dei minimi quadrati);
- autovalori e autovettori (anche in senso generalizzato);
- stima del numero di condizionamento;
- fattorizzazione di matrici.

Il codice delle routine LAPACK è scritto in modo da appoggiarsi il più possibile alle BLAS3; come conseguenza, le prestazioni ottimali si ottengono "calibrando" preliminarmente le BLAS3 alla specifica macchina sulla quale si sta lavorando.
Nel pacchetto LAPACK sono distinguibili tre ampie classi di routine:

1. driver routines;
2. computational routines;
3. auxiliary routines.

Le prime permettono di risolvere problemi generali. Ad esempio, appartengono a questo gruppo le routine per la risoluzione di sistemi lineari e per il calcolo di autovalori e autovettori. Il secondo tipo descrive routine deputate a svolgere uno specifico compito: ad esempio, il calcolo degli autovalori per una matrice bidiagonale o la soluzione di un sistema lineare per una matrice a bande. Le auxiliary routines risolvono anch'esse uno specifico compito, ma ad un livello più basso rispetto a quello delle computational routines. LAPACK è installato anche nelle versioni Matlab dalla 5.3 compresa in poi.

5 Ottimizzazione empirica automatizzata della libreria software BLAS: il progetto ATLAS

Ottimizzare un software è comunque un'impresa costosa, sopratutto in tempo umano. I calcolatori sono differenti tra di loro per clock, dimensioni delle cache, associatività delle cache, numero e profondità delle pipeline di calcolo.... In linea di principio ogni modifica ad un processore (ad esempio dimensione della cache) implica rimodificare (parzialmente) il processo di ottimizzazione. Perchè non farlo fare alla libreria stessa?
Recentemente è stato portato a termine un progetto molto interessante riguardo alle BLAS: la costruzione di un software che durante la sua esecuzione, genera la libreria BLAS ottimizzata per la macchina su cui è in esecuzione. Questo software è disponibile gratuitamente al sito: http://www.netlib.org/atlas.
ATLAS è acronimo per Automatic Tuned Linear Algebra Software, il suo successo è notevole, dato che riesce ad ottenere ottimizzazioni delle librerie tali da raggiungere le prestazioni ottenute dalle librerie commerciali, ottimizzate 'a mano' da esperti. Provandolo su un PC e confrontandone le prestazioni con le BLAS 'generali', si può notare che che la versione 'atlas' delle BLAS è decine di volte più veloce delle BLAS 'generali', già per una moltiplicazione di matrici di dimensione 1000. Questo dimostra come le prestazioni di calcolo possono essere ottenute solo sfruttando in maniera sapiente le risorse hardware a disposizione, che possono variare considerevolmente anche solo restando nel mondo del PC (es. un parametro fondamentale è la dimensione della memoria cache).