Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Partizionamento del grafo dataflow prodotto dal compilatore Chiara per la risoluzione di un sistema di equazioni lineari con il metodo di Gauss-Seidel

Giovanni Gallo, Lorenzo Verdoscia

RT-ICAR-NA-2010-06 **Dicembre 2010**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Partizionamento del grafo dataflow prodotto dal compilatore Chiara per la risoluzione di un sistema di equazioni lineari con il metodo di Gauss-Seidel

Giovanni Gallo, Lorenzo Verdoscia[1]

Rapporto Tecnico N.:
RT-ICAR-NA-2010-06

Data:
Dicembre 2010

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, Via P. Castellino 111, 80131 Napoli

Indice

Elenco delle figure

1 Introduzione

Il presente lavoro affronta lo studio dei problemi di partizionamento ottimo su grafi, una vasta e importante famiglia di problemi di ottimizzazione combinatoria. In linea generale, un problema di partizionamento ottimo richiede di suddividere un insieme di oggetti, correlati da un tessuto più o meno complesso di relazioni, in insiemi a due a due disgiunti in modo tale da soddisfare vincoli di varia natura ed in modo da raggiungere o, quanto meno, tendere il più possibile ad un obiettivo prefissato. Problemi di questo tipo si presentano in numerosissimi ambiti applicativi scientifici e ingegneristici in cui la situazione pratica oggetto d'analisi può essere schematizzata sotto forma di grafo. La riproduzione di immagini digitali, la progettazione di circuiti VLSI, l'uso razionale della memoria distribuita di un calcolatore parallelo, la visualizzazione di grafi di grandi dimensioni, lo scheduling degli equipaggi e la collocazione di servizi fondamentali all'interno di un territorio sono solo alcuni esempi di applicazioni pratiche che fanno uso di tecniche di partizionamento e clustering. Il problema reale modellizzato dal grafo impone vincoli che limitano l'arbitrarietà della partizione. Solitamente, sussistono vincoli di natura topologica sui sottografi indotti dalla partizione. Le grandezze caratterizzanti del problema concorrono alla definizione di una funzione della partizione stessa, scelta in modo tale da ottenere, minimizzandola o massimizzandola, la migliore partizione possibile del grafo, conformemente all'obiettivo preposto. In molte applicazioni reali la dimensione del problema può essere notevole, e si rende quindi necessaria l'adozione di tecniche risolutive efficienti sotto il profilo della complessità computazionale. La risoluzione di tali problemi è tanto più complicata, quanto più complessi sono l'obiettivo che ci si pone e la struttura del problema. Tuttavia, anche obiettivi e strutture apparentemente semplici, possono generare problemi computazionalmente molto complicati: sono i cosiddetti problemi NP-completi. In questo lavoro analizzeremo il metodo iterativo di Gauss-Seidel, la sua implementazione nel linguaggio funzionale Chiara[1] e il partizionamento del grafo risultante dalla fase di compilazione mediante il tool di partizionamento Chaco[2].

2 I metodi iterativi stazionari

La soluzione numerica della maggior parte dei problemi di interesse nella fisica e nell'ingegneria in particolare, anche molto complessi, si riduce alla soluzione di un sistema di equazioni algebriche lineari. Ad esempio, i siste-

mi a parametri concentrati lineari in condizioni stazionarie sono governati da equazioni algebriche lineari. Inoltre, se si risolve un sistema di equazioni algebriche non lineari con il metodo di **Newton-Raphson** ad ogni passo bisogna risolvere un sistema di equazioni algebriche lineari. Più in generale, la discretizzazione di equazioni differenziali e integrali lineari porta alla soluzione di sistemi di equazioni algebriche lineari mrediante l'impiego di metodi iterativi, facilmente utilizzabili dai sistemi di calcolo automatico.
Considereremo il seguente problema. Dato un vettore $b \in \mathbb{R}^n$ e una matrice $A = (a_{i,j})$, $i, j = 1, 2, \ldots, n$, si cerca un vettore $x \in \mathbb{R}^n$ tale che:

$$Ax = b \Leftrightarrow \begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = b_n \end{cases} \tag{1}$$

in forma compatta si ha

$$\sum_{j=1}^{n} a_{i,j}x_j = b_i \qquad i = 1, 2, \ldots, n \tag{2}$$

Se si risolve la i-esima equazione rispetto ad x_i e si assume che gli altri valori di x sono fissi, si ottiene

$$x_i = \left(b_i - \sum_{j \neq i} a_{i,j}x_j \right) / a_{i,i}. \tag{3}$$

Il termine 'metodo iterativo' si riferisce ad un ampio range di tecniche che usano approssimazioni successive, ad ogni step, per ottenere soluzioni molto accurate nella soluzione di sistemi lineari. I metodi iterativi si dividono in stazionari e non stazionari. I metodi stazionari sono antichi, semplici da capire ed implementare, ma solitamente poco efficaci. I metodi non stazionari sono di recente sviluppo; la loro analisi è solitamente difficile da capire, ma possono essere altamente efficaci.
La velocità con la quale un metodo iterativo converge verso la soluzione del sistema dipende molto dallo spettro della matrice dei coefficienti. Per cui, i metodi iterativi solitamente richiedono una seconda matrice che trasforma la matrice dei coefficienti in un'altra dotata di uno spettro favorevole. Tale trasformazione di matrici è comunemente chiamata *preconditioner*. Un buon *preconditioner* migliora sufficientemente la convergenza di un metodo iterativo, in modo da superare il costo extra dovuto alla costruzione e applicazione del *preconditioner*. Infatti, senza il *preconditioner* un metodo iterativo può

perfino non convergere. I metodi iterativi che sono espressi nella seguente forma:

$$x^{(k+1)} = Bx^{(k)} + c \tag{4}$$

(dove B e c non dipendono dal passo di iterazione k) sono chiamati metodi iterativi stazionari.
In questo lavoro presenteremo un metodo iterativo stazionario: il metodo di Gauss-Seidel.
Il metodo di Gauss-Seidel è come il metodo di Jacobi, ad eccezione del fatto che questo metodo effettua la modifica dei valori non appena questi sono disponibili. In generale, se il metodo di Jacobi converge, quello di Gauss-Seidel lo fa molto più velocemente, nonostante risulti essere ancora relativamente lento.

3 Metodo di Gauss-Seidel

Consideriamo le equazioni lineari in (3): se esse sono valutate una alla volta e i risultati di una precedente computazione sono utilizzati appena risultano disponibili, otteniamo il metodo di Gauss-Seidel:

$$x_i^{(k+1)} = \left(b_i - \sum_{j<i} a_{i,j} x_j^{(k+1)} - \sum_{j>i} a_{i,j} x_j^{(k)} \right) / a_{i,i} \tag{5}$$

Da questa equazione è possibile notare due aspetti importanti che differenziano il metodo di Gauss-Seidel da Jacobi. Il primo aspetto evidenzia la natura seriale del calcolo perchè ciascuna componente della nuova iterazione dipende solo da tutte le componenti calcolate precedentemente.
Il secondo aspetto evidenzia la dipendenza tra la nuova iterazione $x^{(k+1)}$ e ordine di valutazione delle equazioni. Per tale motivo il metodo di Gauss-Seidel è anche chiamato *metodo delle sostituzioni successive*, proprio per indicare la dipendenza tra iterazione e ordine: se questo ordine cambia, cambiano anche le componenti di una nuova iterazione.
Questi due aspetti assumono un'importanza cruciale perchè se la matrice dei coefficienti A è sparsa, la dipendenza tra ogni componente di una nuova iterazione e quelle precedenti non è assoluta. La presenza di *zeri* nella matrice può, tuttavia, rimuovere l'influenza di alcune di queste componenti precedenti. Inoltre, scegliendo un ordine appropriato delle equazioni, è possibile ridurre il numero delle dipendenze, e, di conseguenza, rendere possibile la modifica in parallelo di gruppi di componenti. Va infine evidenziato che il riordino delle equazioni può avere effetto sulla velocità di convergenza del

metodo di Gauss-Seidel. Un ordine mal scelto può ridurre la velocità di convergenza; invece, ben scelto può migliorarla.
In termini matriciali, la definizione del metodo di Gauss-Seidel di (5) può essere espressa come

$$x^{(k+1)} = (D - L)^{-1} \left(U x^{(k)} + b \right) \tag{6}$$

Dove D è la matrice diagonale, -L è la matrice triangolare inferiore e -U è la matrice triangolare superiore di A.
Il metodo converge per tutti i sistemi lineari con matrice a diagonale principale strettamente dominante e la condizione d'arresto è $\mid x^{(k+1)} - x^{(k)} \mid < \epsilon$ (approssimazione richiesta).

4 Implementazione dell'algoritmo nel linguaggio Chiara

Per la risoluzione di un sistema lineare mediante il metodo di Gauss-Seidel, è stato scritto il programma 'gauss_seidel.chr' nel linguaggio funzionale Chiara che implementa l'algoritmo per un sistema di equazioni con $n = 14$ il cui grafo dataflow, generato dal compilatore del linguaggio, risulta significativo per essere partizionato ed eseguito da processori dataflow della macchina D^{3}AS, con ciascun processore costituito da 128 Unità Funzionali identiche.

Al fine di fornire una chiara spiegazione del programma scritto, il file del codice sorgente ('gauss_seidel.chr') è stato suddiviso in tre parti. Nella prima parte (Figura 1), sono presenti le costanti che definiscono la matrice dei coefficienti A, nella seconda parte (Figura 2), invece, sono presenti le costanti che definiscono il vettore b dei termini noti e il vettore diagonale principale, mentre nella terza parte sono definite le funzioni (Figura 3) che realizzano l'algoritmo di Gauss-Seidel.
Per quest'ultima parte il significato delle funzioni di Figura 3 è il seguente:

- **eps** definisce l'approssimazione desiderata;
- **IP** calcola il prodotto interno;
- **ABS** fornisce il valore assoluto di un numero;
- **XMENOXUP** calcola $x_i^{(k+1)} - x_i^{(k)}$ per $i = 1, \ldots, 14$;
- **VALOREASSOLUTOVETTORE** fornisce il valore assoluto del vettore calcolato da **XMENOXUP**;

```
def a1x1y = %0
def a1x2y = %3
def a1x3y = %3
def a1x4y = %1
def a1x5y = %2
def a1x6y = %3
def a1x7y = %2
def a1x8y = %2
def a1x9y = %2
def a1x10y = %1
def a1x11y = %1
def a1x12y = %2
def a1x13y = %1
def a1x14y = %2
def a2x1y = %2
def a2x2y = %0
def a2x3y = %2
def a2x4y = %3
def a2x5y = %2
def a2x6y = %1
def a2x7y = %2
def a2x8y = %2
def a2x9y = %1
def a2x10y = %2
def a2x11y = %2
def a2x12y = %3
def a2x13y = %2
def a2x14y = %1
def a3x1y = %3
def a3x2y = %1
def a3x3y = %0
def a3x4y = %3
def a3x5y = %2
def a3x6y = %3
def a3x7y = %2
def a3x8y = %3
def a3x9y = %2
def a3x10y = %2
def a3x11y = %3
def a3x12y = %3
def a3x13y = %1
def a3x14y = %2
def a4x1y = %2
def a4x2y = %3
def a4x3y = %3
def a4x4y = %0
def a4x5y = %3
def a4x6y = %2
def a4x7y = %2
def a4x8y = %2
def a4x9y = %1
def a4x10y = %3
def a4x11y = %1
def a4x12y = %3
def a4x13y = %3
def a4x14y = %2
def a5x1y = %1
def a5x2y = %1
def a5x3y = %1
def a5x4y = %2
def a5x5y = %0
def a5x6y = %1
def a5x7y = %1
def a5x8y = %2
def a5x9y = %3
def a5x10y = %1
def a5x11y = %3
def a5x12y = %1
def a5x13y = %3
def a5x14y = %3
def a6x1y = %1
def a6x2y = %2
def a6x3y = %2
def a6x4y = %1
def a6x5y = %3
def a6x6y = %0
def a6x7y = %3
def a6x8y = %1
def a6x9y = %3
def a6x10y = %1
def a6x11y = %1
def a6x12y = %3
def a6x13y = %1
def a6x14y = %1
def a7x1y = %2
def a7x2y = %2
def a7x3y = %2
def a7x4y = %1
def a7x5y = %1
def a7x6y = %2
def a7x7y = %0
def a7x8y = %2
def a7x9y = %1
def a7x10y = %1
def a7x11y = %2
def a7x12y = %3
def a7x13y = %1
def a7x14y = %2
def a8x1y = %3
def a8x2y = %3
def a8x3y = %2
def a8x4y = %2
def a8x5y = %1
def a8x6y = %2
def a8x7y = %3
def a8x8y = %0
def a8x9y = %3
def a8x10y = %1
def a8x11y = %2
def a8x12y = %1
def a8x13y = %3
def a8x14y = %2
def a9x1y = %3
def a9x2y = %1
def a9x3y = %1
def a9x4y = %2
def a9x5y = %1
def a9x6y = %1
def a9x7y = %1
def a9x8y = %2
def a9x9y = %0
def a9x10y = %2
def a9x11y = %2
def a9x12y = %1
def a9x13y = %1
def a9x14y = %1
def a10x1y = %2
def a10x2y = %1
def a10x3y = %3
def a10x4y = %1
def a10x5y = %3
def a10x6y = %3
def a10x7y = %3
def a10x8y = %3
def a10x9y = %1
def a10x10y = %0
def a10x11y = %3
def a10x12y = %2
def a10x13y = %1
def a10x14y = %1
def a11x1y = %2
def a11x2y = %3
def a11x3y = %3
def a11x4y = %2
def a11x5y = %1
def a11x6y = %1
def a11x7y = %2
def a11x8y = %2
def a11x9y = %2
def a11x10y = %2
def a11x11y = %0
def a11x12y = %1
def a11x13y = %3
def a11x14y = %2
def a12x1y = %2
def a12x2y = %3
def a12x3y = %2
def a12x4y = %2
def a12x5y = %1
def a12x6y = %2
def a12x7y = %2
def a12x8y = %1
def a12x9y = %1
def a12x10y = %2
def a12x11y = %3
def a12x12y = %0
def a12x13y = %2
def a12x14y = %3
def a13x1y = %1
def a13x2y = %2
def a13x3y = %1
def a13x4y = %1
def a13x5y = %1
def a13x6y = %2
def a13x7y = %1
def a13x8y = %1
def a13x9y = %1
def a13x10y = %3
def a13x11y = %2
def a13x12y = %3
def a13x13y = %0
def a13x14y = %1
def a14x1y = %2
def a14x2y = %3
def a14x3y = %2
def a14x4y = %1
def a14x5y = %2
def a14x6y = %2
def a14x7y = %2
def a14x8y = %3
def a14x9y = %3
def a14x10y = %3
def a14x11y = %3
def a14x12y = %3
def a14x13y = %3
def a14x14y = %0
```

Figura 1: Programma gauss_seidel.chr parte 1: costanti che definiscono la matrice dei coefficienti.

```
def riga1 = [a1x1y,a1x2y,a1x3y,a1x4y,a1x5y,a1x6y,a1x7y,a1x8y,a1x9y,a1x10y,a1x11y,a1x12y,a1x13y,a1x14y]
def riga2 = [a2x1y,a2x2y,a2x3y,a2x4y,a2x5y,a2x6y,a2x7y,a2x8y,a2x9y,a2x10y,a2x11y,a2x12y,a2x13y,a2x14y]
def riga3 = [a3x1y,a3x2y,a3x3y,a3x4y,a3x5y,a3x6y,a3x7y,a3x8y,a3x9y,a3x10y,a3x11y,a3x12y,a3x13y,a3x14y]
def riga4 = [a4x1y,a4x2y,a4x3y,a4x4y,a4x5y,a4x6y,a4x7y,a4x8y,a4x9y,a4x10y,a4x11y,a4x12y,a4x13y,a4x14y]
def riga5 = [a5x1y,a5x2y,a5x3y,a5x4y,a5x5y,a5x6y,a5x7y,a5x8y,a5x9y,a5x10y,a5x11y,a5x12y,a5x13y,a5x14y]
def riga6 = [a6x1y,a6x2y,a6x3y,a6x4y,a6x5y,a6x6y,a6x7y,a6x8y,a6x9y,a6x10y,a6x11y,a6x12y,a6x13y,a6x14y]
def riga7 = [a7x1y,a7x2y,a7x3y,a7x4y,a7x5y,a7x6y,a7x7y,a7x8y,a7x9y,a7x10y,a7x11y,a7x12y,a7x13y,a7x14y]
def riga8 = [a8x1y,a8x2y,a8x3y,a8x4y,a8x5y,a8x6y,a8x7y,a8x8y,a8x9y,a8x10y,a8x11y,a8x12y,a8x13y,a8x14y]
def riga9 = [a9x1y,a9x2y,a9x3y,a9x4y,a9x5y,a9x6y,a9x7y,a9x8y,a9x9y,a9x10y,a9x11y,a9x12y,a9x13y,a9x14y]
def riga10 = [a10x1y,a10x2y,a10x3y,a10x4y,a10x5y,a10x6y,a10x7y,a10x8y,a10x9y,a10x10y,a10x11y,a10x12y,a10x13y,a10x14y]
def riga11 = [a11x1y,a11x2y,a11x3y,a11x4y,a11x5y,a11x6y,a11x7y,a11x8y,a11x9y,a11x10y,a11x11y,a11x12y,a11x13y,a11x14y]
def riga12 = [a12x1y,a12x2y,a12x3y,a12x4y,a12x5y,a12x6y,a12x7y,a12x8y,a12x9y,a12x10y,a12x11y,a12x12y,a12x13y,a12x14y]
def riga13 = [a13x1y,a13x2y,a13x3y,a13x4y,a13x5y,a13x6y,a13x7y,a13x8y,a13x9y,a13x10y,a13x11y,a13x12y,a13x13y,a13x14y]
def riga14 = [a14x1y,a14x2y,a14x3y,a14x4y,a14x5y,a14x6y,a14x7y,a14x8y,a14x9y,a14x10y,a14x11y,a14x12y,a14x13y,a14x14y]

def matrice = [riga1,riga2,riga3,riga4,riga5,riga6,riga7,riga8,riga9,riga10,riga11,riga12,riga13,riga14]


def d1 = %26
def d2 = %26
def d3 = %31
def d4 = %31
def d5 = %24
def d6 = %24
def d7 = %23
def d8 = %29
def d9 = %20
def d10 = %28
def d11 = %27
def d12 = %27
def d13 = %21
def d14 = %33

def diagonale = [d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14]

def b1 = %2
def b2 = %2
def b3 = %1
def b4 = %1
def b5 = %2
def b6 = %2
def b7 = %1
def b8 = %1
def b9 = %3
def b10 = %2
def b11 = %1
def b12 = %2
def b13 = %2
def b14 = %3

def termininoti = [b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14]
```

Figura 2: Programma gauss_seidel.chr parte 2: costanti che definiscono il vettore dei termini noti e il vettore diagonale principale.

```
def eps = %0.000001
def IP = ! + @ & * @ trans
def ABS = (geq @ [id,%0] --> id; * @ [%-1,id])
def XMENOXUP = & - @ trans
def VALOREASSOLUTOVETTORE = & ABS
def CONTROLLO = ! + @ & (lt @ id-->%0; %1) @ distr
def CALX1 = [[ / @ [-, d1] @ [b1, IP] @ [1, riga1], 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]]
def CALX2 = [[ 1 @ 1 , / @ [-, d2] @ [b2, IP] @ [1, riga2], 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX1
def CALX3 = [[ 1 @ 1 , 2 @ 1 , / @ [-, d3] @ [b3, IP] @ [1, riga3], 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX2
def CALX4 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , / @ [-, d4] @ [b4, IP] @ [1, riga4], 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX3
def CALX5 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , / @ [-, d5] @ [b5, IP] @ [1, riga5], 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX4
def CALX6 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , / @ [-, d6] @ [b6, IP] @ [1, riga6], 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX5
def CALX7 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , / @ [-, d7] @ [b7, IP] @ [1, riga7], 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX6
def CALX8 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , / @ [-, d8] @ [b8, IP] @ [1, riga8], 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX7
def CALX9 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , / @ [-, d9] @ [b9, IP] @ [1, riga9], 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX8
def CALX10 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , / @ [-, d10] @ [b10, IP] @ [1, riga10], 11 @ 1 , 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX9
def CALX11 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , / @ [-, d11] @ [b11, IP] @ [1, riga11], 12 @ 1 , 13 @ 1 , 14 @ 1 ]] @ CALX10
def CALX12 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , / @ [-, d12] @ [b12, IP] @ [1, riga12], 13 @ 1 , 14 @ 1 ]] @ CALX11
def CALX13 = [[ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , / @ [-, d13] @ [b13, IP] @ [1, riga13], 14 @ 1 ]] @ CALX12
def CALX14 = [ 1 @ 1 , 2 @ 1 , 3 @ 1 , 4 @ 1 , 5 @ 1 , 6 @ 1 , 7 @ 1 , 8 @ 1 , 9 @ 1 , 10 @ 1 , 11 @ 1 , 12 @ 1 , 13 @ 1 , / @ [-, d14] @ [b14, IP] @ [1, riga14]] @ CALX13
def ITER = [2, CONTROLLO @ [VALOREASSOLUTOVETTORE @ XMENOXUP, eps]] @ [1, CALX14]
def TEST = neq @ [2, %0]



(repeat ITER, TEST) :<<2,2,1,1,2,2,1,1,3,2,1,2,2,3>,0>

stop
```

Figura 3: Programma gauss_seidel.chr parte 3: funzioni utilizzate per implementare l'algoritmo di Gauss-Seidel.

- **CONTROLLO** fornisce in uscita il 'token valido' se la condizione $\mid x_i^{(k+1)} - x_i^{(k)} \mid$<eps è soddisfatta, il valore bottom '$\perp$' altrimenti;
- **CALX1** calcola $x_1^{(k+1)} = \frac{b_1 - \sum_{j<1} a_{1,j} x_j^{(k+1)} - \sum_{j>1} a_{1,j} x_j^{(k)}}{a_{1,1}}$;
- **CALX2** calcola $x_2^{(k+1)} = \frac{b_2 - \sum_{j<2} a_{2,j} x_j^{(k+1)} - \sum_{j>2} a_{2,j} x_j^{(k)}}{a_{2,2}}$;
- **CALX3** calcola $x_3^{(k+1)} = \frac{b_3 - \sum_{j<3} a_{3,j} x_j^{(k+1)} - \sum_{j>3} a_{3,j} x_j^{(k)}}{a_{3,3}}$;
- **CALX4** calcola $x_4^{(k+1)} = \frac{b_4 - \sum_{j<4} a_{4,j} x_j^{(k+1)} - \sum_{j>4} a_{4,j} x_j^{(k)}}{a_{4,4}}$;
- **CALX5** calcola $x_5^{(k+1)} = \frac{b_5 - \sum_{j<5} a_{5,j} x_j^{(k+1)} - \sum_{j>5} a_{5,j} x_j^{(k)}}{a_{5,5}}$;
- **CALX6** calcola $x_6^{(k+1)} = \frac{b_6 - \sum_{j<6} a_{6,j} x_j^{(k+1)} - \sum_{j>6} a_{6,j} x_j^{(k)}}{a_{6,6}}$;
- **CALX7** calcola $x_7^{(k+1)} = \frac{b_7 - \sum_{j<7} a_{7,j} x_j^{(k+1)} - \sum_{j>7} a_{7,j} x_j^{(k)}}{a_{7,7}}$;
- **CALX8** calcola $x_8^{(k+1)} = \frac{b_8 - \sum_{j<8} a_{8,j} x_j^{(k+1)} - \sum_{j>8} a_{8,j} x_j^{(k)}}{a_{8,8}}$;
- **CALX9** calcola $x_9^{(k+1)} = \frac{b_9 - \sum_{j<9} a_{9,j} x_j^{(k+1)} - \sum_{j>9} a_{9,j} x_j^{(k)}}{a_{9,9}}$;
- **CALX10** calcola $x_{10}^{(k+1)} = \frac{b_{10} - \sum_{j<10} a_{10,j} x_j^{(k+1)} - \sum_{j>10} a_{10,j} x_j^{(k)}}{a_{10,10}}$;
- **CALX11** calcola $x_{11}^{(k+1)} = \frac{b_{11} - \sum_{j<11} a_{11,j} x_j^{(k+1)} - \sum_{j>11} a_{11,j} x_j^{(k)}}{a_{11,11}}$;
- **CALX12** calcola $x_{12}^{(k+1)} = \frac{b_{12} - \sum_{j<12} a_{12,j} x_j^{(k+1)} - \sum_{j>12} a_{12,j} x_j^{(k)}}{a_{12,12}}$;
- **CALX13** calcola $x_{13}^{(k+1)} = \frac{b_{13} - \sum_{j<13} a_{13,j} x_j^{(k+1)} - \sum_{j>13} a_{13,j} x_j^{(k)}}{a_{13,13}}$;
- **CALX14** calcola $x_{14}^{(k+1)} = \frac{b_{14} - \sum_{j<14} a_{14,j} x_j^{(k+1)} - \sum_{j>14} a_{14,j} x_j^{(k)}}{a_{14,14}}$;
- **ITER** è la funzione di calcolo complessivo da iterare;
- **TEST** valuta la condizione di uscita dalla funzione **ITER**.

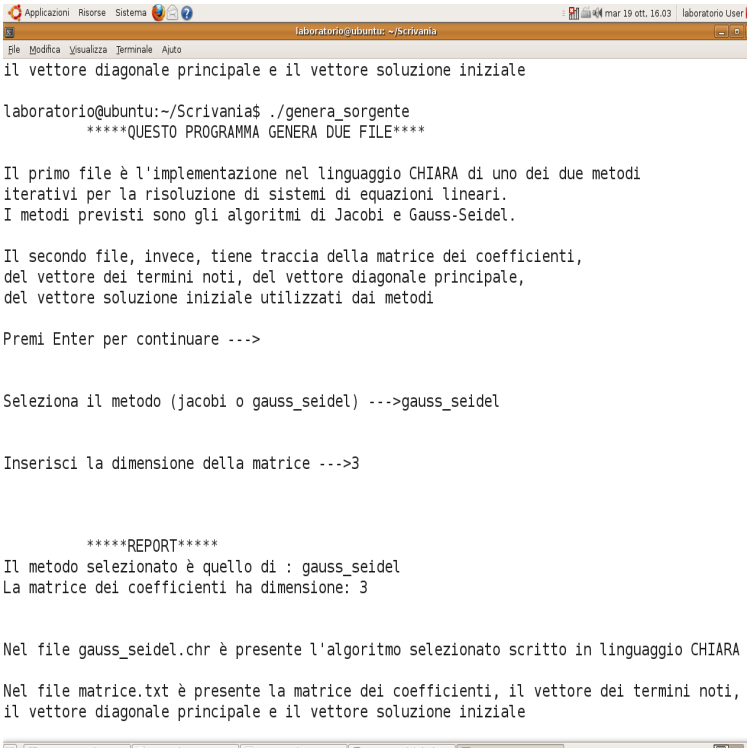

Figura 4: Compilazione ed esecuzione del file genera_sorgente.c .

Inoltre, come valori di input del vettore soluzione iniziale $x^{(0)}$ è stato assunto il vettore dei termini noti b.
Analizzando la Figura 3 del programma 'gauss_seidel.chr', è possibile notare che la definizione di alcune funzioni dipende dalla dimensione del sistema. Tuttavia, se si osserva in dettaglio la struttura di calcolo delle funzioni CALX1, CALX2,.....,CALX14 si nota che le funzioni presentano una regolarità che può essere utilizzata per l'implementazione in Chiara del metodo di Gauss-Seidel per sistemi di qualsiasi dimensione. Al fine di sfruttare questa regolarità, il programma 'genera_sorgente.c', presentato per generare automaticamente il programma in Chiara del metodo di Jacobi, è stato generalizzato per generare anche il sorgente gauss_seidel.chr. La sua esecuzione (comando *./genera_sorgente*) chiede all'utente di selezionare prima quale sorgente generare (jacobi.chr o gauss_seidel.chr) e poi la dimensione della matrice dei coefficienti (Figura 4).

Al termine della sua esecuzione sono creati due file: 'jacobi.chr' o 'gauss_seidel.chr' e 'matrice.txt'.
In Figura 5 è mostrato il file sorgente in Chiara dell'algoritmo di Gauss-Seidel

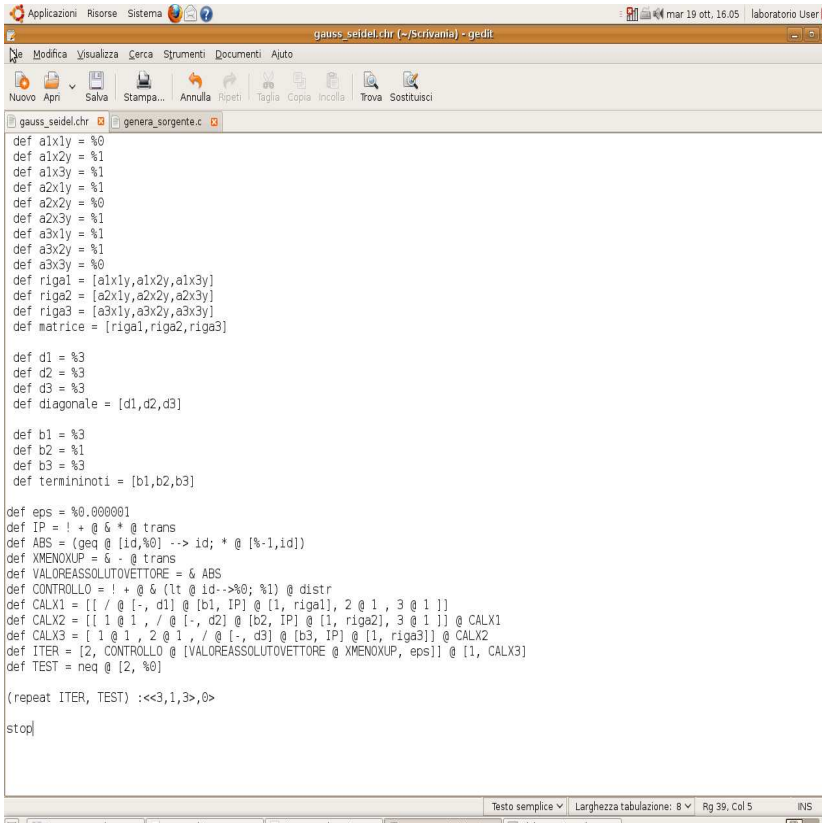

Figura 5: Sorgente gauss_seidel.chr generato da genera_sorgente.c.

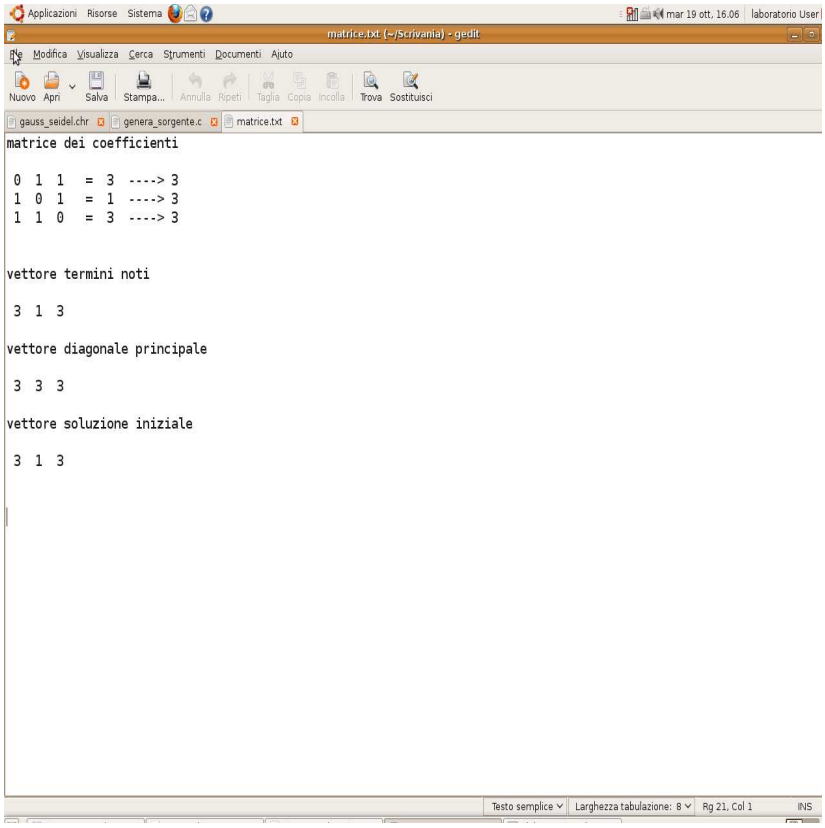

Figura 6: File matrice.txt generato da genera_sorgente.c.

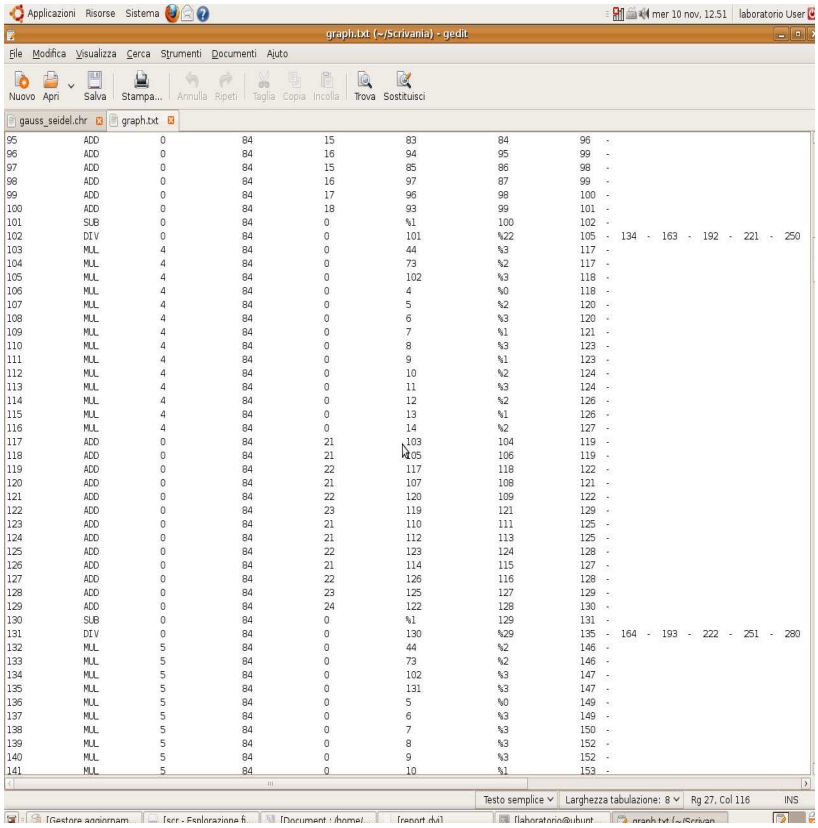

Figura 7: Parte della tabella di configurazione.

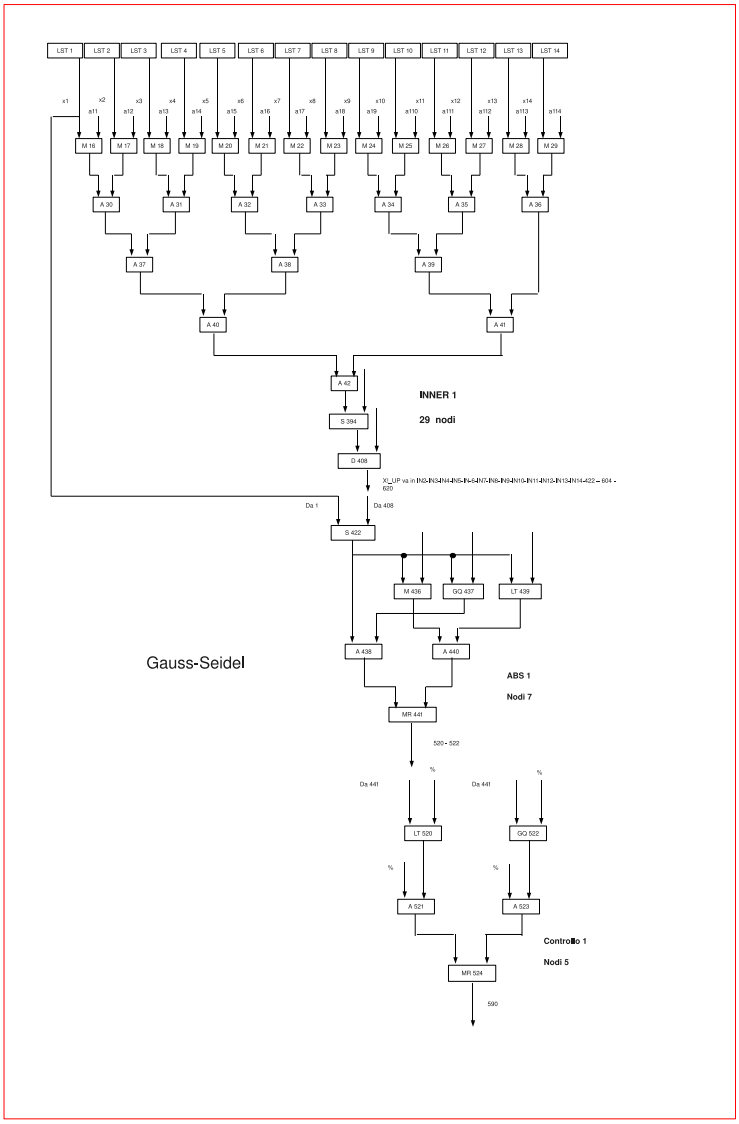

Figura 8: Rappresentazione del dataflow.

per un sistema di tre equazioni in tre incognite generato automaticamente per essere sucessivamente compiltato dal compilatore Chiara.
Il file 'matrice.txt' (Figura 6) contiene le costanti definite in 'gauss_seidel.chr', quali: la matrice dei coefficienti; il vettore diagonale principale; il vettore dei termini noti e il vettore soluzione iniziale.
Tutte le costanti in 'matrice.txt' sono generate dal file 'genera_sorgente.c' in modo casuale ed inoltre la matrice dei coefficienti è creata in modo da garantire la convergenza del metodo (matrice a diagonale dominante in senso stretto). A questo punto il file 'gauss_seidel.chr', generato dal file 'genera_sorgente.c', è pronto per essere compilato dal compilatore del linguaggio Chiara.
Il compilatore, come output, genera la tabella di descrizione del grafo e in Figura 7 è mostrata una sua parte. In Figura 8 è invece mostrata una piccola rappresentazione di tale grafo.

5 Partizionamento del grafo dataflow con Chaco

Dopo aver scritto e compilato il programma 'gauss_seidel.chr', il passo successivo è stato partizionare il grafo dataflow mediante il software Chaco. Prima di presentare la fase del partizionamento, è utile fornire una breve spiegazione del grafo e la sua rappresentazione.
Esso è composto da 634 attori e 929 link come rappresentato nello schema a blocchi di Figura 9. In questa figura, i blocchi più significativi sono stati colorati in modo da spiegarli singolarmente.
Il blocco in rosso è dettagliato in Figura 10 ed è formato da 29 nodi che calcolano il generico passo dell'algoritmo di Gauss-Seidel per l'incognita x_1.

$$x_1^{(k+1)} = \frac{b_1 - \sum_{j<1} a_{1,j} x_j^{(k+1)} - \sum_{j>1} a_{1,j} x_j^{(k)}}{a_{1,1}}$$

Questa struttura è presente 14 volte nella rappresentazione a blocchi di Figura 9, per ogni x_i.
Il blocco in blu è dettagliato in Figura 11 ed è composto da 7 nodi e calcola il valore assoluto della differenza:

$$\mid x_1^{(k+1)} - x_1^{(k)} \mid$$

Anche questa struttura, come la precedente, è presente 14 volte.

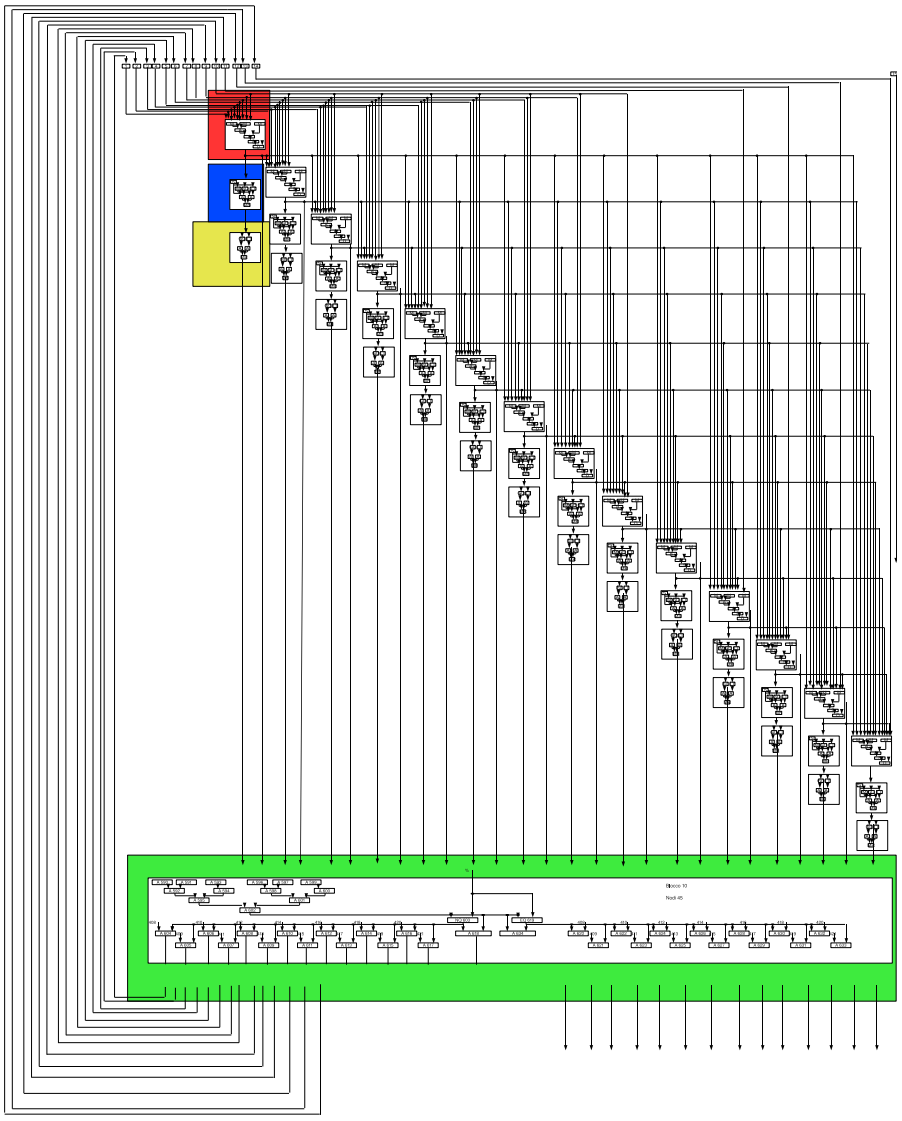

Figura 9: Rappresentaziona a blocchi del dataflow di Gauss-Seidel.

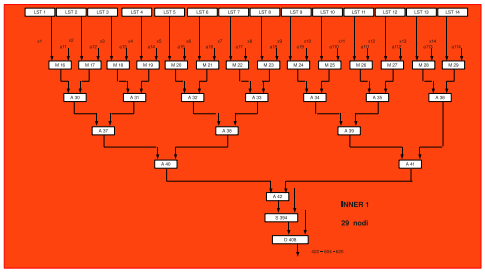

Figura 10: Dettaglio del blocco rosso di Figura 9.

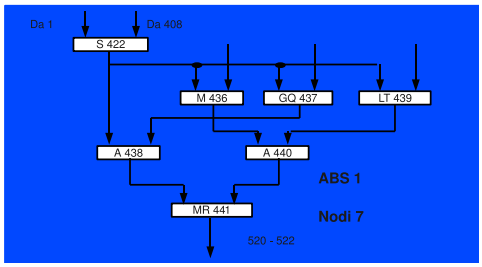

Figura 11: Dettaglio del blocco blu di Figura 9.

Il blocco in giallo è dettagliato in Figura 12 ed è composto da 5 nodi che valutano la condizione di arresto:

$$| x_1^{(k+1)} - x_1^{(k)} | < eps$$

Anche essa è presente 14 volte.
Il blocco in verde è dettagliato in Figura 13 ed è composto da 45 nodi che valutano la condizione di terminazione dell'algoritmo di Gauss-Seidel.

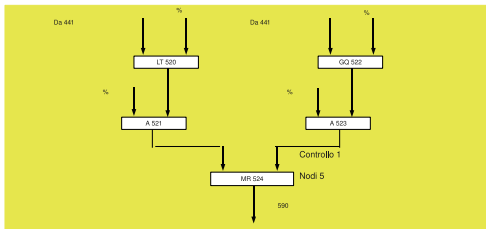

Figura 12: Dettaglio del blocco giallo di Figura 9.

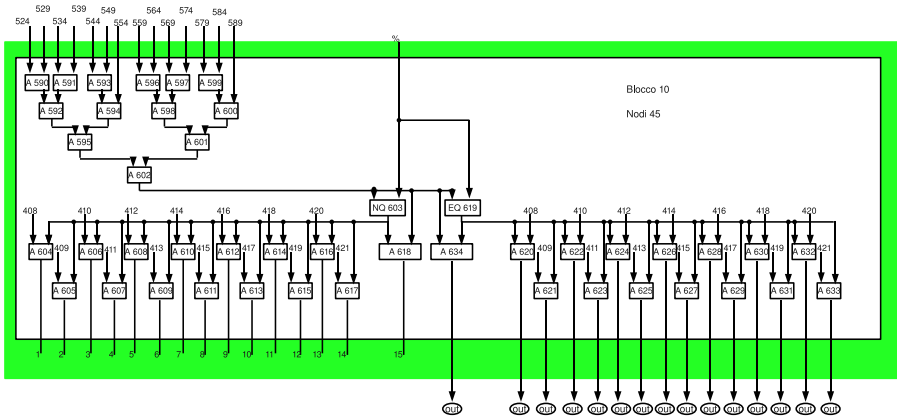

Figura 13: Dettaglio del blocco verde di Figura 9.

Poichè si è ipotizzato di avere un processore dataflow costituito da 128 Unità Funzionali, per eseguire il grafo dataflow del programma, formato da 634 attori, non è possibile creare una corrispondenza 1 ad 1 tra le Unità Funzionali del processore e gli attori del grafo. Di conseguenza, è necessario partizionare il grafo per la sua esecuzione.
Tra le possibili soluzioni di partizionamento si è scelta la strategia mostrata in Figura 14 quale obiettivo da raggiungere. Tale obiettivo, come migliore soluzione, richiede la suddivisione del grafo del programma in 5 partizioni con un numero di attori per partizione inferiore od uguale al numero di Unità Funzionali per processore. Inoltre, questo obiettivo di partizionamento, offre un buon compromesso tra la sequenzialità del metodo di Gauss-Seidel e il parrallelismo messo a disposizione dalle Unità Funzionali del processore.

Per rendere automatico il processo di partizionamento mediante Chaco, partendo dall'informazione del numero di Unità Funzionali per processore, è stato utilizzato, tra i vari algoritmi offerti da Chaco, quello basato sul metodo globale **KL-multilivello** [4] perchè è ritenuto, dagli stessi sviluppatori di Chaco, l'unico che produce sempre partizioni di alta qualità. Gli altri algoritmi implementati in Chaco sono: **Semplice**, **Inerziale**, **Spettrale**, **Random**, **Scattered** e **Kernighan-Lin**.

La Figura 15 mostra le opzioni offerte da Chaco per selezionare le scelte che si desiderano applicare al partizionamento. In Figura 15 è possibile selezionare:

1. il nome del file di input con estensione '.graph' che contiene le informazioni relative al grafo dataflow da partizionare. Esso è generato dal

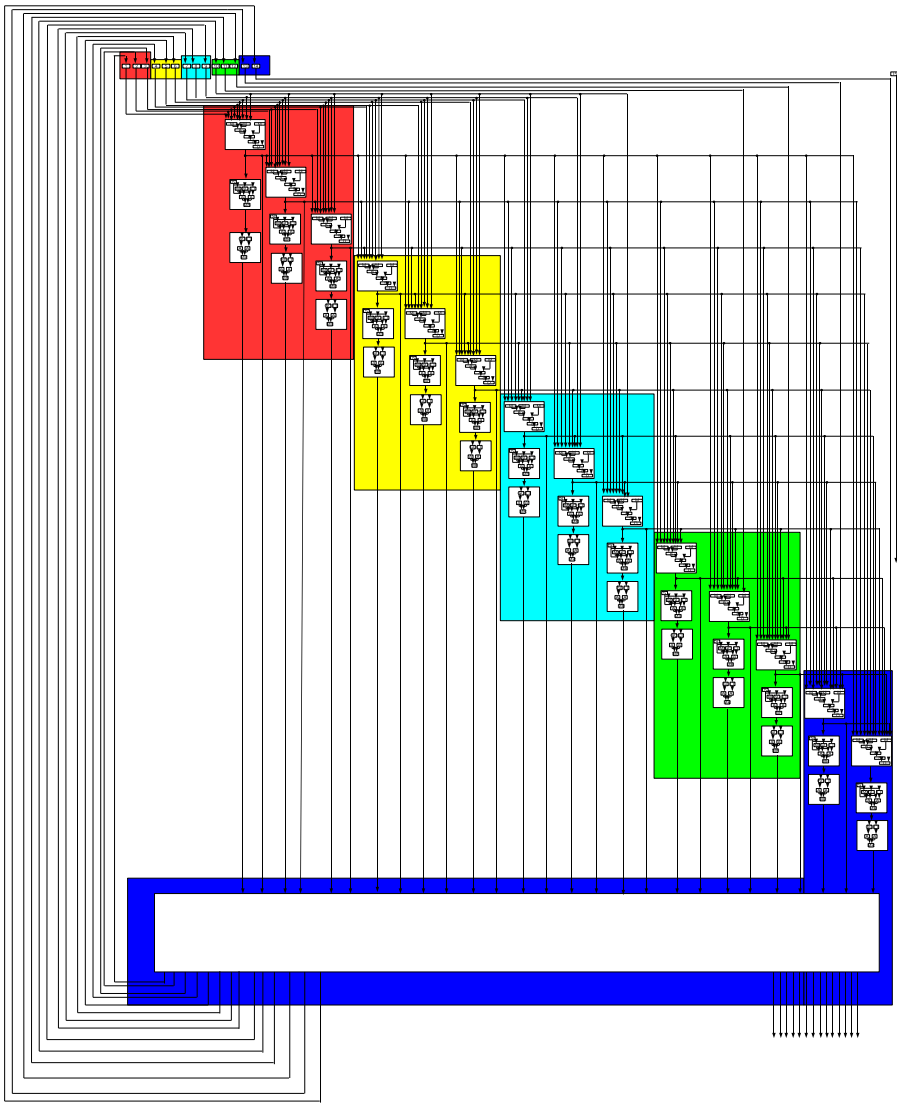

Figura 14: Obiettivo di partizionamento da raggiungere.

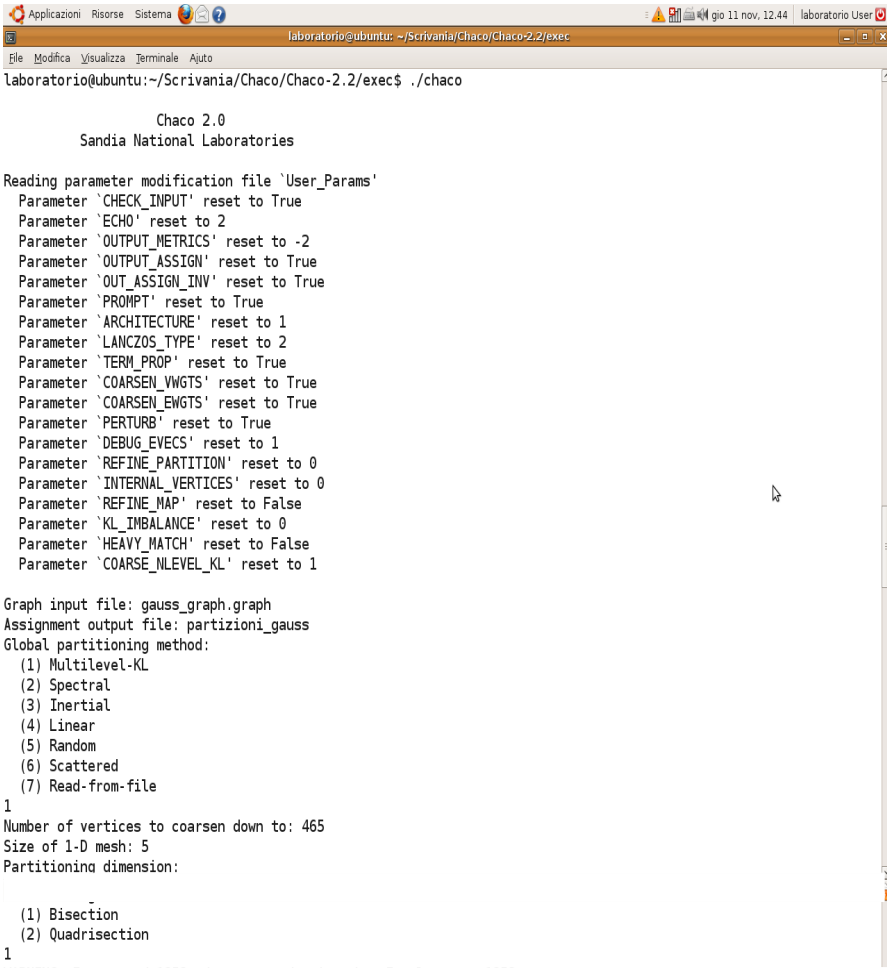

Figura 15: Esecuzione di Chaco per il file 'gauss_graph.graph'.

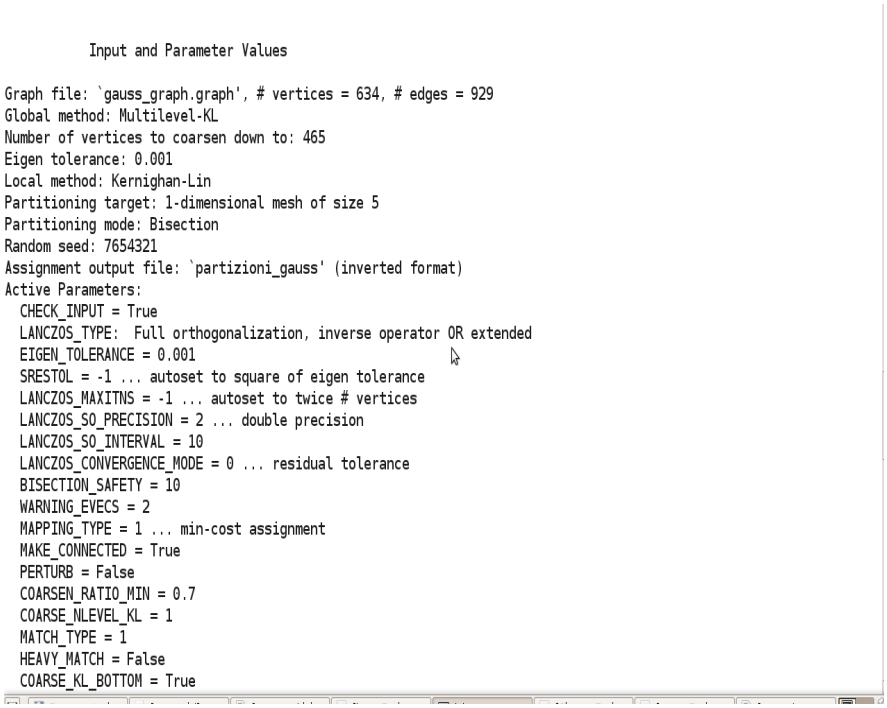

Figura 16: Esecuzione di Chaco per il file 'gauss_graph.graph'.

file della tabella di configurazione prodotta dal compilatore Chiara, mediante una sua trasformazione eseguita dal programma 'gdltograph.c' ed è in formato testo. Le informazioni contenute in questo file sono così organizzate (Figura 17):

- La prima riga contiene il numero di nodi, il numero di archi e la direttiva per Chaco sulla topologia del grafo. La direttiva è costituita da un codice binario a tre cifre con i seguenti significati:

  bit meno significativo
  - valore '0': senza peso associato agli archi.
  - valore '1': peso associato agli archi.

  bit intermedio
  - valore '0': senza peso associato ai nodi.
  - valore '1': peso associato ai nodi.

  bit più significativo:
  - valore '0': per ognuna delle righe successive Chaco assegna automaticamente in sequenza progressiva il valore dei nodi di partenza del grafo.
  - valore '1': per ognuna delle righe successive il valore dei nodi di partenza del grafo sono assegnati manualmente.
- Le righe successive rappresentano le liste di adiacenza associate ai nodi di partenza, costituite dalle coppie 'nodo' 'peso arco' se il bit meno significativo è '1'.

2. Nome del file che contiene le partizioni calcolate (partizioni_gauss).
3. Metodo globale scelto (Multilevel-KL).
4. Numero di nodi del grafo grossolano (Number of vertices to coarsen down).
5. Numero di partizioni (Size of 1-D mesh).
6. Tecniche di partizionamento ad ogni passo (Bisection).

Dopo l'inserimento dei parametri richiesti in Figura 15, Chaco genera un riepilogo delle scelte effettuate corredandole con i valori dei parametri assegnati automaticamente e rappresentati in Figura 16.

Al termine dell'esecuzione, Chaco stampa a video una serie di informazioni

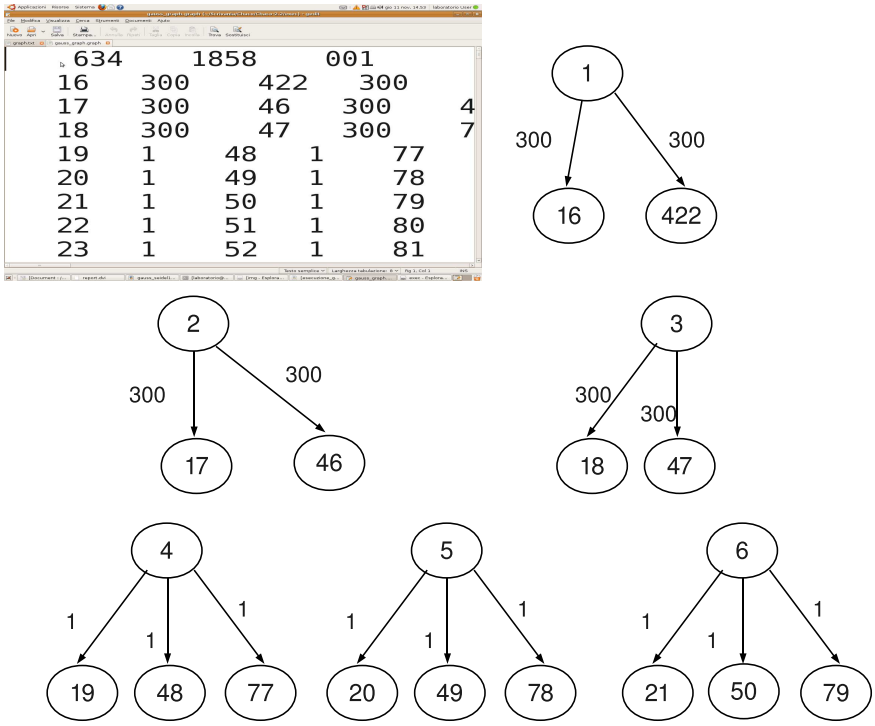

Figura 17: Parte di file 'gauss_graph.graph' contenente il grafo da partizionare.

relative al processo di partizionamento fornendo anche il tempo impiegato dal metodo **KL-multilivello** per raggiungere i risultati finali (Figura 18). Le cinque partizioni prodotte da Chaco sono visualizzate in Figura 19. Il file è composto da cinque colonne, una per ogni partizione generata, e in ognuna di esse è riportato prima il numero complessivo di nodi (attori) che la compongono e poi il numero associato a ciascun nodo (attore) della partizione. Analizzando la composizione delle partizioni ottenute con quelle poste come obiettivo in Figura 14, si osserva che l'obiettivo è stato centrato in modo più che soddisfacente. Infatti, su un totale di 634 nodi solo il 15% di questi non si trovano nelle partizioni desiderate a fronte della generalità delle scelte adottate prima e durante il processo di partizionamento. Infatti, dato che il risultato di un partizionamento, nel caso di grafi con funzione peso associata ai link, dipende fortemente dal settaggio del peso degli archi, una prima scelta effettuata è stata proprio la migliore associazione dei valori dei pesi agli archi in modo da fornire delle direttive a Chaco su come partizionare il grafo. A tale proposito si è assegnato il valore '1', costo di comunicazione basso tra

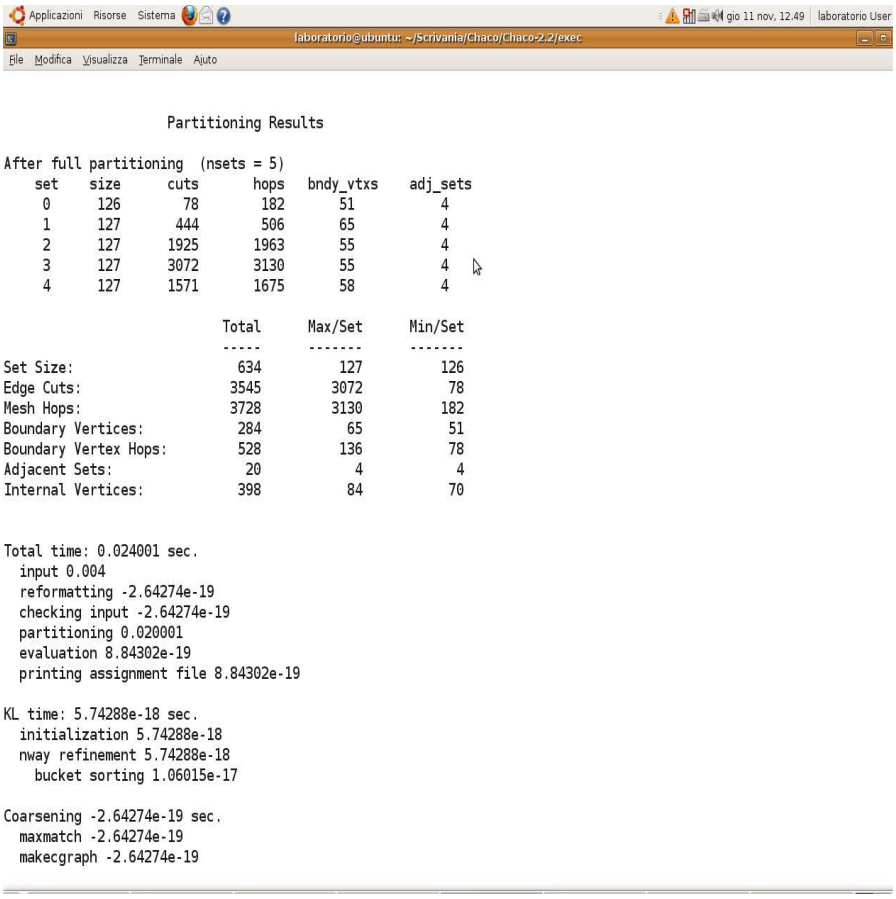

Figura 18: Informazioni prodotte da Chaco dopo l'esecuzione del file 'gauss_graph.graph'.

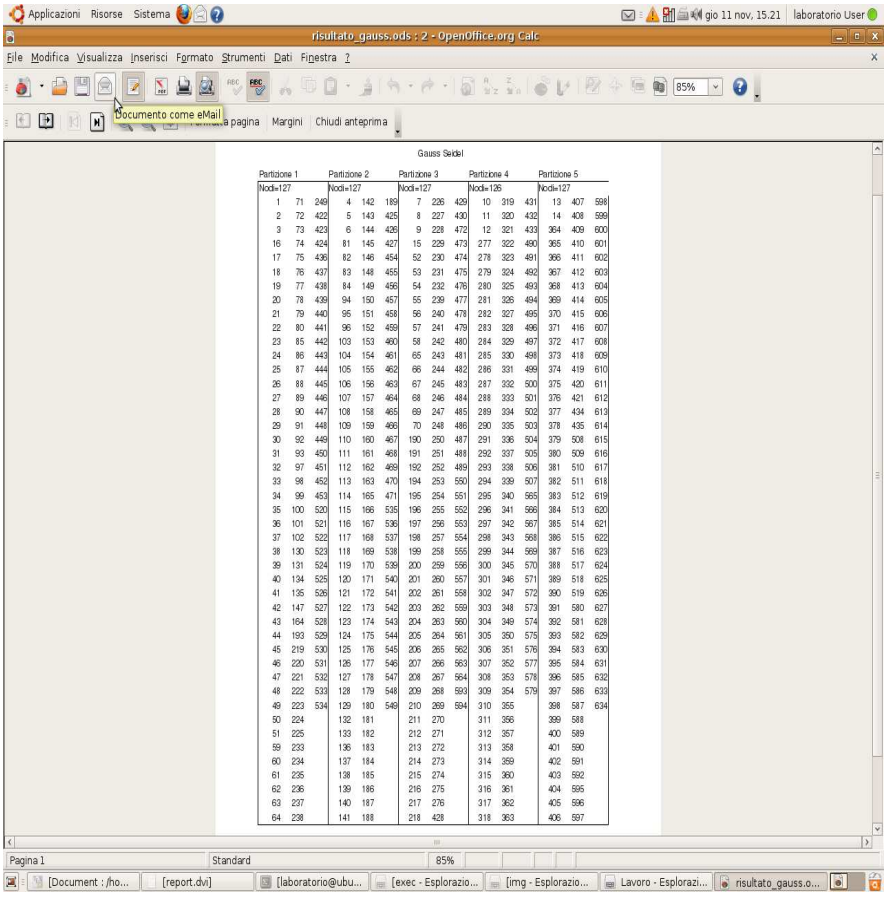

Figura 19: Partizioni prodotte da Chaco.

nodi, agli archi che dovrebbero connettere attori di partizioni diverse e il valore di '300'(quasi la metà del numero di nodi), costo di comunicazione elevato tra nodi, agli archi che dovrebbero connettere nodi appartenenti alla stessa partizione.
Un altro aspetto che si è cercato di rendere generale, è stato l'individuazione del valore appropriato per il numero di vertici del grafo grossolano (Number of vertices to coarsen down), richiesto da Chaco, prima dell'esecuzione del metodo **KL-multilivello**. Infatti, questo valore ha la caratteristica di modificare, per uno stesso grafo in ingresso, il risultato di un partizionamento. Per questo partizionamento è stato utilizzato il valore '465'(metà del numero di archi del grafo).
L'ultima scelta che si è operata è stato l'impiego dei parametri di default offerti da Chaco per l'esecuzione degli algoritmi implementati. Infatti, se da un lato è possibile modificare opportunamente questi parametri per ottenere partizioni che corrispondono al 100% a quelle desiderate in Figura 14, dall'altro lato, nel caso specifico di Gauss-Seidel, se si aumenta o diminuisce la dimensione del sistema di equazioni lineari, tale modifica di parametri produrrà risultati diversi. In pratica, un settaggio appropriato per un problema di una data dimensione non può essere esteso allo stesso problema di differente dimensione.
Infine, in Figura 20 è riportato il tempo di esecuzione:

- di ciascun blocco di Figura 9;
- di una sequenza di blocchi (colore rosso più colore blu più colore giallo di Figura 9);
- complessivo dell'algoritmo.

Supponendo T_op (Figura 20) come tempo di esecuzione di ogni attore del grafo, allora il tempo di esecuzione per un generico passo dell'algoritmo di Gauss-Seidel è di 109 T_op e, inoltre, il numero di passi necessari per ottenere la convergenza del metodo è 12.
La Tabella 1 riporta i tempi di esecuzione degli algoritmi di Jacobi e Gauss-Seidel per uno stesso sistema fornito in ingresso.
Analizzando questi risultati è possibile affermare che l'algoritmo di Gauss-Seidel produce in minor tempo le soluzioni del sistema rispetto a Jacobi nonostante la natura parallela di quest'ultimo.

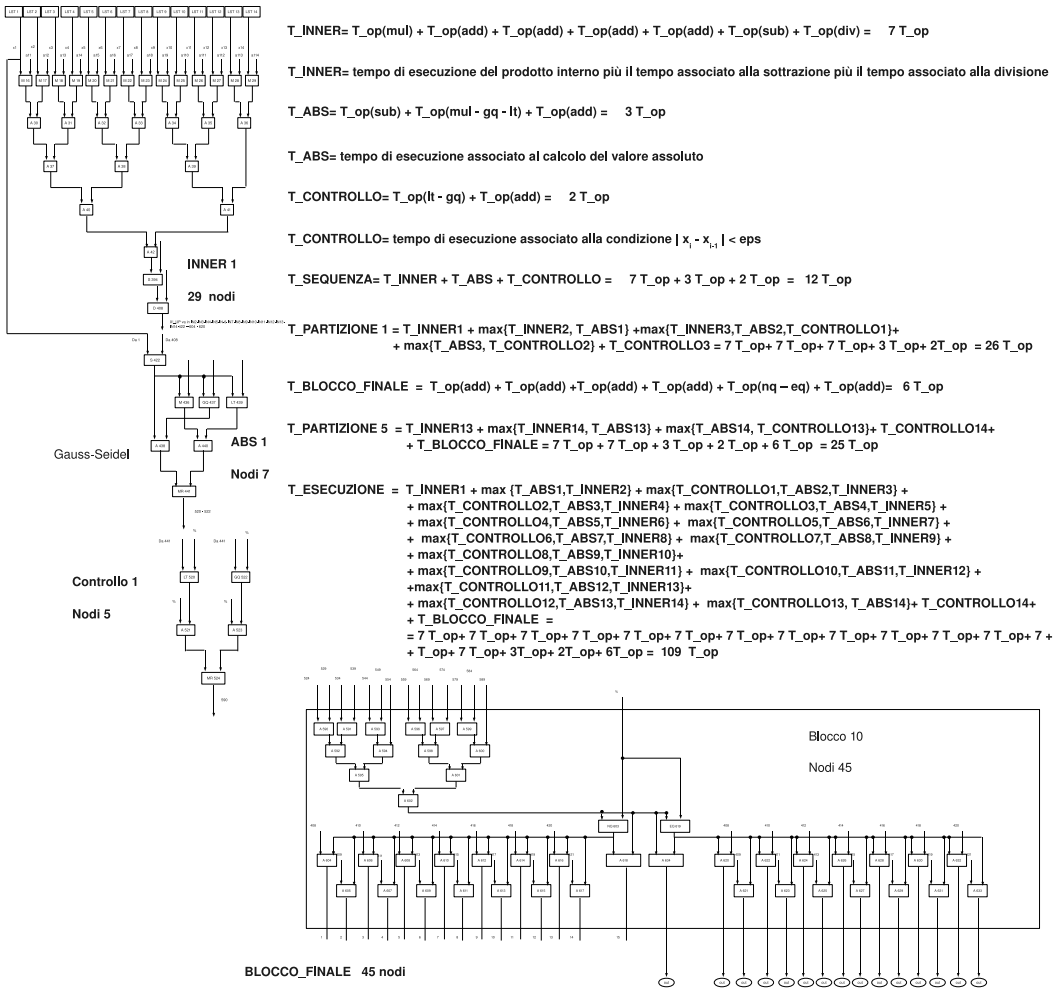

Figura 20: Tempo di esecuzione.

	Jacobi	**Gauss-Seidel**
Tempo esecuzione singolo passo	18 T_op	109 T_op
Passi per convergere	397	12
Tempo esecuzione algoritmo	7146 T_op	1308 T_op

Tabella 1: Tabella confronto.

6 Conclusioni

In questo lavoro sono state analizzate le fasi di generazione e partizionamento del grafo dataflow rappresentante il metodo di Gauss-Seidel. A tal proposito, è stato scritto il programma 'gauss_seidel.chr' che implementa l'algoritmo di Gauss-Seidel per un sistema di quattordici equazioni in quattordici incognite. Inoltre, la natura generale delle funzioni che definiscono l'algoritmo e la possibililità di un'analisi futura del metodo per sistemi di dimensione maggiore hanno suggerito la modifica, attraverso una selezione da menù, del file 'genera_sorgente.c' in modo da generare, oltre al sorgente di Jacobi [3], anche quello di Gauss-Seidel.
Dopo aver scritto e compilato il programma 'gauss_seidel.chr' il passo successivo è stato il partizionamento del grafo mediante il software Chaco. Poichè il partizionamento di un grafo è un problema NP-completo, in questo lavoro si sono prima fissati degli obiettivi di partizionamento da raggiungere e poi si è individuata una metodica automatizzabile per ottenerli sfruttando le direttive di carattere generale di Chaco, il cui risultato finale presenta una soluzione molto prossima a quella desiderata. Sebbene sia possibile, settare ad hoc manualmente i parametri di Chaco per ottenere delle partizioni identiche a quelle fissate negli obiettivi, si è osservato che lo stesso settaggio non sempre fornisce i risultati attesi se la dimensione del sistema di equazioni lineari varia.
Un altro problema riscontrato durante la fase di partizionamento riguarda la topologia di grafo da partizionare. Infatti, Chaco non ammette come ingresso grafi orientati e questo rappresenta comunque un ostacolo per il raggiungimento dei nostri obiettivi, data la natura dei grafi dataflow.
Questo lavoro si conclude con un confronto tra i tempi di esecuzione degli algoritmi di Jacobi e Gauss-Seidel dimostrando che quest'ultimo, quando entrambi convergono, converge molto più velocemente.

Riferimenti bibliografici

[1] L. Verdoscia, M. Danelutto, and R. Esposito. CODACS prototype: CHIARA language and its compiler. In *Proceedings of the First International Workshop on Embedded Computing*, Tokyo University of Technology, Hachioji, Tokyo, Japan, March 23–26, 2004. IEEE Computer Society Press.

[2] B. Hendrickson and R. Leland. The CHACO user's guide – version 2.0. Technical Report SAND94–2692, Sandia National Laboratories, 1994.

[3] G. Gallo and L. Verdoscia. Partizionamento del grafo dataflow prodotto dal compilatore chiara per la risoluzione di un sistema di equazioni lineari con il metodo di jacobi. Technical Report RT-ICAR-NA-01-10, Istituto di Calcolo e Reti ad Alte Prestazioni(ICAR-CNR), 29 Settembre 2010.

[4] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 28, New York, NY, USA, 1995. ACM.