In riferimento al seguente frammento di pseudocodice:

```
for i=N-1, i>0, i--
        { S=0;
          for j=i+1, j<N, j++
                { S=(S+U[i,j]*X[j]) / U[i,i]; }
          X[i]=Y[i] – S;
        }
```

si forniscano:

1) compilazione in assembler D-RISC secondo le regole di compilazione standard;
2) individuazione del working set e calcolo del numero di cache faults per una cache con σ=8 e relativamente ad una esecuzione del ciclo più esterno;
3) prestazioni (tempo di servizio) su un processore D-RISC pipeline con unità EU slave pipeline a 4 stadi per l'esecuzione delle moltiplicazioni e divisioni intere;
4) cause di degrado delle prestazioni (solo nel caso del ciclo più interno);
5) eventuali ottimizzazioni (quantificando il guadagno in termini di tempo di servizio).

Compilazione in Assembler D-RISC (in corsivo neretto il loop centrale) la matrice U è memorizzata per righe:

```
                SUB Rn, #1, Ri // inizializzazione variabile di iterazione i
        LOOPi: ADD R0, R0, Rs // azzeramento S
                ADD Ri, #1, Rj // inizializzazione variabile di iterazione j
                MUL Ri, Rn, Rbasei // calcolo invariante: base riga i rispetto alla base di
                                                        //tutta la matrice U
1       LOOPj: ADD Rbasei, Rj, Rind // offset [i,j]
2              LOAD Rbaseu, Rind, R1 // U[i][j]
3              LOAD Rbasex, Rj, R2 // X[j]
4              MUL R1, R2, R3 // U[i][j] * X[j]
5              ADD R3, RS, RS // S + U[i][j] * X[j]
6              ADD Rbasei, Ri, Rind // offset [i][i]
7              LOAD Rbaseu, Rind, Ruii // U[i][i]
8              DIV Rs, Ruii, Rs // (S + U[i][j] * X[j]) / U[i][i]
9              INC Rj // fine ciclo j, incremento variabile di iterazione
10             IF< Rj, Rn, LOOPj // itero se minore di N
                LOAD Rbasey, Ri, Ryi // Y[i]
                SUB Ryi, Rs, Rtemp // Y[i] - S
                STORE Rbasex, Ri, Rtemp // X[i]
                DEC Ri // fine ciclo i, decrementa variabile di iterazione
                IF>0 Ri, LOOPi // itero se maggiore di 0
                END // fine lavori
```

Working set e numero di cache faults:

Analizzando lo pseudo codice, vediamo che all'iterazione i accediamo l'ultima parte della riga i-esima della matrice U, il valore di tale riga sulla diagonale (primo elemento dell'ultima parte della riga), l'ultima parte del vettore X nonché la posizione i dei vettori X e Y. Abbiamo località su U, X e Y e riuso su X. Per il codice abbiamo località e riuso, come al solito per codice che contiene cicli. Il working set conterrà dunque una linea con dati di U (quella che contiene un pezzo dell'ultima parte della riga i-esima di U), tutto X (anche se via via la parte iniziale del vettore scompare dal working set) e una linea per Y, oltre al codice. Quindi avremo **3** faults per il **codice** (sono **18 istruzioni**) più i faults per ***U***: **C((N-i+1)/8)**, per ***X*** e ***Y***: **C(N/8)** (dove C(z) è la funzione "ceiling" cioè l'intero superiore di z).

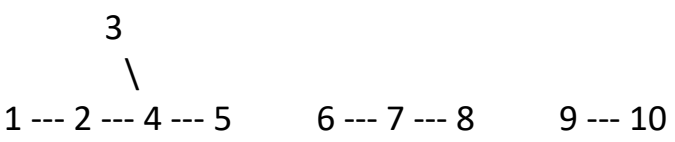
Valutazione del tempo di servizio:

Il tempo di servizio si deduce dal seguente schema:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
ADD	LOAD		LOAD	MUL	ADD						ADD	LOAD			DIV	INC	IF			ADD		
	ADD	LOAD	LOAD	LOAD	MUL	ADD						ADD	LOAD	LOAD		DIV	INC	IF	IF		ADD	
				LOAD	LOAD										LOAD							
		ADD			LOAD	LOAD	MUL	ADD	ADD	ADD	ADD	ADD	ADD			LOAD	DIV	INC				ADD
								MUL	MUL	MUL	MUL							DIV	DIV	DIV	DIV	

Quindi, il tempo di completamento di una iterazione è pari a 20t. Siccome il tempo ideale è di 10 t (abbiamo 10 istruzioni), risulterà un tempo di servizio doppio rispetto a quello ideale.

Cause di degrado delle prestazioni:
Consideriamo ancora la sola esecuzione del ciclo più interno. Abbiamo una dipendenza IU-EU fra la prima ADD e la LOAD successiva (**istruzioni 1 e 2**), fra la terza ADD e la LOAD successiva (**istruzioni 6 e 7**) e fra la INC e la IF< (**istruzioni 9 e 10**). Inoltre, abbiamo una dipendenza EU-EU fra la MUL e la ADD successiva (**istruzioni 4 e 5**). Il grafo data-flow (senza tenere conto delle precedenze molto lontane-ad esempio, la 1 e la 2 devono precedere la 6 perché scrivono o leggono lo stesso registro, Rind, così come la 1 e la 3 devono precedere la 9, a causa del registro Rj) diviene:



Ottimizzazioni:
Il ciclo più interno si può ottimizzare. La seconda LOAD (istruzione 3) può essere anticipata fra la prima ADD e la successiva LOAD, cosicché la prima bolla legata alla dipendenza IU-EU sia di fatto eliminata. Il caricamento di U[i][i] può anche essere anticipato per allontanare la ADD dalla MUL che induce la dipendenza EU-EU (spostamento della istruzione 5 come terzultima istruzione). Dunque, le due istruzioni (calcolo dell'indice e caricamento della posizione U[i][i] possono essere interposte fra la MUL e la ADD. Fra queste due istruzioni (la ADD induce una dipendenza sulla LOAD) possiamo interporre l'incremento della variabile di iterazione. Infine, la DIV può essere utilizzata nel delay slot della IF di fine iterazione. Il codice così ottimizzato diventa dunque (a sinistra i numeri delle istruzioni come nel programma precedente):

```
1     LOOPj: ADD Rbasei, Rj, Rind // offset [i,j]
3            LOAD Rbasex, Rj, R2 // X[j]
2            LOAD Rbaseu, Rind, R1 // U[i][j]
4            MUL R1, R2, R3 // U[i][j] * X[j]
6            ADD Rbasei, Ri, Rind // offset [i][i]
9            INC Rj // fine ciclo j, incremento variabile di iterazione
7            LOAD Rbaseu, Rind, Ruii // U[i][i]
5            ADD R3, RS, RS // S + U[i][j] * X[j]
10           IF< Rj, Rn, LOOPj, delayed // itero se minore di N
8            DIV Rs, Ruii, Rs // ( S + U[i][j] * X[j]) / U[i][i]
```

Guadagno di tempo:

Possiamo vedere dalla simulazione come l'esecuzione del codice arrivi ad un tempo di servizio **quasi ottimale** (***11t/10t***), se teniamo conto della possibilità di realizzare la comunicazione fra unità in modo asincrono con la quantità di posizioni buffer necessarie:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ADD	LOAD	LOAD	MUL	ADD	INC	LOAD	ADD	IF<	DIV		ADD	LOAD	LOAD	MUL			
	ADD	LOAD	LOAD	MUL	ADD	INC	LOAD	LOAD	ADD	IF<	DIV	ADD	LOAD	LOAD	MUL		
			LOAD	LOAD					LOAD					LOAD	LOAD		
		ADD		LOAD	LOAD	MUL	ADD	INC		LOAD	ADD	DIV	ADD		LOAD	LOAD	MUL
							MUL	MUL	MUL	MUL			DIV	DIV	DIV	DIV	