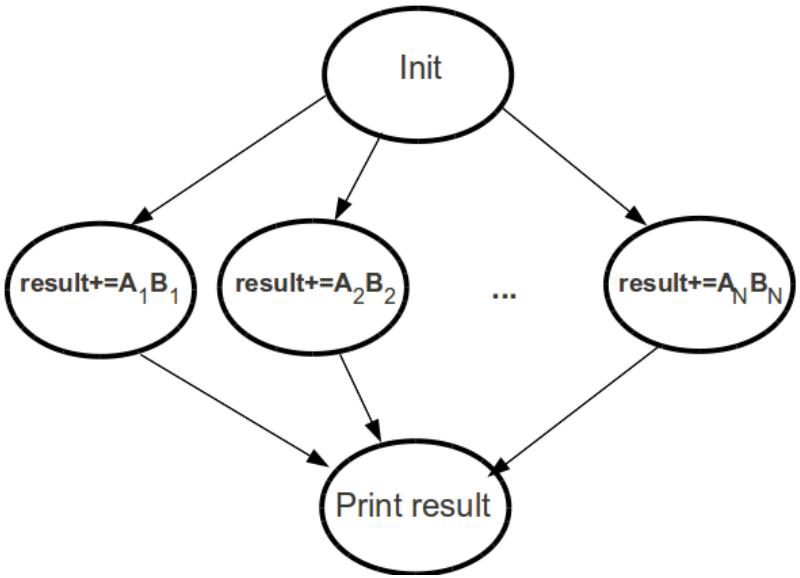
Casi di implementazione di grafi di precedenze

Prodotto tra vettori

- Si tratta di calcolare $A \cdot B^T$ dove A e B sono vettori monodimensionali.
- Il risultato é quindi: $result = A_1 \cdot B_1 + A_2 \cdot B_2 + \cdots + A_N \cdot B_N$
- Il diagramma delle precedenze é dunque:



Implementazione in Java

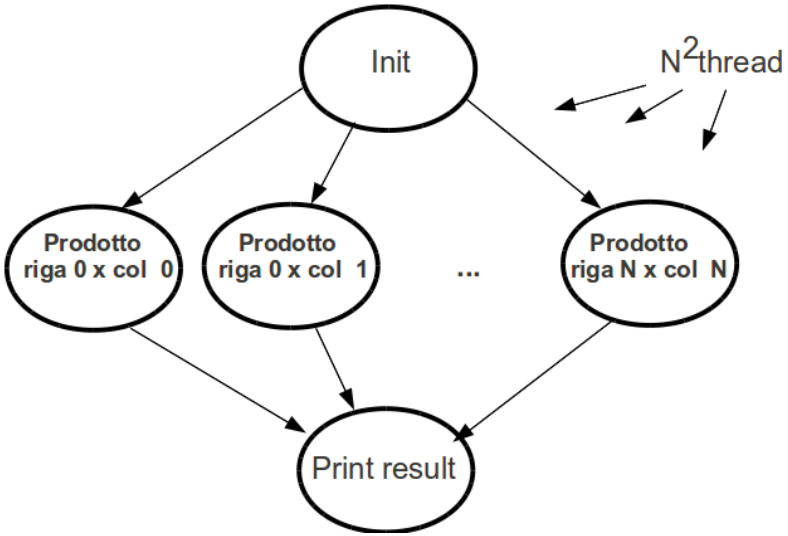
```
class data{ //prodotto scalare tra due vettori
    float a[]=new float[10];
    float b[]=new float[10];
    float result;
}
class Prod extends Thread{ //prodotto del singolo elemento
    data d;  float a; float b;
    public Prod(data d, float a, float b){this.d=d; this.a=a; this.b=b;}
    public void run(){
       d.result+=a*b;
    }
}
public class VetbyVet {
    private static data d = new data();
    public static void main(String[] args) {
       Prod t[]=new Prod[10];
       for(int i=0; i<10; i++){//istanzia i prodotti e fa partire I thread
            t[i]= new Prod(d, d.a[i], d.b[i]);
            t[i].start();
       }
       for(int i=0; i<10; i++){
          try{t[i].join();} catch (Exception e) {};
       }
       System.out.println("risultato = "+d.result);
     }
}
```

Variante: concetto cobegin/coend

```
class data{
    float a[]=new float[10];
    float b[]=new float[10];
    float result;
    Prod t[]=new Prod[10]; //<-- condivido anche l'array dei thread!
}
class Prod extends Thread{ //prodotto del singolo elemento
    data d;  float a; float b;
    public Prod(data d, float a, float b){this.d=d; this.a=a; this.b=b;}
    public void run(){
        d.result+=a*b;
    }
}
public class cobegcoend {
    private static data d = new data();
    public static void Cobegin(int N){
          for(int i=0; i<N; i++)d.t[i].start();
        }
    public static void Coend(int N){
          for(int i=0; i<N; i++)try{d.t[i].join();} catch (Exception e) {};
        }
    public static void main(String[] args) {
        for(int i=0; i<10; i++){//istanzia i thread 'Prod'
             d.t[i]= new Prod(d, d.a[i], d.b[i]);
        }
    Cobegin(10);
    Coend(10);
    System.out.println("risultato = "+d.result);
}
```

Prodotto tra matrici

- Si tratta di calcolare $A \cdot B$ dove A e B sono matrici $M \times M$.
- Il diagramma delle precedenze é dunque:

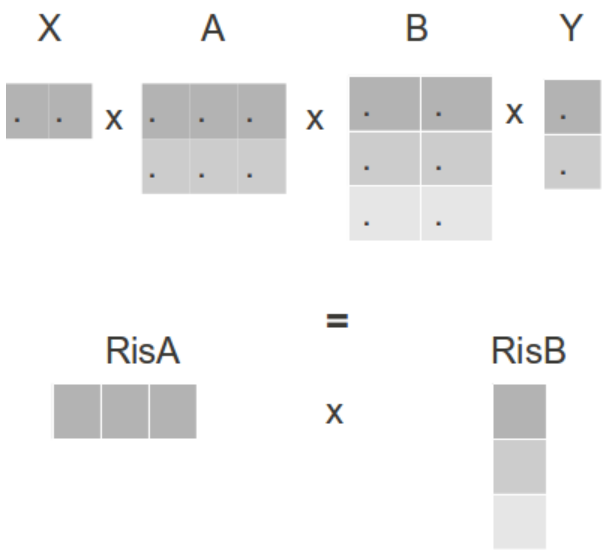


Implementazione in Java

```
public class pm {  //prodottomatriciale
    static public final int M=1000;
    public static Thread t[][]=new Thread[M][M]; //matrice di puntatori a MxM Threads
    public static int a[][]=new int[M][M], b[][]=new int[M][M], c[][]=new int[M][M];
    public static int i,j,l,m;
    public static void init(){ //inizializza le matrici
        for(int l=0;l<M;l++)  for(int m=0;m<M;m++){
                        c[l][m] = 0;
                        a[l][m] = (int)Math.round(100.*Math.random());
                        b[l][m] = (int)Math.round(100.*Math.random());
             }
    }
    public static void main(String[] arg) throws Exception {
      init();
      for(i=0;i<M;i++) for(j=0;j<M;j++)
            {t[i][j]=new P(i,j); t[i][j].start();}
      for(l=0;l<M;l++) for(m=0;m<M;m++)
            t[l][m].join();
    }
}
class P extends Thread{ //thread Prodotto
    int i,j,k;
    public P(int i, int j){this.i=i; this.j=j;}
    public void run(){
        for(k=0; k<pm.M; k++)
            pm.c[i][j] += pm.a[i][k]*pm.b[k][j];
    }
}
```

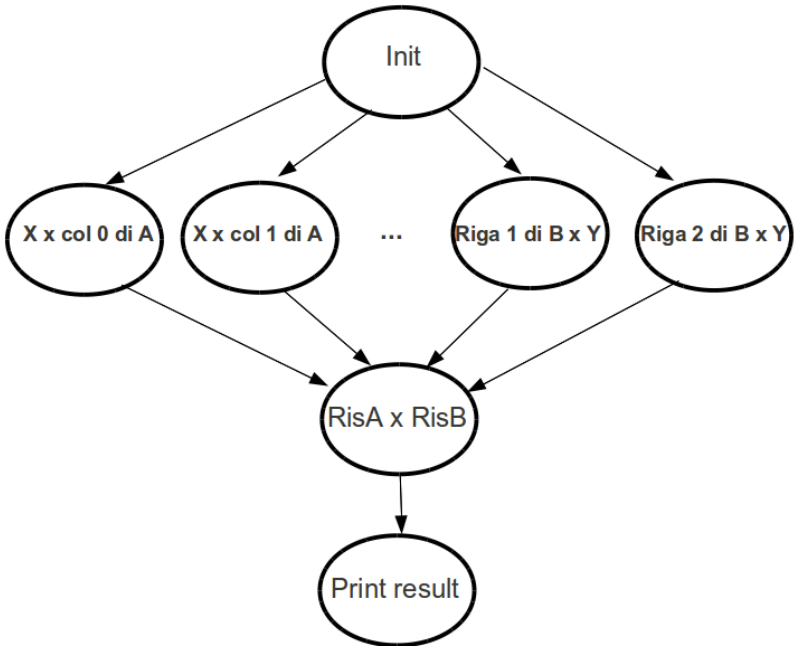
Espressione algebra lineare

- Si tratta di calcolare la seguente espressione di algebra lineare: $X \times A \times B \times Y^T$ dove X é un vettore 1×2, A una matrice 2×3, B é una matrice 3×2, Y é un vettore 2×1. Il risultato evidentemente si ottiene con il seguente calcolo:



Espressione algebra lineare

- Il diagramma delle precedenze é dunque:



Implementazione in Java

```
public class linear{//una operazione di algebra lineare
    private static data d= new data();
    public static void main(String[] args) {
        RisAbyRisB z;
        XbyA t1[]=new XbyA[3];
        BbyY t2[]=new BbyY[3];
        for(int i=0; i<3; i++){//istanzia e fa partire i prodotti X x A
            t1[i]= new XbyA(d,i);
            t1[i].start();
        }
        for(int i=0; i<3; i++){//istanzia e fa partire i prodotti B x Y
           t2[i]= new BbyY(d,i);
           t2[i].start();
        }
        for(int i=0; i<3; i++) try{t1[i].join();} catch (Exception e) {};
        for(int i=0; i<3; i++) try{t2[i].join();} catch (Exception e) {};
        z=new RisAbyRisB(d);//quando sono finiti i prodotti in concorrenza faccio risA x risB
        z.start();
    }
}                          La classe data :
class data{
    float X[]=new float[2];
    float Y[]=new float[2];
    float A[][]=new float[2][3];
    float B[][]=new float[3][2];
    float risA[]=new float[3];
    float risB[]=new float[3];
    float result;
}
```

Implementazione in Java (Cont.)

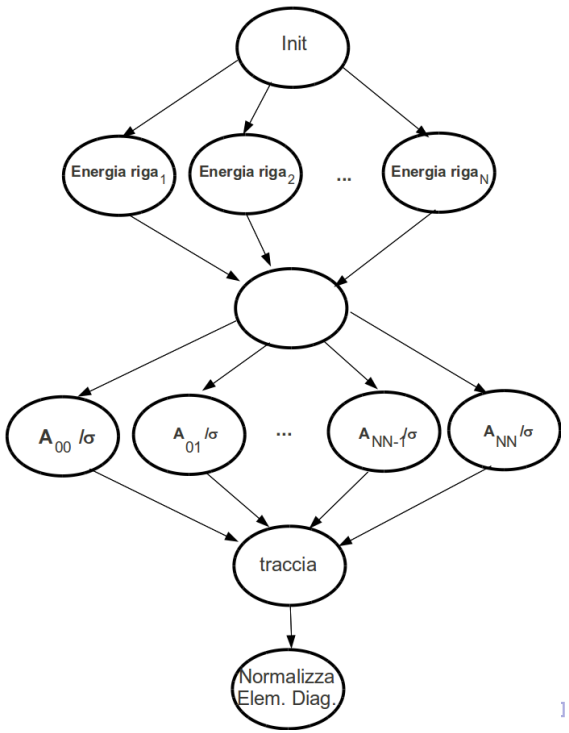
```
class XbyA extends Thread{ //prodotto del vettore riga X con la colonna ind di A
    data d; int ind; float s; int i;
    public XbyA(data d, int indxi){this.d=d; this.ind=indxi;}
    public void run(){
       s=0;
       for(i=0;i<2;i++)
       s+=d.X[i]*d.A[i][ind];  d.risA[ind]=s;
       }
}
class BbyY extends Thread{ //prodotto della riga ind di B con il vettore colonna Y
    data d; int ind; float s; int i;
    public BbyY(data d, int indxi){this.d=d; this.ind=indxi;}
    public void run(){
       s=0;
       for(i=0;i<2;i++)  s+=d.B[ind][i]*d.Y[i];
       d.risB[ind]=s;
    }
}
class RisAbyRisB extends Thread{
   data d; int i; float s;
   public RisAbyRisB(data d){this.d=d;}
   public void run(){
      s=0;
      for(i=0;i<3;i++) s+=d.risA[i]*d.risB[i];
      d.result=s;
   }
}
```

Traduzione di un programma sequenziale

- Si tratta di convertire in calcolo concorrente il seguente algoritmo sequenziale:

```
for(i=0;i<N;i++){
   s=0; for(j=0;j<M;j++) s += a[i][j]*a[i][j];
   s=sqrt(s);
   for(j=0;j<N;j++) a[i][j] = a[i][j]/s;
   traccia += a[i][i];
}
```

La matrice viene normalizzata secondo l'energia delle righe. Poi, calcola traccia e normalizzazione degli elementi diagonali. Diagramma delle precedenze:



Implementazione in Java

```
public class normalized {
    static public final int M=1000;
    private static data d= new data();
    public static void main(String[] args) throws Exception{
    T t[][]=new T[M][M];
    RowEner r[]=new RowEner[M];
    for(int i=0; i<M; i++){ //per ogni riga
      r[i]= new RowEner(d,i);
      r[i].start();
    }
    for(int i=0; i<M; i++) r[i].join();
    for(i=0;i<M;i++) for(j=0;j<M;j++){
          t[i][j]= new T(d,i,j);   t[i][j].start();
    }
    for(int i=0; i<M; i++) t[i].join();
    traccia C= new traccia(d);
    C.start(); C.join();
    }
}
class data {
    float tr, mat[][]=new float [M][M],s[]=new float [M];
    public data() {
        tr=0;
        for(int i=0; i<M; i++){
          for(int j=0; j<M; j++)
              mat[i][j]=(float)Math.random();
        }
    }
}
```

Implementazione in Java (Cont.): i Thread

```
class RowEner extends Thread{
    data d; int i;
    float s=0;
    public RowEner(data d, int indx) { this.d=d;this.i=indx; }
    public void run (){
        for(int j=0; j<M; j++) d.s[i]+=d.mat[i][j]*d.mat[i][j];
        d.s[i]=(float)Math.sqrt(d.s[i]);
    }
}
class T extends Thread {
    data d; int i; int j;
    public T(data d, int indx, int jndx) { this.d=d;this.i=indx;this.j=jndx; }
    public void run (){
         d.mat[i][j] /= d.s(i);
    }
}
class traccia extends Thread {
    private data d;  int i;
    public traccia(data d) {  this.d=d;  }
    public void run (){
    d.tr=0;
    for(i=0; i<M; i++) d.tr+=d.mat[i][i];
    for(i=0; i<M; i++) d.mat[i][i] /= d.tr;
    }
}
```