Assignment #1 - Smart Positioning

1) Analisi del problema

Si consideri un sistema formato da N particelle p[i] su un piano cartesiano. Nel sistema esiste un "campo di forze" per cui su ogni elemento p[i] viene esercitata una forza repulsiva da ogni elemento p[j] in modulo pari a:

$$F[i,j] = alfa[i] * alfa[j] * k / d[i,j]^2$$

Si vuole realizzare una versione concorrente della simulazione che rappresenti il moto delle particelle nel tempo, visualizzabile mediante l'utilizzo di una GUI, la quale deve permettere anche di poter lanciare e mettere in pausa la simulazione, facendola ripartire dallo stesso punto in cui si era fermata.

2) Descrizione della soluzione proposta

Nell'applicazione si utilizza un pattern di tipo Model-View-Controller (MVC), dove il Controller ha il compito di "ascoltare" la View per eventuali cambiamenti (pressione dei button di Start e Stop) e cambiare il contesto di gioco (rappresentato dalla classe "Context" che non è altro che un monitor in cui il thread Master fa un controllo sullo stato della simulazione e si comporta di conseguenza. Per esempio, se nella View viene pressato il pulsante Stop, allora il Master si mette in wait() e aspetta di essere risvegliato dal Controller, che modifica nuovamente il contesto.
Inoltre, si è optato per un'architettura di tipo Master-Slave, in cui il Master è un thread che ha il compito di creare i vari Slave (Workers in questo caso) e aggiornare la View dopo che tutti i Workers abbiamo terminato il loro lavoro.
In particolare, ogni Worker riceve tutta la lista di particelle, ma è delegato ad aggiornare soltanto una sotto-porzione della lista (tenendo conto anche di un numero di particelle non multiplo al numero di Workers), opportunamente calcolata dal Master.
La soluzione proposta prevede l'uso di 1 semaforo e di 3 monitor necessari alla corretta sincronizzazione e coordinazione dei thread in gioco.

Semaforo

Il semaforo è utilizzato per coordinare il Master con i Workers.
In particolare, una volta che il Master aggiorna la View, rilascia nel semaforo un numero di permessi pari al numero di workers, con "release(num_workers)", utilizzati da essi per poter compiere una nuova computazione di aggiornamento sulla lista di particelle, tramite "acquire()".
Nel dettaglio si tratta di un Event Semaphore, in cui l'evento è costituito dal completamento dello step, che comprende i calcoli completati dai Workers e dall'aggiornamento della View.

Monitor CyclicBarrier

Questo monitor rappresenta una barriera ciclica che i workers e il master utilizzano per garantire la sincronizzazione tra essi.
In particolare, viene creata una barriera con un numero di partecipanti pari al numero di workers + 1 (ovvero il master). All'inizio di ogni iterazione, il master si blocca nella barriera aspettando che l'ultimo worker la "rompa" per continuare così il lavoro (ovvero per poter aggiornare la View).

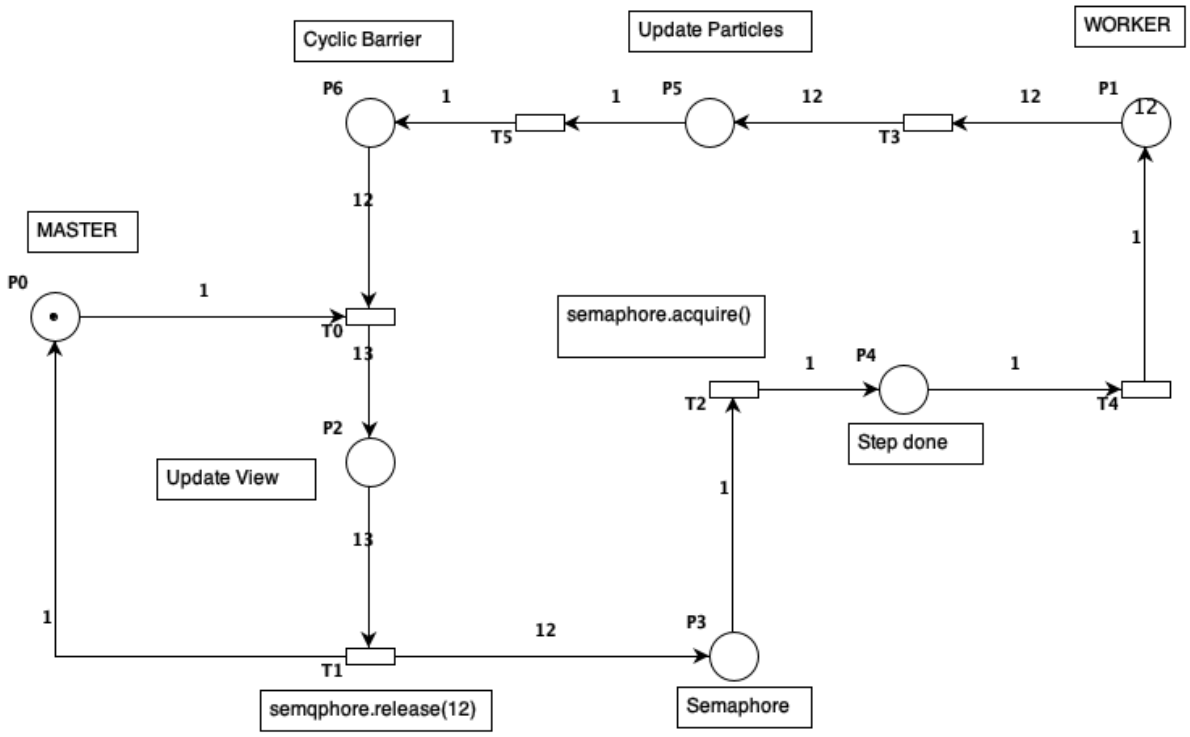
Monitor Coordinator

Questo monitor è utilizzato per "risvegliare" il Master, dopo la pressione del pulsante Start.
In particolare, ogni volta che si verifica un evento nella View, determinato dalla pressione dei button Start e Stop, il Master si mette in wait() aspettando che il Controller lo notifichi di un cambio di stato del gioco mediante una notifyAll(). In questo modo il Controller ha il compito di coordinare il Master, e di conseguenza anche i workers.

Monitor Context

Questo monitor rappresenta il vero contesto della simulazione.
Solo il Controller ha il compito di poter modificare lo stato del gioco, che sarà consultato dal Master, che si comporterà di conseguenza a seconda del contesto.

3) Dinamica del sistema mediante Rete di Petri

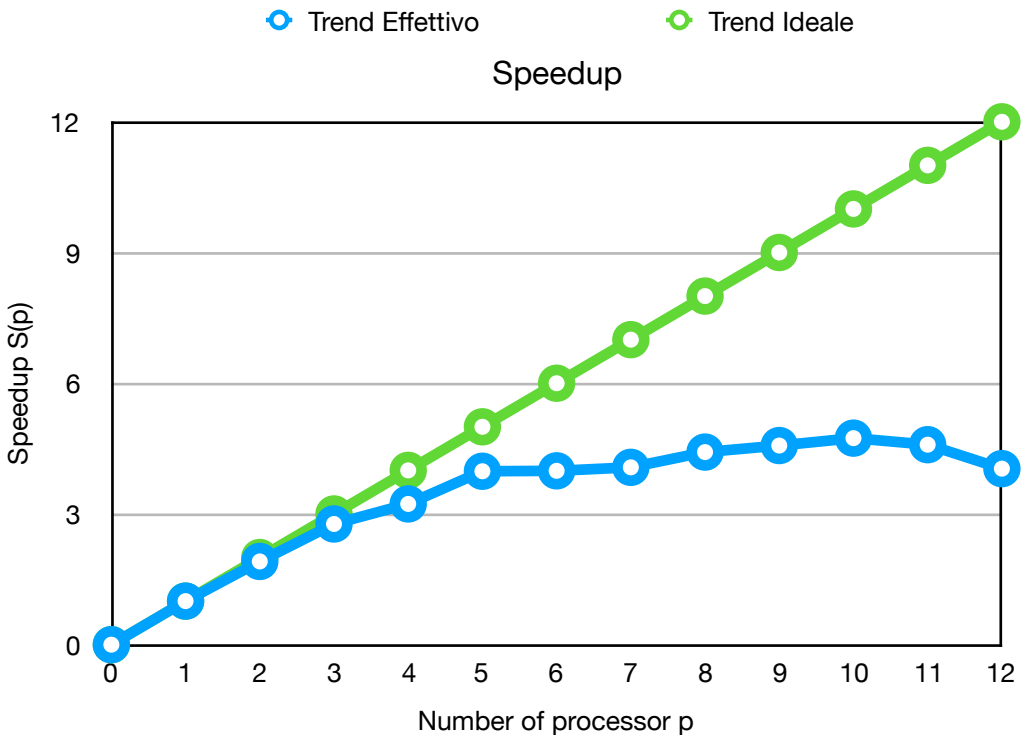


4) Performance della soluzione proposta

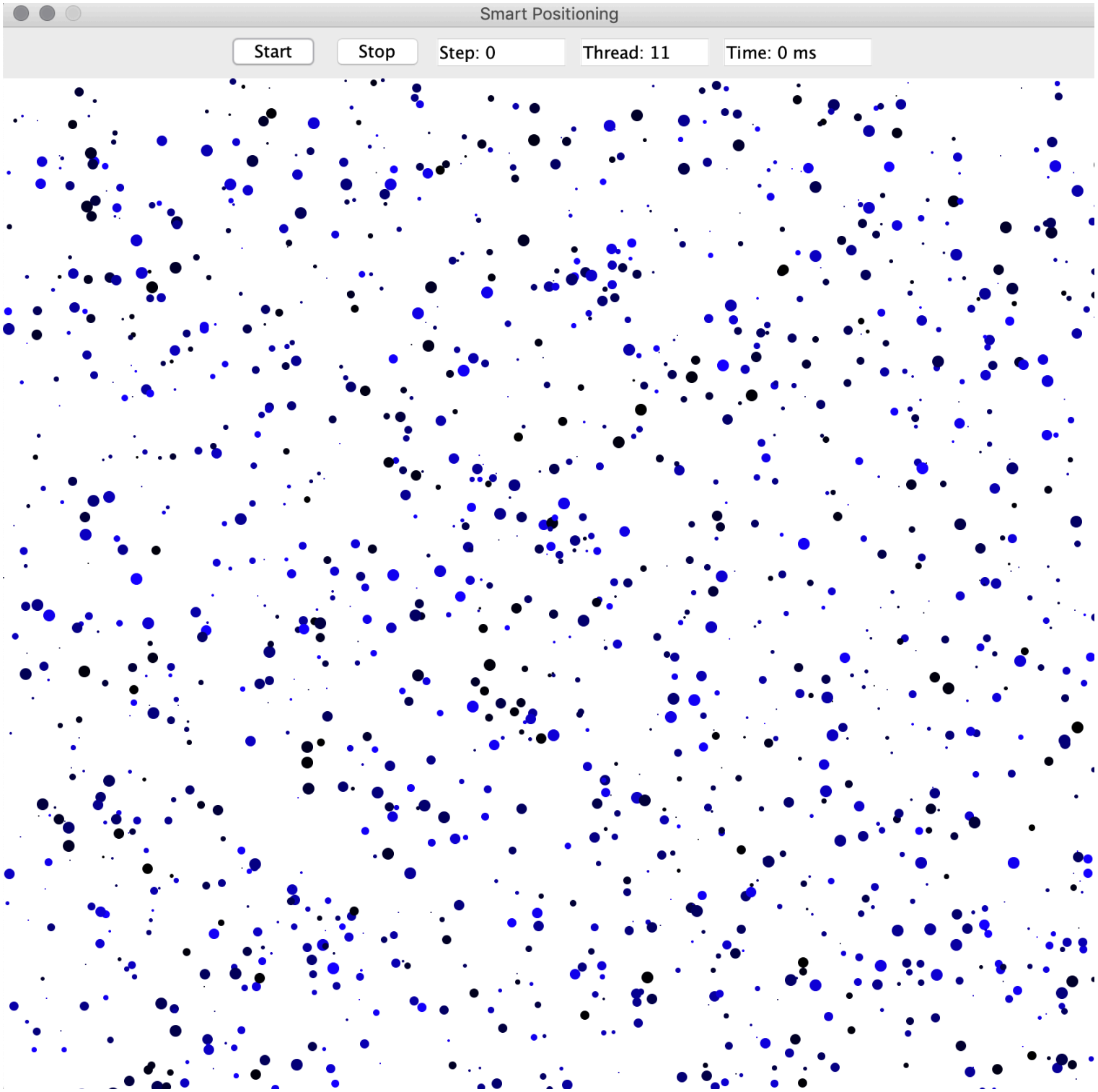
Per verificare l'efficienza della soluzione si sono presi diversi tempi di esecuzione con diverso numero di thread in gioco; confrontandoli con quelli della versione sequenziale con un solo worker (tabella seguente).

	N° PARTICELLE: 4000		N° ITERAZIONI: 1000		
NUMERO THREAD	**TEMPO 1**	**TEMPO 2**	**TEMPO 3**	**MEDIA**	**SPEEDUP**
1	66332 ms	67896 ms	66710 ms	**66979 ms**	**1**
2	34989 ms	35231 ms	35100 ms	**35106 ms**	**1,91**
3	23742 ms	24144 ms	24662 ms	**24182 ms**	**2,77**
4	21671 ms	20263 ms	20192 ms	**20709 ms**	**3,23**
5	16817 ms	16653 ms	17005 ms	**16825 ms**	**3,98**
6	16994 ms	16941 ms	16464 ms	**16800 ms**	**3,99**
7	16469 ms	16346 ms	16519 ms	**16445 ms**	**4,07**
8	15296 ms	14882 ms	15271 ms	**15149 ms**	**4,42**
9	14185 ms	14920 ms	14854 ms	**14653 ms**	**4,57**
10	14511 ms	13979 ms	13932 ms	**14140 ms**	**4,74**
11	14146 ms	15083 ms	14560 ms	**14596 ms**	**4,59**
12	16603 ms	16067 ms	17112 ms	**16594 ms**	**4,04**

Di seguito si mostra il grafico relativo allo speedup ottenuto con diverso numero di processori.



5) Gestione dell'Interfaccia Grafica



Siccome ogni particella ha un valore diverso di "Alpha" e "Massa", si è deciso di visualizzarle di conseguenza nella GUI. Le particelle più grandi avranno un valore di "Massa" superiore. Per quanto riguarda il colore, si è mappato il valore di "Alpha" nel canale Blu, ovvero a valori di Alpha bassi corrisponde il nero, mentre ad Alpha elevati corrisponde il blu (in un range da 0 a 255).