PROGETTO PCPC A.A. 19/20

Nome e cognome: Fulvio Somma

Progetto PCPC

Il progetto si pone come obiettivo la risoluzione del problema noto come N-body. Esso calcola la posizione e velocità, in uno spazio tridimensionale, di N corpi, o particelle, che interagiscono tra di loro modificando le caratteristiche appena citate; tale operazione avviene per I iterazioni ad intervalli di tempo t regolari.

Soluzione proposta

L'idea alla base della soluzione proposta consiste nel dividere il lavoro fra tutti i processi (compreso il master) e ad ogni iterazione terminata aggiornare i valori dei corpi in maniera centralizzata per poi ripetere l'operazione.

La soluzione sfrutta il calcolo parallelo per poter delegare ad ogni processo la computazione su X/p corpi, dove X è il numero totale di corpi e p il numero di processi. Poiché è necessario calcolare la forza esercitata su ogni corpo da parte di tutti gli altri, ad ogni iterazione il master distribuisce gli X corpi ai processi e ognuno calcola il proprio intervallo, successivamente ogni processo esegue il lavoro su tale intervallo e al termine invia i corpi modificati al master; a questo punto il processo con rank 0 aggiorna tutti i valori e invia di nuovo i corpi ricominciando il

processo. La modifica per ogni iterazione avviene anche grazie all'intervallo di tempo t specificato.

Il software presenta una fase di preparazione divisa in varie parti:

- Vengono creati casualmente il numero di iterazioni e di corpi, oltre che allocato lo spazio per ospitare ogni corpo.

- Una struttura specifica è inizializzata per poter semplificare lo scambio di messaggi in MPI.

- Il processo master crea tutti i corpi (in maniera random) salvandoli nella memoria precedentemente allocata e anche in un file per eventuali verifiche di correttezza.

In seguito, inizia la parte di computazione vera e propria che allo stesso modo si scompone in diversi passi:

- Il processo master invia in broadcast gli X corpi creati e poi si dedica alla computazione del proprio intervallo.

- Ogni altro processo riceve i valori, calcola il proprio intervallo di corpi da modificare ed esegue le stesse operazioni del processo con rank 0.

- Al termine si ha una operazione di gather di tutti i valori aggiornati in uno spazio di memoria apposito nel processo master.

- Il processo con rank 0 scrive su un file log tutti i valori ottenuti per quell'iterazione, successivamente si occupa di

inviare nuovamente i valori aggiornati a tutti gli altri processi ricominciando con una nuova iterazione.

Il calcolo delle forze da applicare al corpo viene fatto attraverso la seguente funzione:

```
void calcForce(int particle, Particle *part, int bodies){
	int prtcls;
	float fx = 0, fy = 0, fz = 0, interval = 0.1f;

	for(prtcls = 0; prtcls < bodies; prtcls++){
		if(prtcls != particle){
			float dx = part[prtcls].x - part[particle].x;
			float dy = part[prtcls].y - part[particle].y;
			float dz = part[prtcls].z - part[particle].z;
			float dist = sqrtf((dx * dx) + (dy * dy) + (dz * dz));
			float invDist3 = (1.0f / dist) * (1.0f / dist) * (1.0f / dist);
			fx += dx * invDist3;
			fy += dy * invDist3;
			fz += dz * invDist3;
		}
	}

	part[particle].vx += interval * fx;
	part[particle].vy += interval * fy;
	part[particle].vz += interval * fz;
}
```

Ogni processo esegue la funzione sul proprio intervallo e sovrascrive i vecchi dati grazie al seguente codice:

```
for(relBase = my_rank * intervalRoundBodies; relBase < my_rank * intervalRoundBodies +
intervalRoundBodies && relBase < nBodies; relBase++){
	calcForce(relBase, pr, nBodies);
}

for(relBase = my_rank * intervalRoundBodies; relBase <  my_rank * intervalRoundBodies +
intervalRoundBodies && relBase < nBodies; relBase++){
	pr[relBase].x += pr[relBase].vx * interval;
	pr[relBase].y += pr[relBase].vy * interval;
	pr[relBase].z += pr[relBase].vz * interval;
}
```

Le comunicazioni tra processi invece vengono compiute attraverso le funzioni MPI broadcast e gather:

```
MPI_Bcast(pr, roundBodies, mpi_particle, 0, MPI_COMM_WORLD);

//Task da eseguire per i calcoli...

MPI_Gather(tmpPr, intervalRoundBodies, mpi_particle, tmpRecvPr, intervalRoundBodies,
mpi_particle, 0, MPI_COMM_WORLD);
```

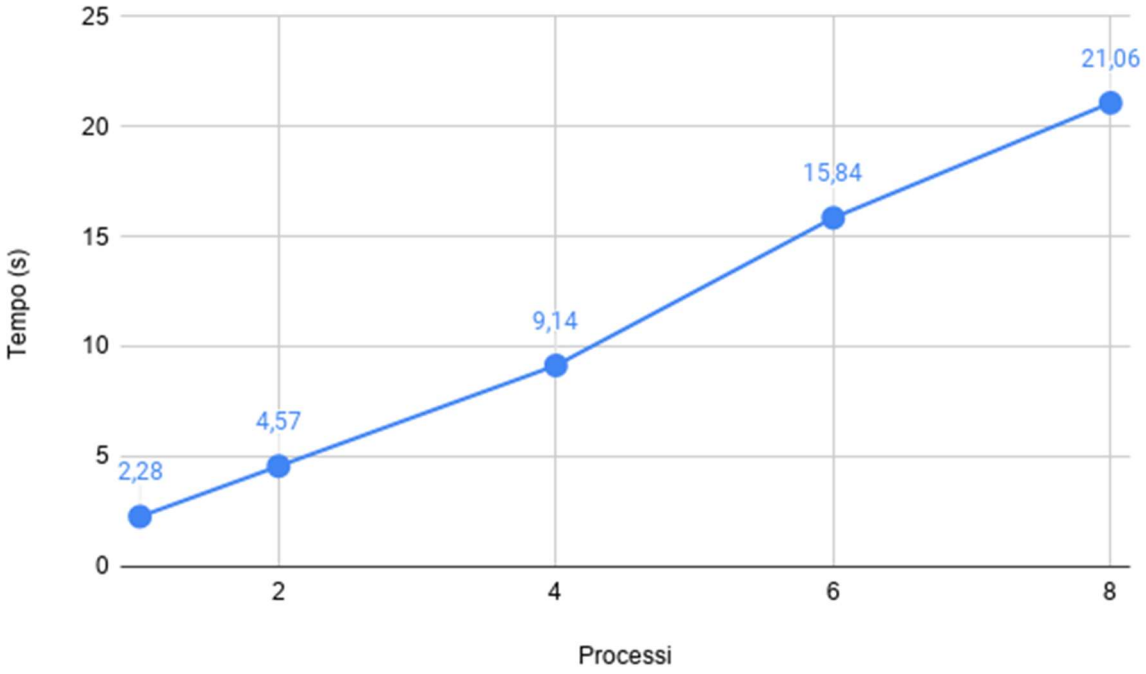
Risultati

Weak Scalability

Nella weak scalability viene calcolato il decremento dell'efficienza all'aumentare dei processi. In questo caso ogni processo ha sempre lo stesso carico di lavoro che quindi aumenta alla stessa velocità dei processi. Il numero di iterazioni è pari a cinque.

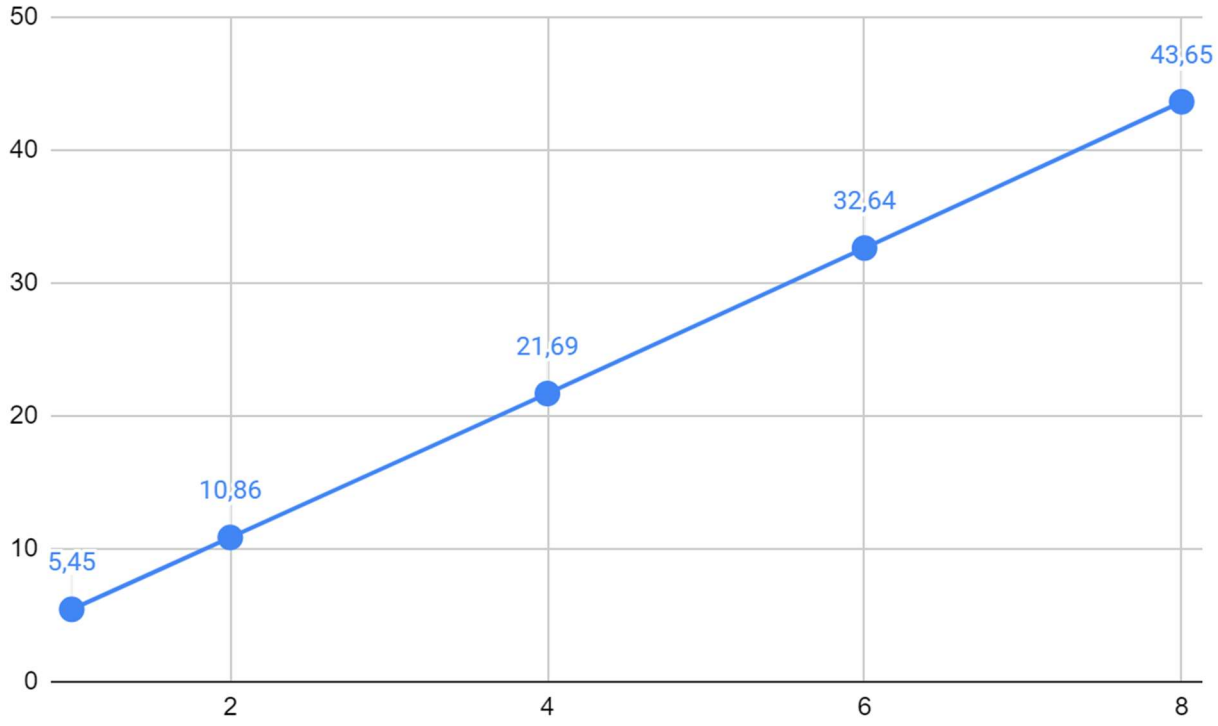
Iterazioni: 5

W/p	1	2	4	6	8
5000	**2.28**	1.15	0.59	0.48	0.38
10000	9.08	**4.57**	2.31	1.82	1.36
20000	36.24	18.17	**9.14**	7.1	5.36
30000	81.44	40.8	20.48	**15.84**	11.89
40000	144.75	72.7	36.35	28.28	**21.06**



Iterazioni: 10

W/p	1	2	4	6	8
5000	**5.45**	2.84	1.32	1.14	0.94
10000	21.29	**10.86**	5.23	4	3.15
20000	84.3	42.38	**21.69**	14.82	11.48
30000	188.66	94.67	48.03	**32.64**	24.96
40000	332.68	165.83	82.94	56.67	**43.65**

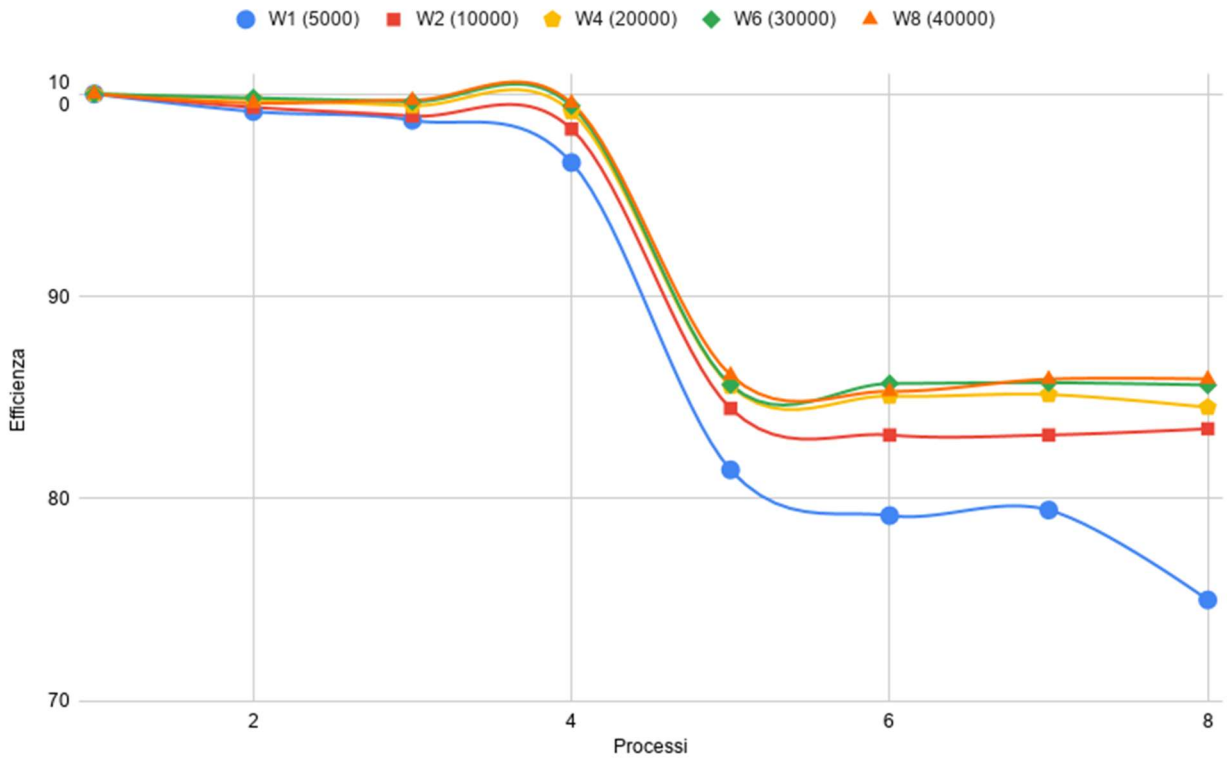


Strong Scalability

Nella strong scalability si calcola quanto velocemente deve aumentare la dimensione del problema per mantenere un'efficienza costante quando il numero i processi aumenta. In pratica, la dimensione del problema rimane la stessa mentre il numero di processi aumenta.

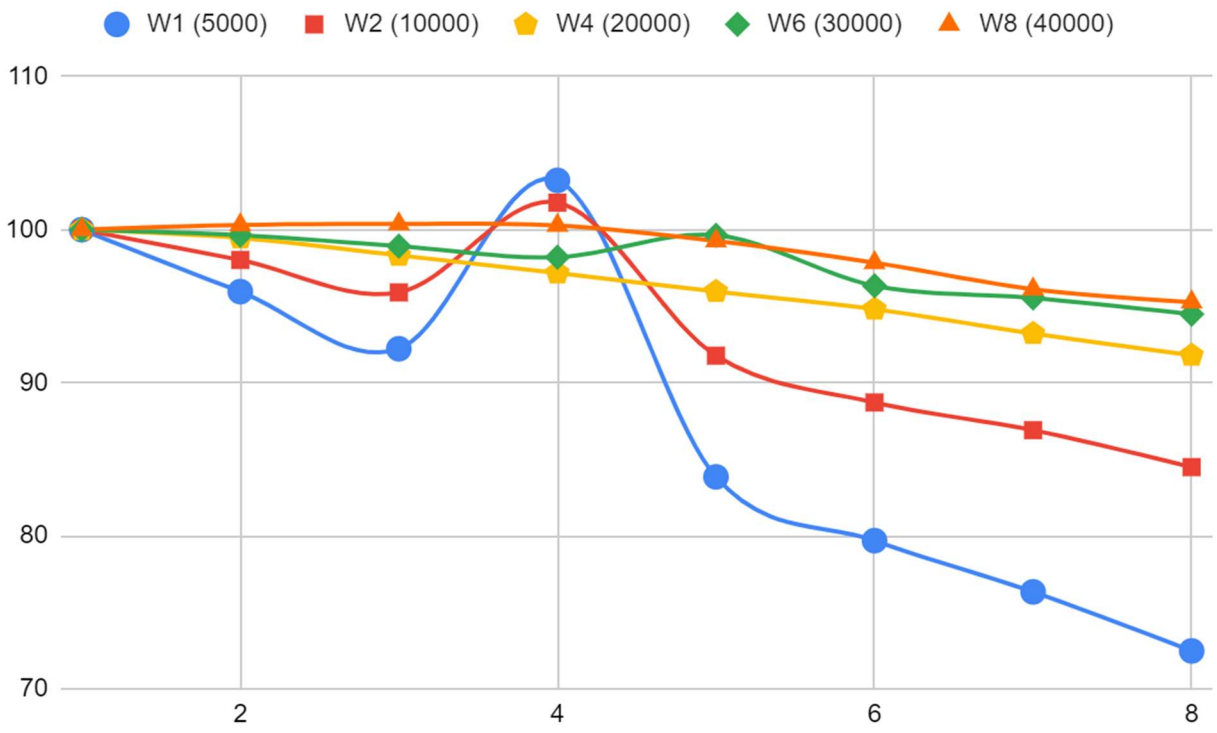
Iterazioni: 5

W/p	1	2	3	4	5	6	7	8
5000	2.28	1.15	0.77	0.59	0.56	0.48	0.41	0.38
10000	9.08	4.57	3.06	2.31	2.15	1.82	1.56	1.36
20000	36.24	18.17	12.15	9.14	8.47	7.1	6.08	5.36
30000	81.44	40.8	27.25	20.48	19.02	15.84	13.57	11.89
40000	144.75	72.7	48.4	36.35	33.62	28.28	24.07	21.06



Iterazioni: 10

W/p	1	2	3	4	5	6	7	8
5000	5.45	2.84	1.97	1.32	1.3	1.14	1.02	0.94
10000	21.29	10.86	7.4	5.23	4.64	4	3.5	3.15
20000	84.3	42.38	28.58	21.69	17.57	14.82	12.92	11.48
30000	188.66	94.67	63.57	48.03	37.87	32.64	28.21	24.96
40000	332.68	165.83	110.48	82.94	67.03	56.67	49.45	43.65



Descrizione risultati

Attraverso il grafo della weak scalability si può notare come i tempi necessari quando il carico di lavoro è costante per ogni processore siano lineari, il tempo necessario non aumenta mai più del doppio; grazie alla strong scalability si può evincere che all'aumentare dei processi, e quindi anche delle comunicazioni, l'efficienza non subisce un degrado eccessivo e resta sempre superiore al 70%, il comportamento peggiore in questo caso è dato dal carico di lavoro più basso poiché l'overhead di comunicazione diventa piuttosto rilevante rispetto al tempo necessario per effettuare il calcolo vero e proprio.

Conclusioni

Le sezioni viste fino ad ora hanno permesso di avere una visione della logica e del funzionamento del codice, nonché delle sue performance. Attraverso i due tipi di scalabilità è stato possibile notare l'efficienza e il tempo necessario a seconda dei processi e del workload. L'algoritmo ha dimostrato, attraverso i benchmark, di comportarsi bene nelle varie situazioni non perdendo troppa efficienza con l'aumento dei processori e non richiedendo un incremento di tempo esponenziale all'aumento del carico di lavoro. All’aumentare delle iterazioni il tempo impiegato aumenta in maniera lineare mentre l’efficienza segue sempre la stessa curva con il carico minore che peggiora più di tutti all’aumento ei processi.