La Matematica nella Società e nella Cultura

Rivista dell'Unione Matematica Italiana

Oscar Salas Huertas

Sviluppo di codici di simulazione numerica in ambienti di calcolo parallelo

La Matematica nella Società e nella Cultura. Rivista dell'Unione Matematica Italiana, Serie 1, Vol. **1** (2008), n.2 (Fascicolo Tesi di Dottorato), p. 339–342.

Unione Matematica Italiana

<http://www.bdim.eu/item?id=RIUMI_2008_1_1_2_339_0>

La Matematica nella Società e nella Cultura. Rivista dell'Unione Matematica Italiana, Unione Matematica Italiana, 2008.

La Matematica nella Società e nella Cultura
Rivista dell'Unione Matematica Italiana
Serie I, Vol. I, Agosto 2008, 339-342

Sviluppo di codici di simulazione numerica in ambienti di calcolo parallelo

Oscar Salas Huertas

1. – Introduzione.

Le applicazioni scientifiche e commerciali rappresentano una notevole forza trainante nello sviluppo di computer sempre più veloci. Applicazioni come esplorazioni petrolifere, lavori ambientali, data mining, ecc., richiedono il processamento di una gran quantità di dati in maniera sofisticata.

Lo scopo di questo lavoro è quello di descrivere i diversi approcci per lo sviluppo di codici di simulazione numerica in ambienti paralleli.

2. – Un precondizionatore parallelo per problemi ai limiti ellittici.

Lo sviluppo di metodi numerici per sistemi lineari algebrici che hanno la loro origine nella discretizzazione di equazioni differenziali alle derivate parziali è cruciale nello sviluppo di codici di calcolo efficienti per la fluido dinamica, l'elasticità ed altri problemi fondamentali della meccanica continua.

I problemi industriali e lo sviluppo di architetture parallele hanno determinato un considerevole sviluppo della tecnica di Domain Decomposition, che offre la possibilità di sfruttare il parallelismo matematico intrinseco.

L'idea dell'algoritmo BPS (J.H. Bramble, J.E. Pasciak, and A.H. Schatz, The Construction of Preconditioners for Elliptic Problems by Substructuring I, Mathematics of Computation, *47*, pp. 103-134, 1986) consiste nello splittare il dominio computazionale in sottodomini. Il precondizionatore è definito in forma *matrix-free*, in modo tale che il calcolo della sua applicazione ad un vettore implica la soluzione di problemi di minore dimensione sui sottodomini e di equazioni di interconnessione sulle interfacce, ed entrambi possono essere risolti in modo parallelo.

Un'analisi accurata dell'algoritmo consente di distinguere in esso tre blocchi computazionali, i quali devono essere eseguiti in forma rigorosamente sequenziale. Ognuno di questi blocchi consiste nella soluzione di sottoproblemi locali indipendenti nei sottodomini e sulle interfacce, che mostrano differenti gradi di parallelismo.

Il parallelismo matematico intrinseco può essere sfruttato in modo naturale assegnando un processore ad ogni sottodominio.

Il codice è stato implementato su un'architettura a memoria distribuita (cluster Beowulf) utilizzando il linguaggio Fortran90, seguendo gli standard Message Passing Interface (MPI) ed utilizzando la libreria parallela PETSc della Argonne

National Laboratory, in modo da assicurarne la portabilità. L'utilizzo della libreria PETSc permette di trarre beneficio sia dalle strutture parallele dell'algoritmo che dalla gestione ottimale della comunicazione e del calcolo.

Come problema modello abbiamo considerato l'equazione di Poisson sul quadrato unitario, discretizzato con elementi finiti lineari a tratti ed utilizzando una mesh triangolare regolare. Gli esperimenti sono stati condotti su due diversi cluster. Forniremo un'analisi dettagliata dei tempi di esecuzione ed alcune considerazioni sulla scalabilità ottenuta faccendo variare il numero di gradi di libertà (*d.o.f.*), e facendo crescere il numero di sottodomini. La scalabilità numerica dell'algoritmo viene studiata in termini di numero di iterazioni del Gradiente Coniugato Precondizionato (PCG), e la scalabilità parallela dell'implementazione è analizzata in termini di tempi di esecuzione.

Consideriamo i seguenti due set di prove numeriche: fissiamo il numero di *d.o.f.* uguale a 6561 per ogni sottodominio e facciamo crescere il numero di sottodomini, in questo modo il numero totale di *d.o.f.* del problema globale cresce (**Caso A**) e fissiamo il numero totale di *d.o.f.* per il problema globale uguale a 58081 e facciamo crescere il numero di sottodomini, in questo modo il numero di *d.o.f.* per sottodominio decresce (**Caso B**). Di seguito presentiamo i risultati ottenuti eseguendo il codice sul cluster AMD Opteron, appartenente all'IMATI-CNR di Pavia e al Dipartimento di Matematica dell'Università degli Studi di Pavia.

Abbiamo implementato anche due diverse strategie di scheduling: la prima è definita allocando un processo MPI per singolo nodo (caso A1, B1); la seconda è definita allocando tutti i possibili processi MPI su ciascun nodo (caso A2, B2), come ad esempio sul cluster AMD Opteron, dove è possibile allocare fino a 4 processi MPI per singolo nodo.

TABELLA 1. – Tempi di esecuzione in *secondi* sul cluster AMD Opteron

	Caso A1					Caso A2				
proc	tt	Init	PCG	bps/it	it	tt	Init	PCG	bps/it	it
4	11.95	9.39	2.17	0.16	13	7.75	5.48	1.93	0.14	13
9	12.30	9.38	2.53	0.21	13	8.07	5.61	2.04	0.15	13
16	14.05	9.46	4.14	0.17	24	13.97	9.49	3.98	0.18	24
25	14.19	9.45	4.21	0.18	24	14.50	9.55	4.31	0.19	24
36	—	—	—	—	—	15.30	9.46	5.18	0.18	28

Nelle Tabelle 1 - 2 vengono riportati i tempi totali di esecuzione del codice per i casi A1, A2, B1 e B2 sul cluster AMD Opteron, rispettivamente. La colonna "proc" è il numero di processori; "tt" è tempo di esecuzione impiegato dal codice; "Init" corrisponde essenzialmente al tempo di assemblaggio della coarse e fine mesh, ed all'inizializzazione ed assemblaggio dei vettori e delle matrici locali; "PCG" corrisponde ai tempi di esecuzione per singola iterazione del Gradiente Coniugato Precondizionato; "bps/it" è il tempo di una singola iterazione del BPS; "it" contiene il numero di iterazioni.

TABELLA 2. – Tempi di esecuzione in *secondi* sul cluster AMD Opteron

	Caso A1					Caso A2				
proc	tt	Init	PCG	bps/it	it	tt	Init	PCG	bps/it	it
4	38.42	28.19	9.45	0.73	14	56.66	44.54	11.24	0.93	14
9	8.49	5.72	2.09	0.15	13	—	—	—	—	—
16	4.29	2.76	1.16	0.05	23	4.50	2.77	1.26	0.05	23
25	2.30	1.05	0.54	0.02	22	—	—	—	—	—
36	—	—	—	—	—	1.59	0.49	0.53	0.02	26

Se si considera il Caso A, i tempi riportati nelle colonne PCG e bps/iter della Tabella 1 ci permettono di osservare la buona scalabilità parallela dell'implementazione: ad esempio, i tempi bps/iter rimangono stabili quando il numero dei sottodomini cresce. In modo analogo abbiamo che i tempi delle colonne PCG e bps/iter nella Tabella 2 relative al Caso B ci confermano la buona scalabilità del codice. Ulteriori esperimenti potrebbero essere eseguiti considerando altri tipi di mesh, modificando il termine noto ed il coefficiente di diffusione per poter affrontare i casi con anisotropia e discontinuità.

3. – Parallelizzazione di un modello ibrido classico-quantistico per un dispositivo MOSFET nanometrico.

Presentiamo lo sviluppo dell'implementazione parallela di un codice per la simulazione numerica di un dispositivo a semiconduttori nanometrico. Più precisamente siamo interessati ad un dispositivo nanometrico MOSFET Double-Gate (Metal Oxide Semiconductor Field Effect Transistor), che è una piccola struttura in Si/SiO_2. Il dispositivo consiste di due regioni ad alto dodraggio vicine ai contatti (source-drain) e una regione attiva, detta canale, a basso dodraggio.

Il progresso continuo delle tecnologie industriali per la produzione di dispositivi a semiconduttore negli ultimi cinquant'anni si è concentrato nella minimizzazione delle dimensioni dei componenti elettronici. Oggi i dispositivi nanometrici possono essere utilizzati in un gran numero di circuiti integrati, con il conseguente notevole miglioramento delle prestazioni. A scala nanometrica si verificano fenomeni quantistici quali l'interferenza, il confinamento, e l'effetto tunnel. Pertanto è necessario sviluppare nuovi modelli in grado di tener conto di questi fenomeni. Nel nostro caso utilizziamo il modello matematico chiamato Drift - Diffusion - Schrödinger - Poisson (DDSP), dove la simulazione degli effetti quantistici viene affidata alla soluzione di un problema di Schrödinger, il trasporto degli elettroni è modellato con un'equazione Diffusione-Trasporto, e il potenziale elettrostatico è calcolato risolvendo un'equazione di Poisson.

In questo lavoro si descrive l'operazione di porting di un codice preesistente su un'architettura a memoria condivisa. Il codice è stato scritto in Fortran90, utilizzando il pacchetto di Sparse Linear Algebra (SLAP), ed alcune subroutine del pacchetto Linear Algebra (LAPACK). L'analisi dell'algoritmo ci ha permesso di

individuare nella soluzione dei problemi di Schrödinger i nuclei computazionali che possono sfruttare un approccio parallelo. Inoltre, dopo una prima valutazione delle prestazioni del codice seriale, e considerando un certo numero di fattori, abbiamo deciso di implementare un parallelismo del tipo NAIF, perciò abbiamo utilizzato la libreria OpenMP API, che ci assicura la portabilità sulle macchine a memoria condivisa ed evita la necessità di dover re-ingegnerizzare il codice. Questo consente, tra l'altro, di effettuare le simulazioni utilizzando le attuali architetture di calcolo multicore che si stanno diffondendo sui personal computer.

Gli esperimenti numerici sono stati effettuati sul cluster AMD Opteron. Sottolineamo che ogni nodo di questo cluster è costituito da 4 CPU che condividono 8 GB di RAM, per cui eseguiamo fino a 4 threads contemporaneamente, (cioè eseguiamo al massimo un thread per singola CPU), inoltre utilizziamo il compilatore PGI versione 6.2 e OpenMP API versione 2.5, la libreria ACML 3.6.0 e il pacchetto SLAP versione 2.0. In seguito riportiamo i risultati ottenuti nella soluzione del problema DDSP considerando un dispositivo nanometrico MOSFET double-gate. L'approssimazione viene effettuata utilizzando una mesh con 2500 e 22500 *d.o.f.*, inoltre facciamo un'analisi del comportamento parallelo della nostra implementazione in termini di tempo totale di esecuzione.

TABELLA 3. – Tempo totale di esecuzione in *secondi* sul cluster AMD Opteron

	Caso con 2500 d.o.f		Caso con 22500 d.o.f	
# threads	tt	Schr/it	tt	Schr/it
1	52.7	0.09	349.3	0.75
2	44.1	0.05	271.6	0.39
3	41.1	0.02	245.8	0.26
4	39.7	0.02	230.8	0.20

Nella Tabella 3 vengono riportati i tempi ottenuti in funzione del numero di threads. "# Threads", contiene il numero di threads utilizzato; "tt" corrisponde al tempo totale di esecuzione speso dall'algoritmo; "Schr/it" è il tempo totale di esecuzione speso per risolvere l'equazione di Schrödinger in una singola iterazione di Gummel.

L'analisi dei valori riportati nella colonna "Schr/it" dimostra che il tempo diminuisce, come previsto, in proporzione al numero di threads utilizzati. Ciò conferma la buona scalabilità della nostra implementazione. I risultati sulla scalabilità appena citati vengono confermati inoltre della riduzione del tempo totale di esecuzione del codice, come si può osservare nella colonna "tempo totale" della Tabella 3

Considerando i risultati ottenuti, riteniamo che sono un buon punto di partenza per migliorare detta implementazione in modo da poterla applicare a futuri modelli.

Dipartimento di Matematica, Università degli Studi di Pavia
e-mail: oscar.salas@unipv.it
Dottorato in Matematica e Statistica (sede amministrativa: Università di Pavia) - Ciclo XIX
Direttore di ricerca: Prof. Giovanni Sacchi, CNR - IMATI, Pavia