UN ALGORITMO GENETICO PARALLEO PER IL PROBLEMA DELLA PARTIZIONE DEL GRAFO

Samuele Schiavi

Dipartimento di Fisica
Università degli Studi di Parma

Sommario

Gli algoritmi genetici sono tecniche di ricerca stocastica e ottimizzazione che possono essere utilizzate per un'ampia gamma di applicazioni. Questo articolo affronta l'applicazione degli algoritmi genetici al problema di partizione di grafi. Gli algoritmi genetici standard con popolazioni grandi soffrono di mancanza di efficienza (tempo di esecuzione abbastanza elevato). Viene proposto un algoritmo genetico massicciamente parallelo, un'implementazione su un Supernode di Transputer® e vengono forniti i risultati di vari benchmark.

L'algoritmo parallelo mostra uno speedup superlineare, nel senso che moltiplicando il numero di processori per p, il tempo impiegato per raggiungere una soluzione con un punteggio dato è diviso per kp (k>1).

Viene inoltre presentata un'analisi comparativa del nostro approccio con algoritmi di hill-climbing e simulated annealing. Le misure sperimentali mostrano che il nostro algoritmo fornisce risultati migliori sia per quanto riguarda la qualità della soluzione che il tempo necessario per raggiungerla.

Parole Chiave

Architetture parallele a memoria distribuita, Algoritmi genetici, Partizione di grafi, Hill-climbing, Problema di mapping, Simulated annealing, Speedup superlineare.

1 INTRODUZIONE

Dato un grafo, il problema di partizione di grafi cerca una partizione dei suoi nodi che ottimizzi una certa funzione di costo.

Ci sono numerose applicazioni pratiche di questo problema, per esempio:

- progettazione di V.L.S.I. (Integrazione su Scala Molto Larga) circuiti, dove, dato un insieme di componenti e un insieme di moduli, si vuole posizionare i componenti in modo da minimizzare il numero di connessioni tra moduli, pur preservando un certo equilibrio riguardante il numero di componenti in ogni modulo;

- routing in sistemi distribuiti, dove il problema considerato è quello di suddividere il componente di rete in cluster più piccoli in modo che l'overhead di controllo per il routing sia minimizzato;

- segmentazione di immagini nel campo della visione artificiale, dove le immagini segmentate sono rappresentate come grafi in cui ogni vertice rappresenta un segmento e ogni arco pesato tra due vertici rappresenta una relazione topologica tra due segmenti dell'immagine;

- sistemi di paging di memoria virtuale, dove si vuole distribuire oggetti diversi su pagine di memoria in modo da minimizzare il numero di riferimenti tra oggetti memorizzati in pagine diverse;

- mappatura di programmi paralleli su architetture parallele.

In questo laboratorio, siamo particolarmente interessati a quest'ultima applicazione, ovvero il posizionamento dei processi di comunicazione su processori di una macchina parallela a memoria distribuita. Un'indagine sui diversi metodi proposti nella letteratura per affrontare questo problema può essere trovata in [Talbi90]. Il programma parallelo è modellato come un grafo dove i vertici rappresentano i processi, i pesi dei vertici rappresentano i costi di calcolo noti o stimati di questi processi, gli archi rappresentano i collegamenti di comunicazione richiesti tra di essi e i pesi degli archi stimano la quantità relativa di comunicazione necessaria lungo quei collegamenti. Quando il numero di processi supera il numero di elementi di elaborazione disponibili, come di solito è il caso nella programmazione massicciamente parallela, il problema di mappatura include il problema di contrazione che è equivalente al problema di partizione di grafi trattato in questo articolo.

Il problema di partizione di grafi è NP-completo. Di conseguenza, dovrebbero essere usati metodi euristici per affrontarlo. Essi possono trovare soluzioni che sono solo approssimazioni dell'ottimo, ma lo faranno in un tempo ragionevole. I diversi approcci che sono stati proposti per questo problema possono essere divisi in due classi principali. Da un lato, gli algoritmi di ottimizzazione per scopi generali indipendenti dall'applicazione data e, dall'altro lato, gli approcci euristici progettati appositamente per un problema unico. Poiché vogliamo evitare lo svantaggio intrinseco degli algoritmi di questa seconda classe (applicabilità piuttosto limitata dovuta alla dipendenza dal problema) la nostra preoccupazione in questo articolo è solo la prima classe di algoritmi.

Due tecniche di ottimizzazione ampiamente utilizzate sono l'algoritmo di hill-climbing e il simulated annealing. L'algoritmo di hill-climbing è sicuro di trovare il minimo globale solo negli spazi convessi. Altrimenti, il più delle volte è un minimo locale piuttosto che un minimo globale che viene trovato. Il simulated annealing offre un modo per superare questo grande svantaggio dell'hill-climbing ma il prezzo da pagare per farlo è un tempo di calcolo importante. Peggio ancora, l'algoritmo di simulated annealing è piuttosto di natura sequenziale, la sua parallelizzazione è un compito abbastanza difficile.

Possono anche essere considerate tecniche di ottimizzazione più distribuite. Alcune di esse sono strettamente correlate agli algoritmi di reti neurali (vedi [Ackley87] & [Peretto90]). Altri, ovvero gli algoritmi genetici (GA) sono considerati in questo articolo. Sono tecniche di ricerca stocastica, introdotte da Holland venti anni fa, ispirate dall'evoluzione biologica delle specie. Lo sviluppo di architetture massicciamente parallele li ha resi molto popolari negli ultimi anni. Sono stati recentemente applicati a problemi di ottimizzazione combinatoria in vari campi, come, per esempio, il problema del commesso viaggiatore, l'ottimizzazione delle connessioni e della connettività delle reti neurali, e sistemi di classificazione.

Lo scopo di questo articolo è provare che il problema di partizione di grafi può essere risolto abbastanza efficientemente da un algoritmo genetico parallelo.

La struttura dell'articolo è la seguente:

- in una prima sezione, diamo una formalizzazione matematica del problema di partizione di grafi e discutiamo alcune istanze di funzioni di costo classiche.

- nella sezione successiva, presentiamo estensivamente l'approccio dell'algoritmo genetico al problema di partizione di grafi. Dopo un richiamo dei principi degli algoritmi genetici, mostriamo come possono essere utilizzati per affrontare il problema di partizione di grafi, discutiamo la questione degli algoritmi genetici paralleli e infine esponiamo la soluzione proposta.

- la terza e ultima sezione, dopo una presentazione dettagliata dell'implementazione del Supernode, presenta i risultati di diversi benchmark confrontando la velocità o la qualità dei risultati di questo algoritmo date diverse dimensioni di popolazione o un dato numero di processori. Viene anche presentata un'analisi comparativa della soluzione dell'algoritmo

genetico con hill-climbing e simulated annealing. Infine, osservazioni conclusive e possibili estensioni di questo lavoro sono proposte.

2 FORMALIZZAZIONE MATEMATICA DEL PROBLEMA DI PARTIZIONE DI GRAFI

Dato:

- un grafo non diretto $G = (V, E)$;
- un'applicazione Ω_1 da V in $\mathbb{Z}^+$, tale che $\Omega_1(v_i) = w_{1i}$ è il peso del vertice v_i;
- un'applicazione Ω_2 da E in $\mathbb{Z}^+$, tale che $\Omega_2(e_i) = w_{2i}$ è il peso dell'arco e_i;
- e un insieme di vincoli numerici $\Phi = \{\phi_1, \phi_2, ..., \phi_m\}$ su questi pesi;

il problema di partizione di grafi deve trovare una partizione Π di V ($\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$) che soddisfi i vincoli Φ.

Un insieme classico e ben studiato $\Phi 1$ di vincoli esprime che:

- per ogni sotto-insieme π_i di V appartenente alla partizione Π, la somma dei pesi dei suoi vertici deve essere inferiore a un dato valore B

$$\forall\ \pi_i \in \Pi, \quad \sum_{v \in \pi_i} \Omega_1(v) \leq B$$

- la somma dei pesi degli archi che vanno da un nodo di π_i a un nodo di qualche altro π_j deve essere inferiore a un dato valore C

$$\sum_{e \in \varepsilon} \Omega_2(e) \leq C$$

con $\varepsilon = \{(x, y)/(x, y) \in E\ \&\ x \in \pi_i\ \&\ y \in \pi_j\ \&\ i \neq j\}$

Il problema di partizione di grafi sotto vincoli $\Phi 1$ è stato dimostrato NP-completo (vedi [Garey79]).

La maggior parte delle applicazioni corrisponde al seguente insieme $\Phi 2$ di vincoli dove per ogni sotto-insieme π_i di V appartenente alla partizione Π, il numero di nodi in π_i è uguale a un dato valore B_i

$$\forall\ \pi_i \in \Pi, \quad \sum_{v \in \pi_i} \Omega_1(v) = B_i$$

con $\forall\ v \in V,\ \Omega_1(v) = 1$

- il costo totale degli archi che vanno da un π_i a un altro π_j dovrebbe essere minimo

$$\text{MIN}(\sum_{e \in \varepsilon} \Omega_2(e))$$

Il problema di partizione di grafi sotto vincoli $\Phi 2$ è stato anche dimostrato NP-completo (vedi [Hyafil73]).

Per la nostra applicazione, il mapping di programmi paralleli su architetture parallele, dobbiamo considerare il seguente insieme $\Phi 3$ di vincoli:

- minimizzare la somma dei costi totali di comunicazione tra processori (costo totale degli archi che vanno da un π_i a un altro π_j) e della varianza dei carichi dei diversi processori (varianza del costo dei vertici appartenenti a un dato π_i)

$$\text{MIN}\left(\sum_{e\in\varepsilon}\Omega_2(e) + (K * \frac{\left(\sum_{\pi_i\in\Pi}\left(\sum_{v\in\pi_i}\Omega_1(v)\right)^2\right)}{|\Pi|} - \left(\frac{\sum_{v\in V}\Omega_1(v)}{|\Pi|}\right)^2)\right)$$

Con $K = 0$ l'insieme di vincoli $\Phi 3$ si riduce a $\Phi 2$. Questo dimostra che il problema di partizione sotto vincoli $\Phi 3$ è NP-completo.

Per il problema di mapping, K è il peso del contributo del costo di comunicazione relativo al bilanciamento del carico totale nel sistema. La scelta di un valore adatto per K dipende dalla conoscenza delle caratteristiche dell'architettura parallela. Valori molto piccoli di K suggerirebbero una soluzione uniprocessore, e valori molto grandi ridurrebbero il problema a uno di scheduling multiprocessore senza costi di comunicazione. L'architettura parallela utilizzata era una rete di transputer e $K = 2$ è stato scelto.

3 SOLUZIONE DELL'ALGORITMO GENETICO AL PROBLEMA DI PARTIZIONE DI GRAFI

3.1 Principi degli algoritmi genetici e loro applicazione al problema di partizione di grafi

Gli algoritmi genetici compongono una famiglia molto interessante di algoritmi di ottimizzazione. Il loro principio base è abbastanza semplice.

Dato uno spazio di ricerca Σ di dimensione M^N, dati M simboli, qualsiasi punto di questo spazio può essere rappresentato da un vettore di N di questi M simboli.

Data una funzione di fitness F da Σ in $\Re$ che associa un valore reale a qualsiasi punto di Σ.

Data un insieme iniziale di vettori, chiamata la popolazione iniziale.

Alcuni operatori genetici vengono utilizzati per generare nuovi punti di Σ dati alcuni vecchi in una fase del processo chiamata riproduzione. Durante questa fase, alcuni punti di Σ vengono sostituiti mantenendo fissa la dimensione della popolazione. Il principio fondamentale di GA è: più adatto un vettore, più probabile la sua riproduzione. In termini matematici significa che la probabilità P di riproduzione è crescente come F aumentava:

$$\forall\ \sigma_1, \sigma_2 \in \Sigma, \quad F(\sigma_1) > F(\sigma_2) \Rightarrow P(\sigma_1) > P(\sigma_2)$$

L'algoritmo genetico standard è:
Generare una popolazione di individui casuali.
Mentre nber_of_generations $\leq$ max_nber_of_generations
Fare

- **Valutazione** - assegnare un valore di fitness a ogni individuale.
- **Selezione** - fare un elenco di coppie di individui che probabilmente si accoppieranno, con individui più adatti elencati più frequentemente.
- **Riproduzione** - applicare operatori genetici alle coppie selezionate.
- **Sostituzione** - formare una nuova popolazione sostituendo i peggiori individui con i migliori.

Gli operatori genetici più comuni utilizzati durante la riproduzione sono il crossover e la mutazione. Crossover, dati due vettori, tagliali entrambi nello stesso punto casuale e scambia i vettori così tagliati (fig.1a). La mutazione è semplicemente cambiare un bit (fig.1b). Devono essere definiti due parametri: Pc e Pm. Rappresentano rispettivamente la probabilità di applicazione del crossover e le mutazioni di operatori. Altri operatori genetici possono essere trovati

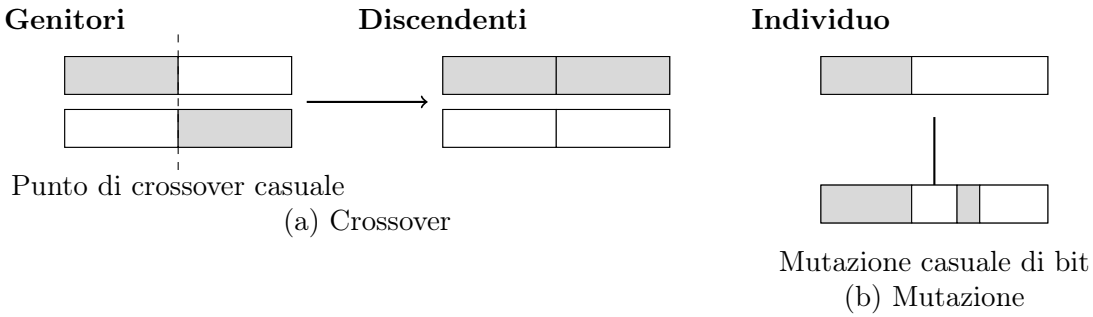

Figura 1: Operatori genetici.

in letteratura. Per esempio, l'operatore di inversione o molte varianti dell'operatore di crossover progettato per domini di problemi specifici.

Per utilizzare l'algoritmo genetico per il problema di partizione di grafi, è necessaria la seguente formalizzazione:

Supponiamo di avere un grafo di N nodi da dividere in M sotto-insiemi. Ciascuno di questi sotto-insiemi è etichettato da un simbolo (per esempio un numero intero tra 0 e M-1). Una data partizione è rappresentata da un vettore N di quei simboli; dove il simbolo p in posizione q significa che il nodo q del grafo è nel sotto-insieme p.

La funzione di fitness F è l'ultima funzione di costo descritta nella sezione II. Usiamo la versione usuale di crossover, ma la mutazione è una prova casuale di uno degli M simboli possibili.

3.2 Un algoritmo genetico parallelo

Due approcci agli algoritmi genetici paralleli sono stati considerati finora:

approccio parallelo standard: In questo approccio, la valutazione e la riproduzione sono fatte in parallelo. Tuttavia, la selezione è ancora fatta sequenzialmente, perché richiederebbe un grafo completamente connesso di individui poiché due individui qualsiasi nella popolazione potrebbero essere accoppiati.

approccio di decomposizione: Questo approccio consiste nel dividere la popolazione in sotto-popolazioni di dimensioni uguali. Ogni processore esegue l'algoritmo genetico sulla propria sotto-popolazione, selezionando periodicamente buoni individui da inviare ai suoi vicini e ricevendo periodicamente copie dei buoni individui dei vicini per sostituire i cattivi nella propria sotto-popolazione. Il vicinato del processore, la frequenza di scambio e il numero di individui scambiati sono parametri regolabili.

Il modello parallelo standard non è flessibile nel senso che l'overhead di comunicazione della popolazione cresce come il quadrato della dimensione della popolazione. Pertanto, questo approccio non è adatto ad architetture a memoria distribuita, dove la comunicazione ha un grande impatto sulle prestazioni dei programmi paralleli. Nel modello di decomposizione, il parallelismo inerente non è completamente sfruttato poiché il trattamento delle sotto-popolazioni può essere ulteriormente decomposto. Questo approccio dovrebbe essere considerato solo quando il numero di processori disponibili è inferiore alla dimensione richiesta della popolazione.

Considerando architetture massicciamente parallele con numerosi processori, scegliamo un modello a grana fine, dove la popolazione è mappata su un grafo di processori connesso come una griglia, un individuo per processore. Abbiamo una biiezione tra l'insieme di individui e l'insieme di processori. La selezione è fatta localmente in un vicinato di ogni individuo. Un'altra versione di questo approccio è stata già proposta in [Mühlenbein89], dove ad ogni generazione viene eseguito un algoritmo di hill-climbing per ogni individuo nella popolazione.

La scelta del vicinato è il parametro regolabile. Per evitare l'overhead e la complessità degli algoritmi di routing in macchine distribuite parallele, una buona scelta può essere quella di limitare il vicinato solo agli individui direttamente connessi.

L'algoritmo genetico parallelo proposto è:

Generare *in parallelo* una popolazione di individui casuali.
Mentre nber_of_generations $\leq$ max_nber_of_generations
Fare

- **Valutazione** - Valutare *in parallelo* ogni individuo.
- **Selezione** - Ricevere *in parallelo* gli individui provenienti dai suoi vicini.
- **Riproduzione** - Ogni individuo si riproduce *in parallelo* con gli individui precedentemente ricevuti.
- **Sostituzione** - Fare *in parallelo* una selezione dei migliori discendenti locali.

È importante notare che queste modifiche del modello standard non causano una degradazione nell'efficienza di ricerca dell'algoritmo genetico standard come mostrato in [Anderson90] e [Mühlenbein88].

4 IMPLEMENTAZIONE DEL SUPERNODE E BENCHMARKING

4.1 Implementazione del Supernode

Il Supernode è una macchina vagamente accoppiata, altamente parallela basata su transputer (fig.2). Una delle sue caratteristiche più importanti è la sua capacità di riconfigurare dinamicamente la topologia di rete utilizzando un dispositivo programmabile V1 switch device. Può passare tra varie configurazioni da 16 a 1024 processori, fornendo da 24 a 1500 Mflops di prestazioni di picco. Per ottenere queste prestazioni, è stata adottata una struttura gerarchica. Il componente di base è un transputer T800 a 32-bit con microprocessore, memoria on-chip (Floating Point Unit), fornendo 10Mips e 1.5Mflops di prestazioni di picco. La comunicazione tra transputer è supportata da collegamenti seriali, bidirezionali, asincroni, point-to-point. Una stazione host Sun viene utilizzata per fornire la connessione tra il processore root e il mondo esterno.

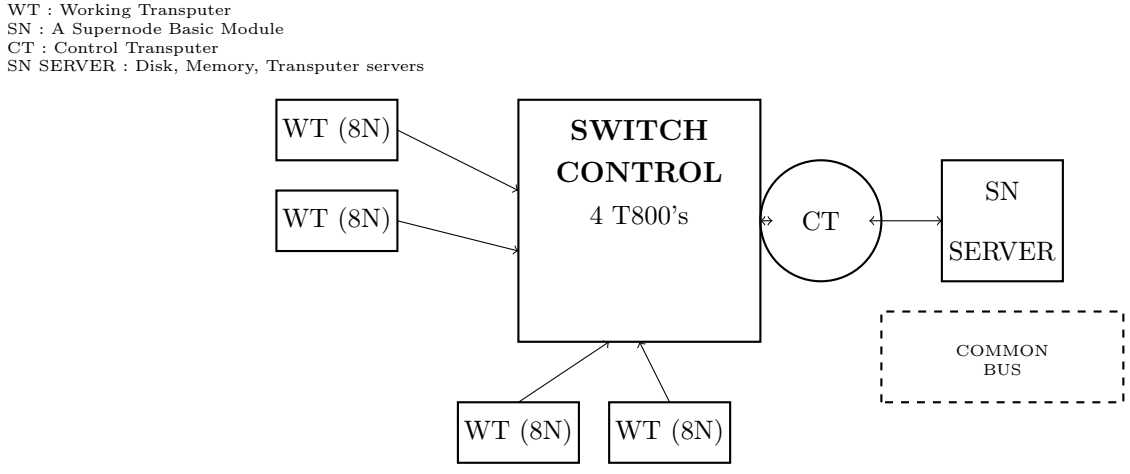

Figura 2: Supernode Parallel Architecture.

L'ambiente di programmazione utilizzato nei nostri esperimenti è Parallel C 3L. Una configurazione del network fisico sviluppata nel nostro laboratorio è stata utilizzata per ottenere la topologia desiderata dell'architettura.

Assumiamo che ogni individuo nella popolazione risieda su un processore e la comunicazione è effettuata mediante passaggio di messaggi. Quanto segue è una descrizione pseudo-Occam del processo eseguito in parallelo da ogni processore:

```
SEQ
   Generate (local_individual)
   Evaluate (local_individual)
   While nber_of_generations <= max_nber_of_generations
      SEQ
         -- communication phase
         PAR i=0 FOR nber_of_neighbors
            PAR
               neighbor_in[i] ? neighbor_individual[i]
               neighbor_out[i] ! local_individual
         -- computation phase
         PAR i=0 FOR nber_of_neighbors
            Reproduction(local_individual,
                         neighbor_individual[i])
         Replacement
```

Ogni riproduzione produce due discendenti. La nostra strategia è scegliere casualmente uno dei discendenti. La fase di sostituzione consiste nel sostituire l'individuo locale corrente con il miglior discendente locale prodotto nella fase di riproduzione.

La popolazione è posta su un toro. Dati i quattro collegamenti del transputer, ogni individuo ha quattro vicini. Non è necessario routing nella rete di processori perché solo i processori direttamente connessi devono scambiare informazioni.

Non consideriamo la migliore soluzione trovata globalmente poiché la comunicazione coinvolta nel determinare questa soluzione sarebbe considerevole. Prendiamo solo la migliore soluzione di routing attraverso un processo spy posizionato sul processore root (vedi fig. 3).

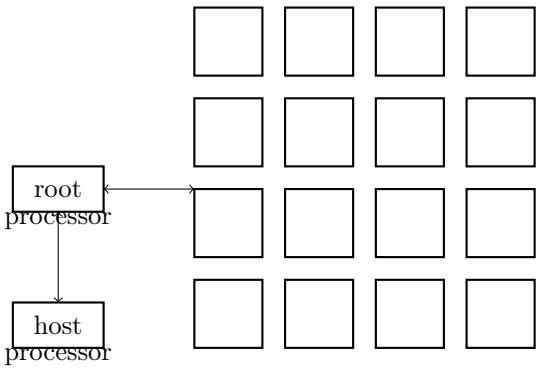

Figura 3 Un toro di 16 processori.

4.2 Variare il numero di processori

Lo scopo di questo benchmark è misurare lo speed-up quando si esegue l'algoritmo genetico parallelo (per una data dimensione di popolazione) su diverse dimensioni di toro di processori.

Usiamo il rapporto di speed-up come metrica per le prestazioni dell'algoritmo genetico parallelo. Il rapporto di speed-up S è definito come $S = Ts/Tp$ dove Ts è il tempo di esecuzione su un singolo processore e Tp corrisponde al tempo di esecuzione per p processori implementazione. La Figura 4 mostra i risultati ottenuti.

L'algoritmo ha uno speed-up quasi lineare. Ciò è dovuto al fatto che il costo di comunicazione tra processi è relativamente piccolo rispetto al costo di calcolo, ed è indipendente dalla dimensione dell'architettura.

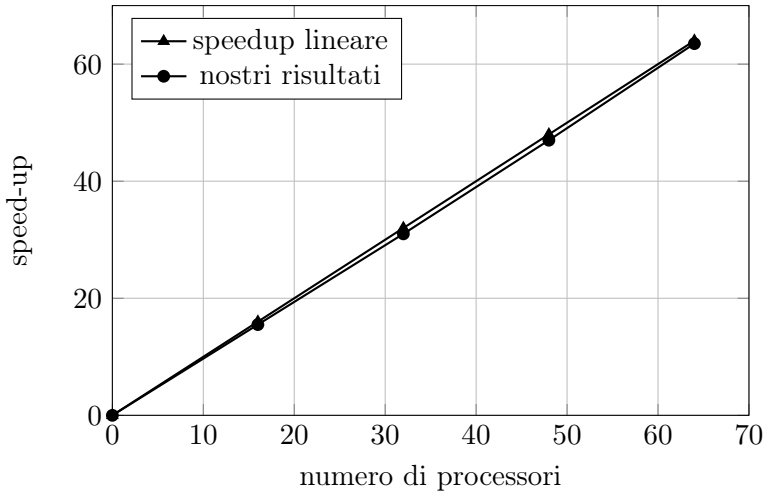

Figura 3: Speed-up dell'algoritmo parallelo

4.3 Variare la dimensione della popolazione

Lo scopo di questo benchmark è misurare l'evoluzione della qualità della soluzione quando si esegue l'algoritmo genetico parallelo con diverse dimensioni di popolazione.

La Figura 5 mostra i risultati ottenuti.

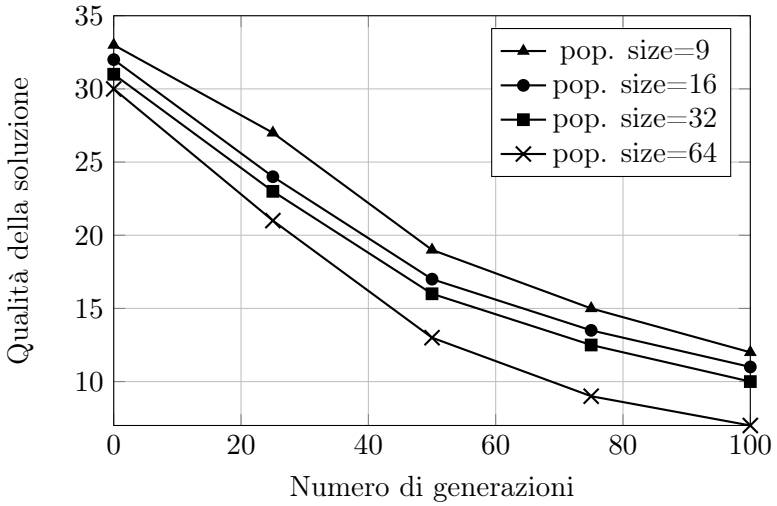

Figura 4: Qualità della soluzione in funzione della dimensione della popolazione.

Si noti che dato il problema di partizione di grafi specifico utilizzato (una pipeline di 32 vertici da partizionare in 8 sotto-insiemi) per questo benchmark, la migliore soluzione possibile ottiene un punteggio di 7.

Come previsto, per un dato numero di generazioni, la qualità migliora con un aumento della dimensione della popolazione.

Può anche accadere che per una popolazione troppo piccola si verifichi una convergenza prematura e che la soluzione ottimale non venga mai raggiunta.

La Figura 5 mostra anche che la maggiore riduzione del costo della partizione si verifica all'inizio. Quindi una qualità moderata del partizionamento può essere ottenuta molto rapidamente.

4.4 Tempo per raggiungere una data soluzione

Lo scopo di questo benchmark è studiare lo speed-up per una data qualità della soluzione quando si esegue l'algoritmo genetico parallelo su diverse dimensioni di toro di processori e con diverse dimensioni di popolazioni (entrambe uguali dato che c'è un individuo per processore).

La Figura 6 mostra l'influenza del numero di processori (e dimensione della popolazione) sul tempo necessario per raggiungere una soluzione con punteggio 8.

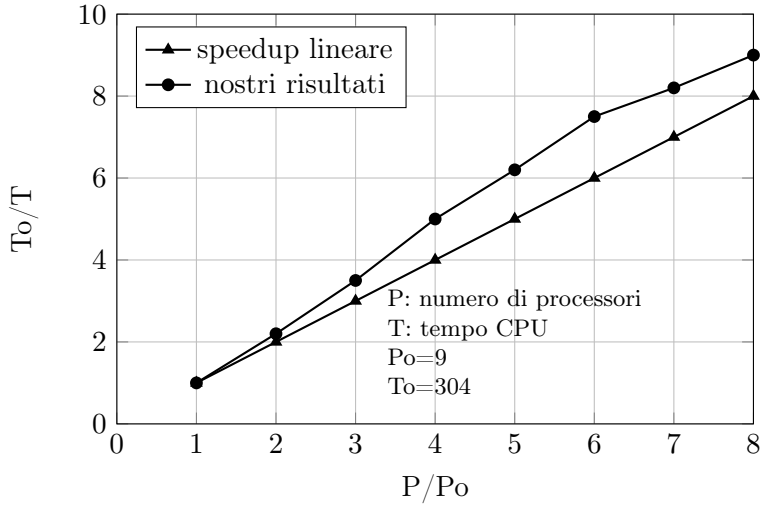

Figura 5: Tempi di esecuzione su diverse dimensioni di architetture parallele.

numero di processori	CPU tempo
9	304
16	164
32	71
64	33

Abbiamo uno speed-up superlineare dell'algoritmo genetico parallelo, nel senso che moltiplicando il numero di processori per p il tempo di esecuzione è diviso per kp ($k > 1$).

4.5 Confronto con algoritmi di hill-climbing e simulated annealing

In questa sezione, le prestazioni dell'algoritmo proposto vengono confrontate con algoritmi di hill-climbing e simulated annealing.

4.5.1 Algoritmo di hill-climbing

L'algoritmo di hill-climbing inizia con una configurazione casuale e cerca di migliorarla. Il miglioramento viene effettuato in piccoli passi consistenti nel spostare un vertice da un sotto-insieme all'altro. Una mossa viene selezionata casualmente, viene valutato il cambiamento di costo della

mossa, e se il cambiamento è per il meglio la mossa viene accettata e viene generata una nuova configurazione. Altrimenti, la vecchia configurazione viene mantenuta. Questo processo viene ripetuto fino a quando non ci sono più cambiamenti alla configurazione che ridurrà ulteriormente la funzione di costo. Quando ciò si verifica, è stato trovato un minimo locale, piuttosto che il minimo globale richiesto. La Figura 7 mostra una versione dell'algoritmo di hill-climbing.

Figura 7: L'algoritmo di hill-climbing.

```
Generate a random initial state S0 (S:=S0).
Repeat
   Compute at random a neighboring state S'.
   if cost(S') < cost(S) then S:=S'
Until there is no better neighbor.
```

4.5.2 Algoritmo di simulated annealing

Il principio dell'algoritmo di simulated annealing è il seguente: il sistema viene messo in un ambiente ad alta temperatura. A questa temperatura viene applicata una sequenza sufficientemente lunga di trasformazioni elementari casuali (catena di Markov) per raggiungere l'equilibrio a questa temperatura. Poi, la temperatura viene leggermente diminuita e viene applicata una nuova sequenza di mosse casuali. Ad ogni temperatura gli stati energetici consentiti sono governati dal criterio della metropoli, che permette alla configurazione di accettare uno stato con una probabilità $P(\Delta E, T)$. Il termine di ricerca termina quando il sistema si stabilizza.

C'è una grande quantità di letteratura su questo argomento, e l'algoritmo base permette una variazione e ottimizzazione considerevoli dei parametri. La versione che abbiamo considerato è una versione semplice, che può probabilmente essere notevolmente migliorata. Tuttavia, poiché l'algoritmo genetico parallelo è anche un algoritmo naive, out-of-the-shell, la versione che abbiamo pensato fornisce i seguenti benchmark che danno, almeno, un interessante ordine di grandezza.

Il numero di cambiamenti disponibili alla configurazione, denotato da L, quando si sposta un vertice in un sotto-insieme diverso, è dato da $L = N * (M - 1)$, dove N è il numero di vertici del grafo da partizionare e M il numero di sotto-insiemi della partizione. Questo valore fornisce una misura della dimensione del problema ed è utilizzato come parametro nello schedule di annealing.

1. (Passo di inizializzazione)
   - iniziare con una configurazione iniziale casuale $S0$ ($S := S0$);
   - $T := Tmax$;
2. (Hill-climb stocastico)
   - generare e calcolare uno stato vicino casuale S';
   - $\Delta E := \text{cost}(S') - \text{cost}(S)$
   - selezionare la nuova configurazione ($S := S'$) con probabilità $P(\Delta E, T) := \min(1, \exp(-\Delta E/T))$;
   - ripetere questo passo $X * N * (M - 1)$ volte; /* lunghezza della catena di markov tenuta ad ogni T */
3. (Test di anneal/convergenza)
   - impostare $T := aT$;
   - se $T \geq Tmin$ andare a step2.

Figura 8 L'algoritmo di simulated annealing.

4.5.3 Protocollo sperimentale e risultati

Ogni algoritmo è stato eseguito 10 volte per ottenere una stima media delle prestazioni. Gli esperimenti sono stati eseguiti su due problemi diversi:

- una pipeline di 32 vertici da partizionare in 8 sotto-insiemi;
- e una griglia di 64 vertici da partizionare in 4 sotto-insiemi.

Per l'algoritmo genetico, utilizziamo una popolazione di 64 configurazioni in esecuzione su un toro 8 per 8 di processori. Lo schedule di annealing e i parametri dell'algoritmo genetico che sono stati utilizzati durante i nostri esperimenti sono dati dalla tabella 1 e 2.

Simbolo	Valore	Descrizione
Tmax	10	Temperatura iniziale
Tmin	0.1	Temperatura minima
a	0.9	Tasso di decadimento della temperatura
X	2	Lunghezza della catena di markov

Tabella 1: Lo schedule di annealing.

Simbolo	Valore	Descrizione
POPS	64	Dimensione della popolazione
Pc	1	Probabilità di crossover
Pm	65	Probabilità di mutazione

Tabella 2: Parametri dell'algoritmo genetico.

Le tabelle seguenti mostrano il valore minimo, massimo, medio e la varianza delle soluzioni ottenute. I risultati per l'hill-climbing e gli algoritmi di simulated annealing sono basati su un'implementazione su un singolo transputer T800.

Si può osservare dalle tabelle 3 e 4 che l'algoritmo genetico supera gli algoritmi di hill-climbing e simulated annealing sia nella qualità della soluzione che nel tempo utilizzato nella ricerca.

Algoritmo	Soluzione				Tempo CPU
	min	max	mean	deviation	(sec)
Hill-climbing	7.5	12.5	9.9	1.94	353
Simulated annealing	7.5	9	8.05	0.37	2296
Algoritmo genetico	7	8	7.5	0.15	64

Tabella 3: Benchmarking con una pipeline di 32 vertici e una partizione di 8 sotto-insiemi.

Per il simulated annealing, i migliori risultati possono essere ottenuti utilizzando valori più grandi di X, tuttavia, aumentare X aumenterà il tempo di calcolo.

5 CONCLUSIONI E DIREZIONI FUTURE

Un algoritmo genetico parallelo per risolvere il problema di partizione di grafi è stato proposto e valutato.

I risultati principali sono i seguenti:

- l'algoritmo mostra uno speedup superlineare;
- è facile da programmare;

Algoritmo	Soluzione				Tempo CPU
	min	max	mean	deviation	(sec)
Hill-climbing	24	47	32.9	42.49	643
Simulated annealing	21	27	24	4.00	3358
Algoritmo genetico	17	25	21	6.25	93

Tabella 4: Benchmarking con una griglia di 64 vertici e una partizione di 4 sotto-insiemi.

- è semplice da implementare su architetture parallele a memoria distribuita massicciamente parallele;
- supera gli algoritmi di hill-climbing e simulated annealing sia nella qualità della soluzione che nel tempo utilizzato per raggiungerla.

Una caratteristica importante degli algoritmi genetici è che possono essere utilizzati per risolvere una grande varietà di problemi di ottimizzazione combinatoria. Li stiamo usando per risolvere tali problemi di ottimizzazione nel campo del controllo robotico e delle reti neurali.

Stiamo anche studiando un miglioramento importante dell'algoritmo, vale a dire, la variazione dinamica dei suoi parametri e in particolare la probabilità di mutazione. L'operatore di crossover diventa meno efficace nel tempo man mano che le stringhe nella popolazione diventano più simili. Un modo per evitare la convergenza prematura e sostenere la diversità genetica è utilizzare la mutazione adattiva. Durante le prime generazioni quando c'è ampia diversità nella popolazione, la mutazione deve verificarsi a tassi molto bassi. Tuttavia, man mano che la diversità diminuisce nella popolazione, il tasso di mutazione deve aumentare.

Sono pianificati più lavori teorici: verrà utilizzato un modello basato su automi cellulari per studiare l'influenza dei parametri dell'algoritmo sulla sua convergenza.

BIBLIOGRAFIA

Riferimenti bibliografici

[1] D.H.Ackley, "A connectionist machine for genetic hillclimbing", *Kluwer Academic Pub, Boston*, 1987.

[2] E.J.Anderson, M.C.Ferris, "A genetic algorithm for the assembly line balancing problem", Tech. Rep. Rpt.926, University of Wisconsin-Madison, Mar 1990.

[3] F.Baiardi, S.Orlando, "Strategies for a massively parallel implementation of simulated annealing", PARLE'89, LNCS, Vol.366, Eindhoven, Netherlands, pp.272-287, June 1989.

[4] F.Berman, L.Snyder, "On mapping parallel algorithms into parallel architectures", *J of Parallel and Distributed Computing 4*, pp.439-458, 1987.

[5] P.Bessiere, "Toward a synthetic cognitive paradigm: probabilistic inference", Proc. of COGNITIVA90, Madrid, Spain, 1990.

[6] A.Bouloutas, P.M.Gopal, "Some graph partitioning problems and algorithms related to routing in large computer networks", *Proc. 9th Int. Conf. on Distributed Computing Systems*, pp.362-370, 1989.

[7] J.P.Cohoon, S.U.Hedge, W.N.Martin, D.Richards, "Punctuated Equilibra: A parallel genetic algorithm", *Proc. of the Second Int. Conf. on Genetic Algorithms*, Cambridge MA, pp.148-154, Jul 1987.

[8] T.A.Feo, M.Khellaf, "A class of bounded approximation algorithms for graph partitioning", *Networks, Vol.20, No.2*, pp.181-195, Mar 1990.

[9] M.R.Garey, D.S.Johnson, "Computers and intractability: A guide to the theory of NP completeness", *Freeman, San Francisco*, 1979.

[10] J.J.Grefenstette, "Incorporating problem specific knowledge into genetic algorithms", *in Genetic algorithms and Simulated annealing, L.Davis ed., Morgan Kaufman Publishers*, 1987.

[11] P.Haden, F.Berman, "A comparative study on mapping algorithms for an automated parallel programming environment", Tech. Rep., CS-088, Univ. of California, San Diego, 1988.

[12] L.Hérault, J-J.Niez, "How neural networks can solve the graph K-partitioning", *Neuro-Nimes'89 Int. Workshop on Neural Networks and applications, Nimes, France*, pp.237-255, Nov 1989.

[13] J.H.Holland, "Adaptation in natural and artificial systems", *Ann Arbor: Univ. of Michigan Press*, 1975.

[14] L.Hyafil, R.L.Rivest, "Graph partitioning and constructing optimal decision trees are polynomial complete problems", *RR No.33, IRIA, Rocquencourt, France*, Oct 1973.

[15] C.R.Jesshope, T.Muntean, C.Whitby-strevens, J.G.Harp, "Supernode Project P1085: Development and application of a low cost high performance multiprocessor machine", *ESPRIT'86, Brussels*, 1986.

[16] D.S.Johnson, C.H.Papadimitriou, M.Yannakakis, "How easy is local search ?", *Proc. Annual Symp. on Foundations of Computer Science*, pp.39-42, 1985.

[17] B.W.Kernighan, S.Lin, "An efficient heuristic procedure for partitioning graphs", *The Bell System Tech. Journal*, pp.291-307, Feb 1970.

[18] S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi, "Optimization by simulated annealing", *Science, Vol.220, No.4598*, pp.671-680, May 1983.

[19] P.J.M.Laarhoven, E.H.L.Aarts, "Simulated annealing: Theory and applications", *D. Reidel Pub. Comp.*, 1987.

[20] E.L.Lawler, K.N.Levitt, J.Turner, "Module assignment for delay in digital networks", *IEEE Trans. on Comp., Vol.C-18, No.1*, pp.47-57, Jan 1969.

[21] D.Macfarlane, I.East, "An investigation of several parallel genetic algorithms", *Proc. of the Occam User Group, Exeter, UK*, pp.60-67, Apr 1990.

[22] R.M.MacGregor, "On partitioning a graph: a heuristic and empirical study", *Memorandum No.UCB/ERL M78/14, Electronics Research Laboratory, Univ. of California, Berkeley*, 1978.

[23] H.Mühlenbein, M.Gorges-schleuter, O.Krämer, "Evolution algorithms in combinatorial optimization", Parallel Computing, Vol.7, No.2, pp.65-85, Apr 1988.

[24] H.Mühlenbein, J.Kindermann, "The dynamics of evolution and learning: Towards genetic neural networks", *in Perspective, R.Pfeifer et al. eds., North-Holland*, pp.173-197, 1989.

[25] C.B.Pettey, M.R.Leuze, J.J.Grefenstette, "A parallel genetic algorithm", *Proc. of the Second Int. Conf. on Genetic Algorithms, MIT, Cambridge*, pp.155-161, Jul 1987.

[26] P. Peretto, "Neural networks and combinatorial optimization", *Proc. of the international conference on Neural Networks, E.N.S. of Lyon, Lyon, FRANCE*, 1990.

[27] E.Robertson87 P.Robertson, "Parallel implementation of genetic algorithms in a classifier system", *in Genetic algorithms and Simulated annealing, L.Davis ed., Morgan Kaufman Publishers*, pp.129-140, 1987.

[28] R.L.Russo, P.H.Oden, P.K.Wolff, "A heuristic procedure for the partitioning and mapping of computer logic graphs", *IEEE Trans. on Comp., Vol.C-20, No.12*, pp.1455-1462, 1971.

[29] J.E.Savage, M.G.Wloka, "On parallelizing graph-partitioning heuristics", *Automata, Languages and Programming, Warwick Univ., UK, LNCS No.443*, pp.476-489, Jul 1990.

[30] K.Shahookar, P.Mazumder, "A genetic approach to standard cell placement using meta-genetic parameter optimization", *IEEE Trans. on Computer-Aided Design, Vol.9, No.5*, pp.500-511, May 1990.

[31] J.Sheild, "Partitioning concurrent VLSI simulation programs onto a multiprocessor by simulated annealing", *IEE Proceedings, Vol.134, Pt.E, No.1*, pp.24-30, Jan 1987.

[32] E-G.Talbi, T.Muntean, "Static allocation of communicating processes on a parallel architecture", *Research Rep. RR-833-I-, LGI/IMAG, INPG*, Nov 1990.

[33] R.Tanese, "Parallel genetic algorithms for a hypercube", *Proc. of the Second Int. Conf. on Genetic Algorithms, MIT, Cambridge*, pp.177-183, Jul 1987.

[34] D.Whitley, T.Starkweather, C.Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity", *Parallel Computing, Vol.14, No.3*, pp.347-361, Aug 1990.