Algoritmi genetici massivamente paralleli

Samuele Schiavi
Dipartimento di Fisica
Università degli Studi di Parma

Sommario

Gli algoritmi euristici sono solitamente impiegati per trovare una soluzione ottimale ai problemi NP-Completi. Gli algoritmi genetici sono tra questi algoritmi e sono algoritmi di ricerca basati sulla meccanica della selezione naturale e della genetica. Poiché gli algoritmi genetici lavorano con un insieme di soluzioni candidate, la parallelizzazione basata sul paradigma SIMD sembra essere il modo naturale per ottenere un aumento di velocità. In questo approccio, la popolazione di stringhe è distribuita tra gli elementi di elaborazione. Ciascuna delle stringhe viene quindi elaborata indipendentemente dalle altre. Il guadagno di prestazioni per questo approccio deriva dall'esecuzione parallela delle stringhe e, quindi, dipende fortemente dalla dimensione della popolazione. L'approccio è favorito per le applicazioni di algoritmi genetici in cui l'insieme di parametri per una particolare esecuzione è ben noto in anticipo e in cui tali applicazioni richiedono una grande dimensione della popolazione per risolvere il problema. DDAP si adatta perfettamente ai requisiti sopra indicati. L'obiettivo della parallelizzazione è duplice: il primo è accelerare il processo di allocazione in DDAP che di solito consiste in migliaia di documenti e deve utilizzare una grande dimensione della popolazione, e secondo, può essere visto come un tentativo di portare i processi dell'algoritmo genetico nelle macchine SIMD.

1 Introduzione

Gli algoritmi genetici sono solitamente utilizzati per risolvere problemi NP-hard. L'intrattabilità totale del problema utilizzando metodi convenzionali, così come le soluzioni più veloci rendono gli algoritmi genetici la scelta preferibile. Inoltre, per il tipo di problema in cui è impossibile ottenere una conoscenza a priori del problema, gli algoritmi genetici superano qualsiasi altro metodo.

A differenza dei metodi convenzionali, gli algoritmi genetici richiedono che l'insieme di parametri di un problema sia codificato come una stringa di lunghezza finita su un certo alfabeto. La bontà del modello nella stringa viene quindi utilizzata per trovare una soluzione al problema. Pertanto, invece di regolare il codice stesso, gli algoritmi genetici sfruttano le somiglianze di codifica per arrivare alla soluzione ottimale.

Un problema durante la ricerca di una soluzione ottimale nei problemi NP-hard è che, invece di raggiungere l'ottimo globale, si può finire in un ottimo locale. Gli algoritmi genetici, d'altra parte, lavorano con un insieme di soluzioni candidate in parallelo. Pertanto, la possibilità che tutti i punti colpiscano un ottimo locale è molto minima. Gli algoritmi genetici di solito manipolano una popolazione di stringhe e successivamente generano un'altra popolazione di stringhe costruite da pezzi delle popolazioni precedenti. Ciò si traduce in uno spazio di ricerca più completo rispetto ad altri metodi, con meno sforzo. Di conseguenza, questo approccio crea un robusto algoritmo di ricerca.

La maggior parte delle macchine a processori massicciamente paralleli attualmente disponibili fino ad ora rientra nella classe SIMD (Single Instruction Multiple Data). Queste macchine consistono in migliaia di processori che eseguono lo stesso insieme di istruzioni. Ciascuno dei processori di solito contiene un diverso insieme di dati, quindi in un ciclo di esecuzione, la macchina può produrre risultati multipli. Questo tipo di architettura di macchina è utile per problemi di manipolazione di array.

Poiché gli algoritmi genetici manipolano una popolazione di stringhe, l'implementazione di questi algoritmi sulla macchina SIMD è desiderabile. La maggior parte delle operazioni eseguite

su una stringa sono indipendenti dalle altre stringhe nella popolazione. Pertanto, distribuire le stringhe sui processori e manipolarle individualmente su processori diversi è il modo naturale per parallelizzare gli algoritmi genetici. Solo una piccola quantità di elaborazione come il calcolo del valore di fitness relativo delle stringhe deve essere eseguita centralmente, ma la maggior parte dell'elaborazione può essere eseguita localmente su processori separati.

L'algoritmo parallelo ideale deve esibire due caratteristiche chiave: (1) speedup lineare: il doppio dell'hardware può eseguire il compito nella metà del tempo trascorso, e (2) scaleup lineare: il doppio dell'hardware può elaborare un problema due volte più grande nello stesso tempo trascorso. In questo articolo, proponiamo un algoritmo genetico parallelo per macchine SIMD che ha queste caratteristiche desiderabili. Queste caratteristiche possono essere ottenute solo se tutti i processi sono eseguiti localmente, cioè nessun calcolo è eseguito centralmente e nessun dato deve essere trasmesso da un processore all'altro. Deve essere sviluppata una strategia per minimizzare il tempo sprecato nella parallelizzazione degli algoritmi genetici. La prossima sezione è dedicata alla discussione dei vari paradigmi paralleli e continua con una visione di ciò che è stato fatto nella parallelizzazione degli AG. La sezione quattro propone un nuovo algoritmo genetico parallelo. La discussione evidenzierà anche lo speedup ottenuto da questo approccio. Nella sezione cinque, implementiamo l'algoritmo parallelo utilizzando il problema di allocazione di documenti a memoria distribuita (DDAP) come esempio, e quindi discutiamo i guadagni di prestazione. La sezione sei conclude questo articolo.

2 Paradigmi Paralleli

Flynn ha definito quattro organizzazioni di computer: SISD, SIMD, MISD e MIMD. Dei quattro, MISD non ha ricevuto molta attenzione, mentre SISD è solo un computer sequenziale. Quindi ci rimangono due organizzazioni di computer paralleli: SIMD e MIMD. Di conseguenza, gli algoritmi paralleli possono essere classificati in due classi: (1) classe SIMD e (2) classe MIMD. Nella classe di algoritmi SIMD, le istruzioni sono eseguite in serie, ma le istruzioni che operano su dati locali sono eseguite simultaneamente su ogni processore. Fondamentalmente, ogni processore serve solo come memoria per dati diversi che verranno eseguiti da un'applicazione in parallelo. Pertanto, un'applicazione che impiega un calcolo omogeneo su un grande insieme di dati, come il calcolo di array o matrici, beneficia di questa disposizione.

La classe MIMD di algoritmi paralleli è destinata a gestire applicazioni più eterogenee e irregolari. Gli algoritmi consistono in diversi sotto-processi. Questi sotto-processi possono essere eseguiti indipendentemente e cooperare inviandosi messaggi reciprocamente. Un sotto-processo può essere considerato come un programma completo e opera sui propri dati. Le applicazioni che impiegano un approccio a pipeline, o applicazioni che consistono in diversi sotto-compiti indipendenti sono adatte per questo paradigma.

Per parallelizzare un'applicazione arbitraria, la natura dell'applicazione deve essere ispezionata prima. Se consiste in diversi compiti indipendenti, allora possiamo trasformarla in uno stile di esecuzione MIMD assegnando ogni compito a un processore diverso e permettendo loro di eseguire indipendentemente. Se l'applicazione impiega un approccio a pipeline, allora ogni fase della pipeline può essere assegnata a un processore diverso e i dati possono essere fatti fluire da un processore all'altro.

Non tutte le applicazioni si adattano perfettamente a queste categorie. Per esempio, se tutti i compiti sono interdipendenti, o nel caso in cui la pipeline forma un ciclo, parallelizzare tale applicazione aumenterà il tempo di elaborazione. Per le applicazioni che non possono essere prontamente parallelizzate in uno stile MIMD, è disponibile un approccio SIMD.

L'approccio SIMD ha limitazioni che derivano dal fatto che ogni processore su di esso agisce all'unisono su una singola istruzione, ma su punti dati diversi. Pertanto, le applicazioni da parallelizzare devono essere eseguite su una grande quantità di dati, come nei calcoli di array. Inoltre, alcune applicazioni basate su statistiche richiedono esecuzioni multiple delle applicazioni

per raccogliere sufficientemente dati per ulteriori analisi. Assegnare ciascuna di queste esecuzioni a un processore diverso e eseguirle con parametri diversi può eliminare la necessità di eseguirle sequenzialmente. Questi tipi di applicazioni sono tra le applicazioni più adatte per essere implementate nello stile SIMD.

3 Revisione della Letteratura

Attualmente, c'è molta letteratura che riporta lo sforzo nella parallelizzazione degli algoritmi genetici. La maggior parte di questi sforzi utilizza l'approccio MIMD, con l'eccezione del lavoro di Hermann e Reczko.

Il lavoro di parallelizzazione degli AG è stato pionierato da Grefenstette nel suo articolo, in cui ha delineato quattro modelli di algoritmi genetici paralleli:

- **Master-slave sincrono**: un singolo processo AG master coordina k processi slave. Il processo master controlla la selezione, l'accoppiamento e le prestazioni degli operatori genetici, mentre gli slave eseguono semplicemente le valutazioni delle funzioni. Sebbene questo approccio sia molto diretto e facile da implementare, l'approccio soffre di due grandi svantaggi:

  1. Una discreta quantità di tempo viene sprecata se c'è molta varianza nel tempo di valutazione della funzione e quindi, il tempo totale è sempre il tempo per completare la funzione più lunga.
  2. L'approccio non è molto affidabile: dipende dalla salute del master, se va giù, l'intero sistema si ferma.

  La Figura 1 illustra questo approccio.

- **Master-slave semisincrono**: Rilassa il requisito per l'operazione sincrona inserendo e selezionando membri al volo mentre gli slave completano il loro lavoro. Sebbene sia destinato a superare il primo svantaggio dell'approccio precedente, questo approccio soffre ancora del secondo svantaggio.

- **Concorrente asincrono distribuito**: k processori identici eseguono sia operazioni genetiche che valutazioni di funzioni indipendentemente l'uno dall'altro, accedendo a una memoria condivisa comune. Poiché la memoria condivisa impedisce ai processori di accedere alla stessa posizione simultaneamente, ogni processore lavora su un sottoinsieme non sovrapposto della popolazione. Sebbene questo approccio sia meno diretto da implementare, supera entrambi gli svantaggi introdotti dal primo approccio. La Figura 2 illustra questo approccio.

- **Rete**: In questo approccio, k semplici AG indipendenti funzionano con memorie indipendenti, operazioni AG indipendenti e valutazioni di funzioni indipendenti su processori indipendenti. La popolazione utilizzata da ciascun AG è considerata come un sottoinsieme della popolazione totale. I k processi lavorano come AG normali, con l'eccezione che i migliori individui scoperti in una generazione vengono trasmessi alle altre sottopopolazioni tramite una rete di comunicazione. Quindi, la larghezza di banda del collegamento è ridotta rispetto ad altri approcci. Inoltre, l'affidabilità è molto alta a causa dell'autonomia dei processi indipendenti.

Bianchini e Brown hanno riportato il loro sforzo nell'implementazione di un algoritmo genetico parallelo su un sistema MIMD basato su transputer. In questo articolo Bianchini e Brown hanno realizzato che implementare un AG parallelo con l'approccio a rete soffre di un grave svantaggio del sistema che deve dividere la popolazione disponibile in sottopopolazioni più piccole

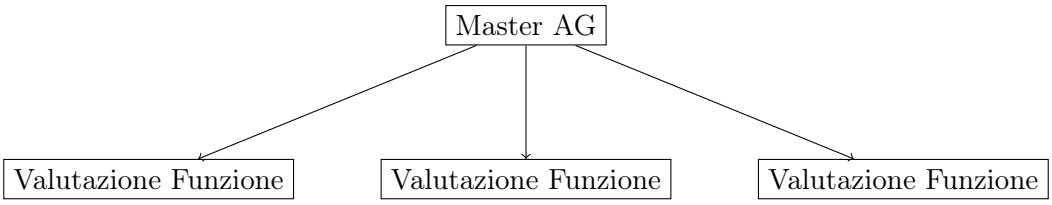


Figura 1: Modello AG master-slave sincrono.

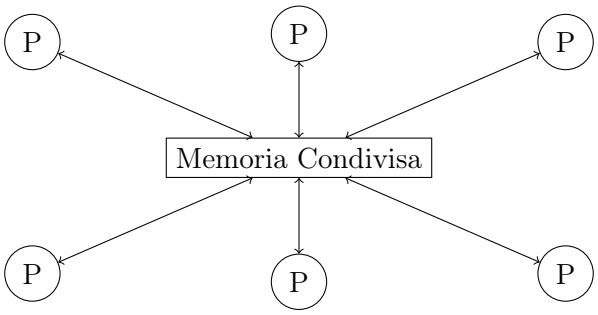


Figura 2: Modello AG concorrente asincrono distribuito.

da elaborare da AG indipendenti. Poiché ogni AG lavora su una popolazione più piccola, il risultato può essere previsto di qualità inferiore. Sebbene questo svantaggio, in una certa misura, possa essere evitato scambiando i migliori individui tra le sottopopolazioni, certamente innescherà un eccessivo traffico di comunicazione. Hanno studiato diverse alternative di implementazione AG per computer a memoria distribuita:

- **Centralizzato**: Questo approccio è simile all'approccio master-slave sincrono di Grefenstette. La differenza è che il master esegue solo la parte di replicazione, mentre gli slave fanno le operazioni genetiche localmente e le valutazioni delle funzioni. Riconoscono che questo approccio non è pratico per diverse ragioni:
  - il processore master rappresenta un collo di bottiglia nel sistema,
  - una grande porzione del calcolo è sequenziale (la fase di replicazione),
  - la granularità del processo è troppo fine, generando comunicazione eccessiva.

  Quindi, propongono diverse misure per ridurre l'impatto di questi problemi:

  1. gli slave gestiscono sottopopolazioni più grandi;
  2. gli slave calcolano un certo numero di generazioni (in cui eseguono anche replicazioni sulle sottopopolazioni che ottengono). Il master continua ad essere responsabile dell'esecuzione della fase di replicazione sull'intera popolazione, ma avverrà meno frequentemente.
  3. Inoltre, riconoscono che il master rimane inattivo mentre gli slave fanno il calcolo, e quindi, assegnare al master di lavorare su un sottoinsieme più piccolo e non sovrapposto della popolazione subito dopo l'invio delle sottopopolazioni agli slave è un altro perfezionamento che può essere fatto.

- **Semi-distribuito**: L'implementazione precedente può essere ulteriormente perfezionata avendo diverse implementazioni master-slave che lavorano su cluster non sovrapposti di processori. L'idea è di avere un cluster di processori che lavorano come nell'implementazione precedente, e a una frequenza desiderata scambiare alcuni dei migliori individui tra i cluster di processori.

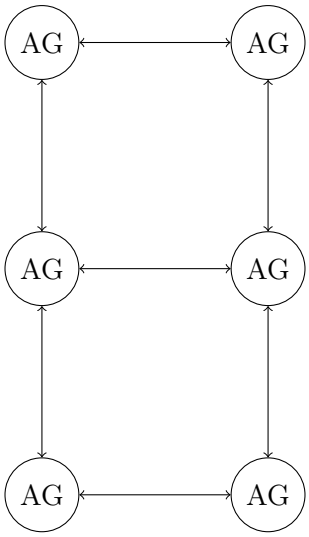


Figura 3: Modello AG a rete.

- **Distribuito**: Questo approccio è identico al modello AG a rete di Grefenstette.
- **Totalmente distribuito**: Questo approccio è una variante dell'approccio distribuito in cui la comunicazione avviene solo durante la fase di inizializzazione e terminazione.

Bianchini e Brown hanno concluso che le implementazioni con un certo livello di centralizzazione tendevano a trovare soluzioni in meno generazioni rispetto alle implementazioni distribuite. In termini di tempo di esecuzione, l'implementazione centralizzata e semi-distribuita aveva prestazioni paragonabili alle altre implementazioni su grandi numeri di processori. Le prime, tuttavia, non scalano così bene come le ultime.

Muhlenbein et al descrivono diverse applicazioni del loro algoritmo genetico parallelo. L'algoritmo è una variante del modello AG a rete di Grefenstette. La differenza è che ogni processo può migliorare la fitness dei suoi individui eseguendo processi di scalata locale. Inoltre considerano una popolazione come individui che vivono in un mondo 2-D con relazioni di vicinato ristrette. Pertanto, le operazioni AG sono eseguite rispetto ai vicini.

Diversi altri lavori sono anche considerati come la parallelizzazione di AG sequenziali esistenti. Inoltre, altri lavori sono attualmente in corso presso la Michigan State University e presso il Naval Research Laboratory per parallelizzare GENESIS di Grefenstette. Il lavoro svolto presso il Naval Research Laboratory è eseguito da DeJong, Grefenstette e van Lent. L'AG parallelo è mirato al computer basato su ipercubo, la Connection Machine, CM-200 ed è programmato in C*. Il risultato, che si chiama PARAGENESIS, è un tentativo di migliorare le prestazioni il più possibile senza cambiare il comportamento dell'algoritmo genetico. A differenza dei modelli di equilibri punteggiati e selezione locale, PARAGENESIS non modifica l'algoritmo genetico per essere più parallelizzabile poiché queste modifiche possono alterare drasticamente il comportamento dell'algoritmo. Invece ogni individuo è posto su un processore separato permettendo l'inizializzazione, la valutazione e la mutazione di essere completamente parallele. I costi del controllo globale e della comunicazione nella selezione e nel crossover sono presenti ma minimizzati. In generale PARAGENESIS su un CM-200 8k sembra funzionare 10-100 volte più velocemente di GENESIS su una SUN SPARCSTATION2 e trova soluzioni equivalenti. Le soluzioni non sono identiche solo perché il generatore di numeri casuali parallelo dà un diverso flusso di numeri. PARAGENESIS include tutte le caratteristiche di GENESIS seriale più le aggiunte che includono la capacità di raccogliere statistiche di temporizzazione, selezione probabilistica, crossover uniforme e selezione locale o di vicinato.

I ricercatori della Michigan State University hanno sviluppato due versioni di GENESIS che funzionano su computer paralleli. La prima era un port all'architettura Butterfly. Funzionando su una versione TC2000 (Motorola 88000) o GP1000 (Motorola 68020), mostrava uno speedup approssimativamente lineare dei problemi, con il TC2000 che funzionava circa 5 volte più velocemente (o richiedeva 1/5 dei nodi).

Come con PARAGENESIS, il lavoro non ha cambiato affatto il codice in GENESIS, ma ha semplicemente fornito un corretto wrapping per passare ogni valutazione di un cromosoma a un processore diverso. In termini di esecuzione su una SUN SPARCstation I, mostrava circa 25 volte speedup su problemi di significato. I test sono eseguiti sull'applicazione di AG alla classificazione di pattern e data-mining. Tuttavia, la serie Butterfly è un'architettura morente, quindi è stata tentata un'altra implementazione. Questo lavoro è unico, in un modo che è implementato usando p4, un linguaggio macro che permette al codice di essere scritto in modo tale che possa essere usato su molte architetture parallele, ma soprattutto può essere usato in un'architettura distribuita. L'architettura distribuita è attraente per un numero di ragioni, principalmente perché nodi processori più veloci possono essere ottenuti senza richiedere hardware parallelo specializzato. Invece, una rete di workstation può essere usata come processore parallelo. Quindi, una rete di SUN SPARCSTATION 2 è usata dove ogni SPARCSTATION agisce come un nodo processore parallelo.

Tutti gli sforzi sopra citati sono specificamente mirati a computer MIMD, e solo il lavoro di Herrmann e Reczko è progettato per una macchina SIMD. Nella loro implementazione, un elemento di elaborazione del computer parallelo (in questo caso MASPAR MP1 contenente 16384 elementi di elaborazione) è responsabile dell'esecuzione della valutazione della funzione di un individuo. Il resto del processo è fatto centralmente nel computer front-end del sistema. Poiché questo approccio lavora su una singola popolazione, e quindi è centralizzato, questo approccio è più attraente delle implementazioni basate su rete di Grefenstette. Sebbene questo tentativo induca chiaramente un'enorme quantità di traffico di comunicazione, gli autori affermano che il tentativo ha raggiunto 430 volte lo speedup in confronto alle prestazioni di esecuzione di AG su SUN SPARCSTATION ELC. Una vasta quantità di traffico di comunicazione è richiesta, tuttavia, per distribuire tutti gli individui nei nodi di elaborazione e quindi raccogliere il valore di fitness valutato di nuovo al front-end in ogni iterazione.

4 Algoritmi Genetici Paralleli: un Approccio SIMD

Ricordiamo il fatto osservato da Bianchini e Brown che gli algoritmi genetici paralleli che hanno un certo livello di centralizzazione performano meglio dei loro rivali. Inoltre dal fatto che ogni individuo nella popolazione esegue processi identici, è possibile ragionare che la macchina SIMD è più appropriata alla parallelizzazione degli algoritmi genetici. Assegnando ogni individuo a un processore SIMD, come nel lavoro di Herrmann e Reczko, e modificando le altre operazioni AG per minimizzare la quantità di comunicazione, possiamo ottenere un efficiente algoritmo genetico parallelo centralizzato.

Siamo ora pronti a definire una nuova implementazione parallela degli algoritmi genetici. Questo approccio assegna ogni individuo a un processore diverso e esegue l'intero processo genetico simultaneamente. Un semplice algoritmo genetico consiste in tre operatori che manipolano la popolazione di stringhe: (1) Riproduzione, (2) Crossover e (3) Mutazione, che continuamente manipolano la popolazione di stringhe fino a quando le condizioni di terminazione sono soddisfatte. Prima che queste fasi di manipolazione siano eseguite, un processo di inizializzazione come l'impostazione delle stringhe iniziali su ogni processore deve essere eseguito. Inoltre, un processo per valutare la fitness degli individui è anche necessario durante la fase di manipolazione. Pertanto, per parallelizzare gli Algoritmi Genetici dobbiamo discutere il parallelismo dei cinque compiti menzionati sopra.

4.1 Fase di inizializzazione

Questa fase inizia generando una stringa casuale su ogni processore. Le caratteristiche della stringa come la sua lunghezza, l'insieme di alfabeti usati e altre limitazioni sono specificate dal problema. Il numero di processori utilizzati rappresenta la dimensione della popolazione, p. Un algoritmo che descrive il processo sopra può essere rappresentato come:

```
Inizializzazione
begin
    Inizializzare parametri necessari per il problema,
    cio\‘e variabili della funzione di fitness
    for each processor
        call permute(string)
end Inizializzazione
```

L'unico possibile collo di bottiglia di questa fase è il processo I/O che potrebbe essere eseguito durante l'inizializzazione dei parametri. Altrimenti, lo speedup ottenuto da questa fase è proporzionale alla dimensione della popolazione (numero di processori).

4.2 Calcolo del valore della funzione obiettivo

Questa fase comporta il calcolo della fitness localmente, cioè in parallelo. La funzione utilizzata per calcolare il valore di fitness è fornita come parte del problema. Poiché l'intera fase è eseguita localmente, lo speedup ottenuto da questa fase è uguale alla dimensione della popolazione.

4.3 Fase di riproduzione

Proponiamo un nuovo processo di riproduzione. Lo schema utilizzato è basato sulla fase di riproduzione del semplice algoritmo genetico di Goldberg, che consiste nel raccogliere valori di fitness, classificarli e scegliere stringhe per la nuova popolazione basata sulla loro fitness. Una modifica a questo schema è necessaria poiché è molto orientata sequenzialmente.

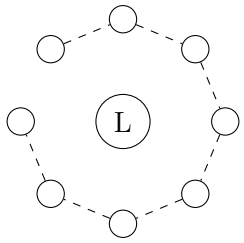
Innanzitutto, possiamo raggruppare i processori paralleli in gruppi di processori vicini. Il cluster di processori contenente stringhe è quindi chiamato una comunità. La fase di riproduzione è eseguita localmente sui cluster. La forma e la dimensione della comunità possono essere configurate dinamicamente secondo le caratteristiche del computer parallelo. Per esempio, se l'algoritmo genetico parallelo è implementato sulla famiglia di computer paralleli Maspar-MP1, la forma della comunità sarà come mostrato nella Figura 4. In questa implementazione, ogni comunità consiste di nove processori con il processore al centro che agisce come leader della comunità.

La Figura 4 illustra una comunità di processori che risiedono in un mondo 2-D (a volte questo è riferito come una topologia a mesh x x y). La dimensione della comunità, n, è uguale al numero di processori, che nella Figura 4 è nove. La forma della comunità può variare a seconda dell'architettura parallela impiegata.

La dimensione di una comunità può essere variata. Man mano che la dimensione della comunità si riduce, l'overhead di comunicazione si abbassa corrispondentemente. Mentre aumentare la dimensione della comunità all'estremo, cioè la dimensione della comunità è uguale alla dimensione della popolazione, significa che arriviamo di nuovo agli algoritmi genetici semplici originali come nel libro di Goldberg.

Il ruolo del leader è raccogliere i valori di fitness da tutti i membri della comunità, classificarli e generare da questo una nuova popolazione, così come performare come un membro ordinario a pieno diritto. Questo processo può essere rappresentato nella forma di un algoritmo:

```
Riproduzione
```



Una comunità
L = leader

Figura 4: Una comunità come implementata sulla famiglia Maspar-MP1.

```
begin
    calcolare la fitness usando il metodo discusso
    sotto il titolo precedente
    for each community leader
        raccogliere il valore di fitness dai membri
        classificarli
        generare una nuova popolazione
end Riproduzione
```

Questa fase comporta costi di elaborazione e comunicazione come segue. Innanzitutto, ogni processore deve calcolare la funzione di fitness e inviarla al leader della comunità. Per una comunità di dimensione nove, come nella Figura 4, questo processo richiede otto trasmissioni. In secondo luogo, il processo di creazione di una nuova popolazione comporta la trasmissione di stringhe da un processore all'altro. Questa trasmissione è necessaria per simulare il processo di duplicazione delle stringhe in un algoritmo genetico sequenziale.

Consideriamo una comunità con la seguente popolazione:

Processore #	Contenuti
1	stringa 1
2	stringa 2
3	stringa 3
4	stringa 4
5	stringa 5
6	stringa 6
7	stringa 7
8	stringa 8
leader	stringa 9

Dopo una fase di riproduzione arbitraria, la prossima popolazione decisa dal leader consistente di stringhe: 1, 5, 7, 2, 9, 4, 7, 2, 3. Questo significa che le stringhe 6 e 8 non sono selezionate per l'inclusione nella nuova popolazione, e le stringhe 2 e 7 producono due discendenti ciascuna. La nuova popolazione apparirà così:

Processore #	Contenuti
1	stringa 1
2	stringa 2
3	stringa 3
4	stringa 4
5	stringa 5
6	stringa 2
7	stringa 7
8	stringa 7
leader	stringa 9

Poiché le nuove popolazioni sono create da vecchie stringhe selezionate, allora i trasferimenti di stringhe sono eseguiti solo quando una stringa viene selezionata più di una volta. Nell'esempio sopra, la riproduzione necessita di due trasferimenti di stringhe: il primo è trasmettere una stringa dal processore 2 al processore 6, e il secondo è trasmettere una stringa dal processore 7 al processore 8. Con questa disposizione, al massimo dobbiamo propagare una stringa a 8 processori (nel caso estremo in cui scegliamo una stringa 9 volte, questo accade solo quando una stringa ha un notevole valore di fitness rispetto alle altre stringhe), e nel caso migliore non facciamo alcun trasferimento di stringhe (quando tutte le vecchie stringhe sono scelte).

Per i processori che risiedono in un mondo 2-D, l'intero processo richiede:

$$\text{ritardo di riproduzione} \approx \text{ritardo di polling} + \text{ritardo di distribuzione}$$
$$\text{ritardo di polling} = (n-1) \times \text{tempo di comunicazione} \times \text{distanza}$$
$$\text{ritardo di distribuzione} = i \times \text{lunghezza stringa} \times \text{tempo di comunicazione} \times \text{distanza}$$

dove: $0 \leq i \leq n-1$
n = dimensione comunità

4.4 Fase di crossover

Il crossover è fatto selezionando casualmente coppie di stringhe dalla popolazione. Se la dimensione della popolazione è dispari, allora ignorare la permutazione non accoppiata. Per ogni coppia di stringhe, selezionare il sito di incrocio e eseguire lo scambio posizionale di simboli che appaiono nel sito di incrocio della prima stringa con i simboli che appaiono nel sito di incrocio della seconda stringa.

In questo articolo riorganizziamo questo processo in modo che possa massimizzare il grado di parallelismo. Sotto questa disposizione, l'accoppiamento è fatto solo tra una coppia di stringhe che sono in una direzione. Quindi, sul mondo 2-D, questo si traduce in otto possibilità di accoppiamento, come raffigurato nella Figura 4. La decisione di scegliere una delle diverse possibili direzioni è fatta centralmente, e le stringhe sono quindi scambiate tra i processori simultaneamente. Le stringhe che sono adiacenti l'una all'altra sono di solito usate per minimizzare la distanza dei trasferimenti di stringhe sostenuti. Un algoritmo che rappresenta il processo sopra:

```
Crossover
begin
    scegliere la direzione casualmente
    for each alternate processor (primo processore della coppia)
        selezionare il sito di incrocio chiamarlo Start e End
        inviare String[Start..End], cos\‘i come Start e End
        al secondo processore della coppia nella direzione
        definita sopra
```

```
    end parallel for
    for each alternate processor (secondo processore della coppia)
        inviare String[Start..End] al primo processore
        della coppia nella direzione inversa definita sopra
    end parallel for
    for each pair of processors
        Child := String[1..Start-1] + Received String[Start..End]
                + String[End+1..]
    end parallel for
    rendere la nuova popolazione (figli) la popolazione corrente
end Crossover
```

Il processo per formare una nuova stringa di questa fase comporta la trasmissione di dati da un processore all'altro. Assumendo che il processo di comunicazione possa essere eseguito in parallelo, allora il tempo trascorso per questa fase è proporzionale al tempo di propagazione di j-byte di dati tra due processori adiacenti, dove j è i byte massimi da trasmettere. Lo speedup che possiamo ottenere da questa fase è proporzionale alla dimensione della popolazione.

4.5 Fase di mutazione

La fase di mutazione è eseguita occasionalmente con una probabilità di a, dove a si trova nell'intervallo [0..1]. Durante questa fase una posizione casuale in una stringa è alterata. L'algoritmo adatto per il processo è:

```
Mutazione
begin
    for each processor
        generare numero casuale per decidere se eseguire
        questa fase o no
        if deve essere eseguita
            selezionare una posizione nella stringa da alterare
            eseguire l'alterazione
        end if
    end parallel for
end Mutazione
```

Poiché l'intero processo di questa fase è eseguito localmente, lo speedup sostenuto da questa fase è proporzionale alla dimensione della popolazione.

Lo speedup complessivo ottenuto dall'approccio è la somma di tutto il tempo speso in ogni fase. Inoltre, le condizioni di terminazione usate nella versione sequenziale degli algoritmi genetici possono essere applicate per la versione parallela con un cambiamento minore nell'implementazione. Il cambiamento è necessario per accelerare il compito.

Ricordiamo che l'AG è terminato ogni volta che una di queste condizioni è soddisfatta: (1) se tutte le stringhe sono identiche, o (2) se il numero di iterazioni supera un certo limite superiore. Quest'ultima può essere implementata in modo diretto. La prima condizione necessita di qualche modifica, altrimenti il processo diventerà un collo di bottiglia. Confrontare le stringhe da un processore all'altro per verificare se sono identiche è un processo sequenziale, quindi è necessaria una modifica per rendere il processo parallelizzabile. Nel nostro approccio, invece di eseguire il confronto globalmente, il processo è fatto localmente in ogni comunità. Questo processo richiede $O(\log_2$ dimensione comunità) trasferimenti di stringhe, cioè per una comunità consistente di nove processori che risiedono in un mondo 2-D il numero di trasferimenti richiesti è quattro. La Figura 5 illustra il processo di confronto delle stringhe di una comunità in modo passo-passo. Poiché

questo processo può essere eseguito da ogni comunità simultaneamente, il costo totale di questi processi è $O(\log_2$ dimensione comunità$)$.

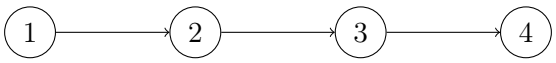


Figura 5: Confronto delle stringhe.

Se tutte le comunità riportano membri omogenei, allora una stringa di comunità viene confrontata con le loro comunità vicine. Ma questo processo aggiuntivo è eseguito molto raramente, probabilmente solo una volta per l'intero tempo di esecuzione. Pertanto questo costo può essere trascurato dal costo totale.

Ora possiamo calcolare lo speedup ottenuto dalla parallelizzazione dell'algoritmo genetico. Come nostro riferimento, un algoritmo genetico sequenziale richiede il seguente tempo per compiere il compito:

$$\text{Tempo trascorso} = \text{Inizializzazione} + l \cdot \begin{pmatrix} \text{Calcolo Fitness}+ \\ \text{Riproduzione}+ \\ \text{Crossover}+ \\ \text{Mutazione}+ \\ \text{Controllo Omogeneità} \end{pmatrix}$$

dove l è il numero di iterazioni. Ricordiamo dalle discussioni precedenti, l'algoritmo genetico parallelo richiede il seguente tempo per finire l'esecuzione:

$$\text{Tempo trascorso} = \frac{1}{m} \cdot \text{Inizializzazione} + l \cdot \begin{pmatrix} \frac{1}{m} \cdot \text{Calcolo Fitness}+ \\ ((n-1) \cdot \text{trasferimenti interi} + i \cdot \text{trasferimenti stringhe})+ \\ (2 \cdot j \cdot \text{trasferimenti byte})+ \\ \frac{1}{m} \cdot \text{Mutazione}+ \\ O(\log_2 n) \end{pmatrix}$$

dove m è la dimensione della popolazione
n è la dimensione della comunità e $m > n > 1$
i è il numero di trasferimenti di stringhe durante la fase di riproduzione, e
j è il numero massimo di byte trasmessi durante la fase di crossover.
Tempo trascorso $\approx O(l \cdot n)$

Dall'equazione data sopra, è facile vedere che la parallelizzazione può raggiungere uno speedup lineare entro una costante.

5 Valutazione delle Prestazioni

L'approccio sopra è implementato su DECmpp 12000, un computer parallelo della famiglia MASPAR-MP1. Per questo tipo di computer, i processori sono disposti in una griglia 2-D. Ogni processore è collegato ai suoi immediati 8 vicini tramite xnet, così come interconnesso ad altri processori tramite il router globale. Quindi, nella fase di riproduzione, una comunità con una forma che assomiglia a quella illustrata nella Figura 4 è usata. Xnet è usato come mezzo per trasportare i valori di fitness dai membri della comunità ai loro rispettivi leader. Inoltre, xnet è anche usato per trasferire i siti di incrocio durante la fase di crossover.

L'AG parallelo qui è usato per allocare documenti di un sistema di recupero informazioni in un sistema a memoria distribuita (Distributed-Memory Document Allocation Problem - DDAP). Il sistema di recupero informazioni raggruppa la collezione di documenti secondo un certo metodo, in modo tale che ci sia una probabilità che i documenti appartenenti a un cluster saranno acceduti simultaneamente da una query. Certamente, i documenti appartenenti allo stesso cluster sono

strettamente correlati. Questa relazione può essere definita sugli argomenti, i loro soggetti o qualsiasi altra relazione predefinita.

Formalmente, il DDAP può essere definito come segue:

Sia: X_i, $0 \leq i \leq n-1$, un nodo nel sistema distribuito,
D_i, $0 \leq i \leq d-1$, un documento nel sistema di recupero informazioni,
C_i, $0 \leq i \leq c-1$, un cluster dei documenti;
A_{ij}, $0 \leq i \leq c-1$, $0 \leq j \leq n-1$, il numero di documenti del cluster C_i allocati a X_j;
$T(C_i)$ il costo di elaborazione di una query relativa al cluster i, e
$F(p, c)$ il costo totale di elaborazione delle query relative a tutti i cluster che coprono il costo di elaborazione, p, e il costo di comunicazione, c;
Il problema è trovare A_{ij}, $0 \leq i \leq c-1$, $0 \leq j \leq n-1$, tale che $F(p, c)$ sia minimizzato.

Uno schema di clustering speciale in forma di (D, C, x, y) è usato dove:

- D è il numero di documenti,
- C è il numero di cluster, e
- x percentuale di cluster contiene y percentuale di documenti

Quattro diverse topologie di rete sono state usate per testare le prestazioni dell'algoritmo: architetture ipercubo a 8 nodi e 16 nodi, una configurazione a mesh 1x4 e una configurazione a mesh 1x4 leggermente modificata (vedi Fig. 6).

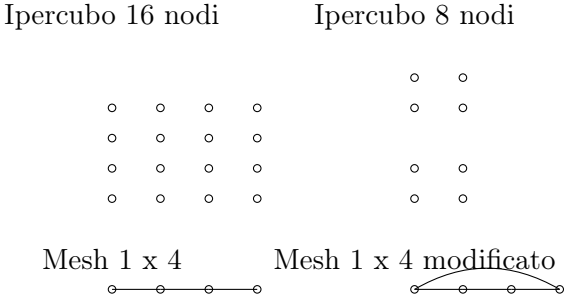


Figura 6: Varie topologie di rete usate nell'esperimento

I risultati ottenuti da questa implementazione sono stati espressi in forma di $(a + b\beta)$, dove a e b corrispondono agli elementi delle funzioni obiettivo: il costo di comunicazione e il costo di elaborazione. Mentre la costante β rappresenta la velocità relativa del processo di comunicazione nel sistema rispetto alla velocità di elaborazione. Gli esperimenti sono stati eseguiti su due insiemi di documenti che entrambi consistono di 64 documenti raggruppati in 8 gruppi. Il primo insieme, denotato da $(64, 8, x, x)$, è ottenuto distribuendo uniformemente i documenti tra i cluster disponibili, cioè ogni cluster contiene 8 documenti. Il secondo insieme, denotato da $(64, 8, 25, 50)$ è stato creato mettendo il 50% dei documenti nei primi 25% cluster, cioè i primi due cluster contengono 16 documenti ciascuno, mentre il resto dei documenti è distribuito uniformemente tra il resto dei cluster. La Tabella 1 presenta i risultati dell'applicazione dell'AG nelle varie topologie di rete usate negli esperimenti, dove ciascuna è stata testata sui due insiemi di documenti.

	$(64, 8, x, x)$	$(64, 8, 25, 50)$
Ipercubo 16 nodi	$8 + 27\beta$	$10 + 29\beta$
Ipercubo 8 nodi	$12 + 28\beta$	$11 + 29\beta$
Mesh 1 x 4	$16 + 48\beta$	$30 + 36\beta$
Mesh 1 x 4 Modificato	$16 + 32\beta$	$21 + 33\beta$

Tabella 1: Il costo minimo di allocazione dei documenti su varie topologie

Le Tabelle 2 e 3 rappresentano i risultati dell'esecuzione dell'AG per DDAP su un computer IBM RS 6000. Le tabelle illustrano il tempo trascorso dell'AG e il numero medio di iterazioni richieste per trovare un'allocazione per due topologie campione. Il numero di iterazioni rappresenta il numero medio di iterazioni necessarie per terminare l'esecuzione dell'algoritmo. L'algoritmo è stato terminato quando una delle condizioni sotto è stata soddisfatta:

1. Il numero di iterazioni supera 50.000
2. Tutte le stringhe nella popolazione sono identiche.

	dimensione popolazione = 30	dimensione popolazione = 50
Ipercubo 8 nodi	1:02.03	13:38.36
Mesh 1 x 4 Modificato	1:02.13	21:03.46

Tabella 2: Tempo trascorso per documenti $(64, 8, x, x)$ e probabilità di mutazione uguale a 0.001

	dimensione popolazione = 30	dimensione popolazione = 50
Ipercubo 8 nodi	491	3973
Mesh 1 x 4 Modificato	510	5681

Tabella 3: Numero di iterazioni richieste per documenti $(64, 8, x, x)$ e probabilità di mutazione uguale a 0.001

L'insieme di documenti usato per testare l'algoritmo genetico parallelo per DDAP è lo schema di insieme di documenti $(64, 8, x, x)$. L'algoritmo è usato per allocare questi documenti su ipercubo a 8 nodi e sulle topologie mesh modificate. Qui, due dimensioni di popolazione sono usate: (1) dimensione popolazione di 50; (2) dimensione popolazione di 30. La probabilità di mutazione in questi esperimenti è mantenuta costante. Sebbene la probabilità di mutazione sia nota giocare un ruolo nel determinare la qualità della soluzione e influenzare il numero di iterazioni, focalizziamo la nostra attenzione sulla variazione della dimensione della popolazione.

Le Tabelle 4 e 5 contengono i risultati ottenuti dall'esecuzione dell'algoritmo genetico parallelo per DDAP. Si noti che le dimensioni della popolazione nell'algoritmo genetico parallelo sono leggermente più grandi di quelle sequenziali. Le dimensioni sono l'esatta moltiplicazione di 9, la dimensione della comunità. Può essere visto confrontando queste tabelle con le tabelle precedenti che l'algoritmo genetico parallelo ottiene uno speedup lineare rispetto all'algoritmo genetico ordinario. Tuttavia, come ci si potrebbe aspettare, il numero di iterazioni richieste per completare lo stesso compito è leggermente più alto sugli algoritmi genetici paralleli. Questo viene come risultato della localizzazione manifestata dalla comunità. Come discusso precedentemente, rispetto alla versione sequenziale, qualsiasi svolta raggiunta dagli individui nell'algoritmo genetico parallelo non può essere propagata così velocemente. Invece questo effetto sarà condiviso tra i membri della comunità, che nel prossimo turno passano questo alla comunità adiacente, e così via. È necessario uno studio ulteriore sul comportamento dell'algoritmo genetico riguardo alla localizzazione. Inoltre, quando viene fatta un'osservazione attenta, il tempo richiesto dall'algoritmo genetico parallelo per completare il compito è direttamente influenzato dal numero di iterazioni eseguite. Questa caratteristica desiderabile dell'algoritmo genetico parallelo ci permette di usare dimensioni di popolazione molto grandi senza doverci preoccupare dell'overhead risultante dall'uso di dimensioni di popolazione più grandi.

6 Conclusione

In questo articolo abbiamo discusso un metodo per parallelizzare gli algoritmi genetici. Il metodo segue il paradigma di programmazione parallela SIMD. Viene fornito sufficiente background, così

	dimensione popolazione = 36	dimensione popolazione = 54
Ipercubo 8 nodi	00:03.49	00:19.65
Mesh 1 x 4 Modificato	00:03.51	00:26.06

Tabella 4: Tempo trascorso per documenti $(64, 8, x, x)$ e probabilità di mutazione uguale a 0.001 su Algoritmi Genetici Paralleli

	dimensione popolazione = 36	dimensione popolazione = 54
Ipercubo 8 nodi	525	4398
Mesh 1 x 4 Modificato	538	6252

Tabella 5: Numero di iterazioni per documenti $(64, 8, x, x)$ e probabilità di mutazione uguale a 0.001 su Algoritmi Genetici Paralleli

come la discussione sugli sforzi precedenti nella parallelizzazione degli algoritmi genetici, prima che ci impegniamo a parallelizzare gli algoritmi genetici usando uno stile SIMD. Gli algoritmi genetici lavorano su una popolazione di stringhe e ogni stringa nella popolazione subisce processi identici in ogni iterazione. Quindi, eseguire questi processi identici simultaneamente sull'intera popolazione può essere considerato come il modo naturale per parallelizzare gli algoritmi genetici.

Inoltre, Bianchini e Brown hanno sottolineato che le implementazioni con un certo livello di centralizzazione della popolazione tendono a trovare soluzioni in meno generazioni rispetto alle implementazioni distribuite. Gli approcci MIMD di solito dividono la popolazione in diverse sottopopolazioni e impiegano algoritmi genetici indipendenti per lavorare sulle sottopopolazioni. Chiaramente, gli approcci MIMD limitano gli algoritmi genetici dall'uso di informazioni sufficienti per risolvere il problema. Naturalmente, possiamo aumentare la dimensione delle sottopopolazioni in questo approccio, ma tale parallelizzazione è allora senza valore poiché non si ottiene alcun guadagno di prestazioni dallo sforzo.

Come in qualsiasi altra applicazione basata su parallelo, l'algoritmo genetico parallelo deve soddisfare questi obiettivi: (1) massimizzare la quantità di parallelismo, e (2) minimizzare la quantità di traffico di comunicazione. Di conseguenza, alcune operazioni genetiche devono essere modificate. Viene fornita una discussione dettagliata delle operazioni dell'algoritmo genetico parallelo, evidenziando le modifiche apportate all'AG originale, così come le prestazioni attese. Infine, l'algoritmo genetico parallelo è usato per trovare un'allocazione ottimale dei documenti. Vengono prese alcune misure di prestazione e sono confrontate con le rispettive prestazioni sequenziali. È stato anche dimostrato che tale parallelizzazione dimostra uno speedup lineare.

Bibliografia

Riferimenti bibliografici

[1] Anderson, E.J. and M., F., "Parallel Genetic Algorithms in Optimization," In Proceedings of the 4th SIAM Conference on Parallel Processing for Scientific Computation, 1989.

[2] Bianchini, R. and Brown, C., "Parallel Genetic Algorithms on Distributed-Memory Architectures," The University of Rochester, Computer Science Department, no. 436, Rochester, New York, Aug 1992.

[3] Brown, D., Huntley, C., and Spillane, A., "A Parallel Genetic Heuristic for the Quadratic Assignment Problem," In Proceedings of the 3rd International Conference on Genetic Algorithms, Schaffer, J. (Editor), Morgan Kaufmann, San Mateo, CA, 1989, pp. 406–415.

[4] Butler, R. and Lusk, E., "User's Guide to the p4 Programming System," Argonne National Laboratory, no. ANL-92/17, Argonne, IL, Oct 1992.

[5] DeJong, K., "Adaptive System Design: A Genetic Approach," IEEE Transaction on Systems, Man, and Cybernetics, Vol. SMC-10, no. 9, Sep 1980, pp. 566-574.

[6] DECmpp12000: DECmpp System Overview Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1st, January, 1992, Part Number: AA-PMAPA-TE.

[7] Earickson, J.A., Smith, R.E., and Goldberg, D.E., "SGA-Cube: A Simple Genetic Algorithm for nCUBE 2 Hypercube Parallel Computers," The Clearinghouse for Genetic Algorithms, Department of Engineering Mechanics(TCGA Report), no. 91005, University of Alabama, Tuscaloosa, AL, May 1991.

[8] Flynn, M.J., "Very High Speed Computer," In Proceedings of the IEEE, Dec 1966, pp. 1901-1909.

[9] Fogarty, T. and Huang, R., "Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems," In Proceedings of the Parallel Problem Solving from Nature, Schwefel, H.P. and Männer, R. (Editors), Springer-Verlag, 1991, pp. 145–149.

[10] Ghazfan, D. and Srinivasan, B., "Document Allocation in Distributed Memory Information Retrieval System," In Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence, Seoul, Korea, Sep 1992, pp. 1123-1129.

[11] Goldberg, D., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Inc., 1989.

[12] Goldberg, D., "Sizing Populations for Serial and Parallel Genetic Algorithms," In Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann, 1989.

[13] Gorges-Schleuter, M., "Explicit Parallelism of Genetic Algorithms through Population Structures," In Proceedings of the Parallel Problem Solving from Nature, Schwefel, H.P. and Männer, R. (Editors), Springer-Verlag, 1991, pp. 150–159.

[14] Grefenstette, J.J., "GENESIS: A System for Using genetic search procedures," In Proceedings of the Conference on Intelligent Systems and Machines, 1984, pp. 161-165.

[15] Grefenstette, J.J., "Parallel Adaptive Algorithms for Function Optimization," Vanderbilt University, Computer Science Department, no. CS-81-19, Nashville, 1981.

[16] Herrmann, F. and Reczko, M., "Optimization of Protein Structures using Genetic Algorithms/Sequence Processing with Partially Recurrent Neural Networks," Maspar Computer Corporation, Technical Report, June 1992.

[17] Jog, P., Suh, J., and Gucht, D., "Parallel Genetic Algorithms Applied to the Travelling Salesman Problem," Indiana University, no. 314, 1990.

[18] Kroger, B., Schwenderling, P., and Vornberger, O., "Parallel Genetic Packing of Rectangles," In Proceedings of the Parallel Problem Solving from Nature, Schwefel, H.P. and Männer, R. (Editors), Springer-Verlag, 1991, pp. 160–164.

[19] von Laszewski, G. and Muhlenbein, H., "Partitioning a Graph with a Parallel Genetic Algorithm," In Proceedings of the Parallel Problem Solving from Nature, Schwefel, H.P. and Männer, R. (Editors), Springer-Verlag, 1991, pp. 165–169.

[20] Liepins, G. and Baluja, S., "apGA: An Adaptive Parallel Genetic Algorithm," Oak Ridge National Laboratory, 1991.

[21] Lusk, E. et al., Portable Programs for Parallel Processors, Holt, Rinehart and Winston, Inc., 1987.

[22] Muhlenbein, H., "Parallel Genetic Algorithms, Population Genetics, and Combinatorial Optimization," In Proceedings of the 3rd International Conference on Genetic Algorithms, Schaffer, J.D. (Editor), Morgan Kaufmann, San Mateo, CA, 1989, pp. 416-421.

[23] Muhlenbein, H., "Evolution in Time and Space-the Parallel Genetic Algorithm," In Foundations of Genetic Algorithms, Morgan Kaufmann, Rawlins, G. (Editor), Los Altos, CA, 1991.

[24] Muhlenbein, H., Schomich, H., and Born, J., "The Parallel Genetic Algorithms as Function Optimizer," Parallel Computing, Vol. 17, 1991, pp. 619-632.

[25] Pettey, C., Leuze, M., and Grefenstette, J., "A Parallel Genetic Algorithm," In Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications, Grefenstette, J. (Editor), Lawrence Erlbaum Associates, 1987, pp. 155–161.

[26] Potter, J.L., The Massively Parallel Processor, MIT Press, Cambridge, Mass., 1985.

[27] Radcliffe, N.J., Genetic Neural Networks on MIMD Computers, Ph.D. dissertation, Physics Ph.D thesis, University of Edinburgh, 1990.

[28] Sharma, R., "A Generic Machine for Parallel Information Retrieval," Information Processing and Management, Vol. 25, no. 3, 1989, pp. 223-235.

[29] Stanfill, C. and Kahle, B., "Parallel Free-Text Search on the Connection Machine System," Communications of the ACM, Vol. 29, no. 12, Dec 1986, pp. 1229-1239.

[30] Starkweather, T., Whitley, D., and Mathias, K., "Optimization Using Distributed Genetic Algorithms," In Proceedings of the Parallel Problem Solving from Nature, Schwefel, H.P. and Männer, R. (Editors), Springer-Verlag, 1991, pp. 176–185.

[31] Talbi, E.G. and Bessiere, P., "A Parallel Genetic Algorithm for the Graph Partitioning Problem," In Proceedings of the ACM International Conference on Supercomputing, ACM, Cologne, 1991.

[32] Talbi, E.G. and Muntean, T., "A Parallel Genetic Algorithm for the Processor-Processor Mapping," In Proceedings of the 2nd Symposium on High Performance Computing, North-Holland, Montpellier, France, Oct 1991.

[33] Tanese, R., "Parallel Genetic Algorithms for a Hypercube," In Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications, Grefenstette, J. (Editor), Lawrence Erlbaum Associates, 1987, pp. 177–183.

[34] Tanese, R., "Distributed Genetic Algorithms," In Proceedings of the 3rd International Conference on Genetic Algorithms, Schaffer, J. (Editor), Morgan Kaufmann, San Mateo, CA, 1989, pp. 434–440.

[35] Tanese, R., Distributed Genetic Algorithms for Function Optimization, Ph.D. dissertation, University of Michigan, Electrical Engineering and Computer Science Department, 1989.

[36] Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best," In Proceedings of the 3rd International Conference on Genetic Algorithms, Schaffer, J.D. (Editor), Morgan Kaufmann, San Mateo, CA, 1989.

[37] Whitley, D. and Starkweather, T., "Genitor II: A Distributed Genetic Algorithms," Journal of Experimental and Theoretical Artificial Intelligence, 1991.