Algoritmo de Recocido Simulado Secuencial y Paralelizado con Memoria Distribuida para el Problema de Máquinas Paralelas no Relacionadas Ponderadas

Marco Antonio Cruz-Chávez[1], Fredy Juárez-Pérez[1], José Críspin Zavala-Díaz[2], Erika Yesenia Ávila-Melgar[1]

[1]Centro de Investigación en Ingeniería y Ciencias Aplicadas – UAEM
Facultad de Contaduría, Administración e Informática - UAEM
Avenida Universidad 1001. Col. Chamilpa, C.P. 62209. Cuernavaca, Morelos, México.
{mcruz, juarezfredy,crispin_zavala,erikay}@uaem.mx

Resumen. En este artículo se comparan dos soluciones para el problema de máquinas paralelas no relacionadas con el objetivo de minimizar el tiempo total de termino ponderado ($\sum kC_i$). Se aplica Recocido Simulado clásico secuencial y paralelizado con memoria distribuida sobre un cluster de alto rendimiento. El problema es modelado como un problema de Emparejamiento Bipartita Ponderado. Se presentan resultados experimentales con Benchmarks y se evalúa la eficiencia y eficacia de los Algoritmos. Los resultados demuestran que el Algoritmo de Recocido Simulado paralelo tiene un mejor desempeño sobre el Recocido Clásico Secuencial debido a que disminuye drásticamente los tiempos de procesamiento. Ambos Soluciones presentan un alto desempeño, debido a que para todas las pruebas siempre encuentran el óptimo.

Palabras Clave. Recocido Simulado, Problema de Máquinas Paralelas no Relacionadas, Programación Lineal Entera, Emparejamiento Bipartita Ponderado, Interfase de Pase de Mensajes, Speedup.

1. Introducción

El problema general de secuenciado de *n* trabajos sobre *m* máquinas paralelas no relacionadas sin interrupciones es clasificado como un problema de tipo NP-Completo [2]. Los investigadores se han enfocado en este tipo de problemas debido a la dureza que el problema representa cuando se busca la solución óptima, además de ser de especial interés que pueda ser aplicado a la industria de la manufactura. Muchas investigaciones se han hecho en la búsqueda de algoritmos que

M.A Cruz-Chávez, J.C Zavala Díaz(Eds):CICos2009, ISBN:978-607-00-1970-8, pp. 34 - 51, 2009.

resuelvan el problema. En [7], el modelo PMPNR (acrónimo de Problema de Máquinas Paralelas no Relacionadas) ha sido usado para resolver problemas de transporte resolviéndolo con algoritmos de aproximación, demostrando un mejor rendimiento que los obtenidos mediante formulaciones de programación lineal entera. En [8], una búsqueda es realizada para buscar el total de retrasos ponderados, en el se propone el uso de un algoritmo con colonia de hormigas, el cual incorpora un operador de transferencia genética en la búsqueda local, añadiendo un excelente desempeño al algoritmo. En [9], Se presenta un método eficiente para optimizar el total de retrasos ponderados, aplicando colonia de hormigas y usando rutas de feromonas artificiales e información de la heurística, se obtiene una optimización aproximada. En [10], Se toman en cuenta los tiempos de iniciado para minimizar el número total de trabajos retrasados ponderados aplicando un algoritmo de búsqueda local interactiva. En [11], el problema es simplificado en un problema de asignación con el objetivo de reducir los tiempos de iniciado requeridos para intercambiar la producción de un tipo de producto a otro. Un algoritmo Branch and Bound es usado. En [12], se presenta un algoritmo cuello de botella, con este algoritmo el cuello de botella es identificado y la calendarización del cuello de botella es construida. Es usado para elaborar calendarizaciones con páginas de retraso que son usadas en la asignación de trabajos entre los intervalos identificados como cuellos de botella. En [13], se describe una aplicación enfocada en el análisis de una línea de soporte de procesos automática para un centro de computación, las actividades de consultas en línea del centro es modelado como un grupo de máquinas paralelas no relacionadas, así los usuarios tienen procesos como aplicaciones de ayuda en línea a los cuales llega una lluvia de trabajos. Un índice automático de Procesamiento de Lenguaje Natural es aplicado a cada trabajo en el sentido de estimar los tiempos de procesamiento dependiente de una máquina respectiva. En [14], un algoritmo de redondeo es desarrollado para buscar una calendarización en máquinas paralelas no relacionadas; este algoritmo trabaja con modelos de programación lineal, programación cuadrática y programación convexa extendida de una calendarización para optimizar el Makespan.

En el presente trabajo se muestran dos soluciones al modelo de Programación Lineal Entera ponderado (PLE) de máquinas paralelas no

relacionadas mostrado en [3]. El estudio de máquinas paralelas no relacionadas es muy importante debido a que es muy frecuente en la industria. La solución eficiente y efectiva de modelos teóricos hace una aproximación simple al estudio de modelos reales que permiten a la industria de la manufactura incrementar su desarrollo y competitividad. En este trabajo se presentan dos soluciones al PMPNR usando un algoritmo de Recocido Simulado clásico secuencial y un algoritmo de Recocido Simulado paralelo con memoria distribuida ejecutado sobre un cluster de alto rendimiento para resolverlo como un problema de Emparejamiento Bipartita Ponderado.

El artículo se divide en las siguientes secciones: La sección 1 presenta la introducción del problema y menciona algunas investigaciones hechas por la comunidad científica. La sección 2 describe el modelo de programación lineal entera para el problema PMPNR. La sección 3 explica el modelo de grafos bipartita que es usado por el algoritmo de Recocido Simulado clásico y paralelo. La sección 5 reporta la metodología de solución aplicando Recocido Simulado. La sección 6 describe el escenario de pruebas, la sección 7 muestra los resultados experimentales y la sección 8 refiere a las conclusiones.

2. Modelo de Programación Lineal Entera Ponderado.

El problema Rm || ΣkC_j [1], es un modelo de tipo NP-complete. Puede ser formulado como un problema de programación lineal entera (PLE) con una estructura especial que hace posible resolver el problema en tiempo polinomial usando un algoritmo determinista. Este modelo PLE modela el problema como un conjunto de máquinas que deben procesar un conjunto de trabajos, cada trabajo requiere una operación para terminar. Cada máquina puede procesar cualquier trabajo uno a la vez. El tiempo de procesamiento de un trabajo depende de la máquina en la cual se procese.

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kp_{ij}\, x_{ikj} \qquad (1)$$

Sujeto a:

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ikj} \qquad j = 1,\ldots,n \qquad (2)$$

$$\sum_{j=1}^{n} x_{ikj} \le 1 \qquad \begin{matrix} i = 1,\ldots,m, \\ k = 1,\ldots,n \end{matrix} \qquad (3)$$

$$x_{ikj} \in \{0,1\} \qquad \begin{matrix} i = 1,\ldots,m, \\ k = 1,\ldots,n \end{matrix} \qquad (4)$$

El problema PLE presentado en [3] es formulado como sigue: La contribución a la función objetivo en (1) de cada trabajo *j*, depende de la máquina *i* donde el trabajo *j* es procesado, así como de la posición *k* donde el trabajo es procesado en la máquina *i*, el conjunto de trabajos que se procesan en la máquina *i* es ordenado. El trabajo *j* contribuye en kp_{ij} al valor de la función objetivo. La función objetivo optimiza el tiempo total de termino ponderado (ΣkC_j). En este modelo, la restricción en (4) indica que las variables *x* toman solo el valor de 0 ó 1. Si $x_{ikj} = 1$, entonces el trabajo *j* es asignado a la máquina *i* en la *k*-th posición. Si $x_{ikj} = 0$, en caso contrario. Las restricciones en (2) aseguran que cada trabajo es asignado a una solo posición *k*. Las restricciones en (3) aseguran que cada posición *k* para cada máquina *i* es tomado por al menos un trabajo *j*. Este modelo puede ser mapeado para problemas de transporte [4] si las restricciones del conjunto (4) son reemplazadas por un conjunto de restricciones de valores no negativos, esto es, $x_{ikj} >= 0$. Esto es importante debido a que con este modelo relajado se obtendrían límites con algoritmos exactos para problemas NP-Completos como el modelo *Rm|| Cj*. En el caso del modelo con la presente estructura, la formulación relajada tiene resultados de 0 o 1 para las variables *x*. De esta manera, es posible usar un algoritmo exacto como el SIMPLEX en el sentido de tener una solución para el modelo sin necesidad de relajar las restricciones en (4).

3. Modelo de Grafos Bipartita.

Un caso especial del problema de transporte es el PEBP (acrónimo de Problema de Emparejamiento Bipartita Ponderado). El modelo *Rm* |

| ΣkC_j para la estructura definida en el modelo de PLE en la sección 2 puede ser representado como PEBP [3]. La figura 1 presenta una instancia de *n=3* trabajos y *m=2* máquinas para el problema *Rm* | | ΣkC_j usando un grafo bipartita. El grafo esta compuesto de *n* trabajos en la parte superior y *mn* posiciones en la parte inferior. Las *mn* posiciones se dan en pares (*i,k*) donde *i* es la máquina asignada para procesar el trabajo *j* y *k* es la posición dentro de la cola de la máquina *i*, donde cada máquina *i* puede procesar a lo más *n* trabajos. Si el trabajo *j* que es asignado a la posición *ik* presenta un costo kp_{ij}. El objetivo es determinar la asignación *ik* para cada trabajo *j* en el grafo bipartita que tenga el mínimo costo de tiempo total.

La interpretación se da sobre la base de la figura 1 como sigue: Para la máquina *i=1* el trabajo *j=1* puede ser procesado en primer lugar (p_{11}), segundo lugar ($2p_{11}$) o en tercer lugar ($3p_{11}$). El trabajo *j=2* puede ser procesado en primer lugar (p_{12}), segundo lugar ($2p_{12}$) o en tercer lugar ($3p_{12}$). El trabajo *j=3* puede ser procesado en primer lugar (p_{13}), segundo lugar ($2p_{13}$) o en tercer lugar ($3p_{13}$).

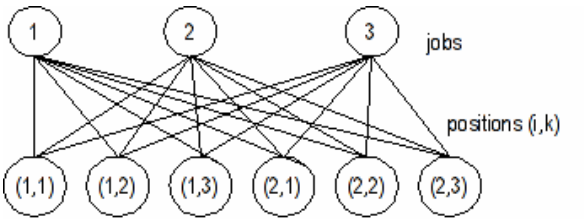

Fig. 1. Grafo Bipartita para ΣkC_j

El modelo PEBP puede ser solucionado con varias meta-heurísticas. La siguiente es una meta-heurística de Recocido Simulado mediante la cual se obtiene la solución de este artículo.

4. Algoritmo de Recocido Simulado.

Los fundamentos de la meta heurística denominada RS (acrónimo de Recocido Simulado) fueron introducidos por Kirkpatrick [5]. El propuso un método probabilístico de búsqueda local de manera que escapara de los óptimos locales.

El nombre del método proviene de su similitud con el proceso metalúrgico de Recocido lento, con el que se consigue un sólido de mínima entropía. El RS sigue un esquema similar para ayudar a resolver problemas de optimización. La figura 2 muestra el RS usado para este trabajo. El proceso de RS requiere del conocimiento de los siguientes parámetros:

s_o : Solución inicial del problema.
T : Parametro de control de RS.
r : Coeficiente de control. $0 < r < 1$.
$MCS > 0$: Cadena de Markov (Markov Chain Size).
C: Función de costo.
N: Estructura de vecindad.
FROZEN: Criterio de paro. Mínimo valor de T.

```
function SimulatedAnnealing ( )
Begin
  s:= s_o, T = T_0
    Repeat
      cycle := 1 to MCS
          generate new solution, s' = N(s)
          if C(s') <  C(s) then  s := s'
          else
            d := C( s') - C(s)
             Random ρ  (0 ,1)
            if ρ ≤ e^(-d/T) then  s := s'
          end else
        end cycle
        T := r T
    Until FROZEN
  return C(s)
End
```

Fig. 2. Algoritmo de Recocido Simulado.

En la Figura 2, al principio el RS con una temperatura alta T, acepta casi todas las transiciones entre las soluciones. Después T gradualmente disminuye debido al factor r, esto hace que la aceptación de movimientos sea cada vez más y más selectiva. Finalmente solo se aceptan aquellos movimientos que mejoren la solución actual. Un movimiento es un vecino s' de s que es obtenido de una estructura de vecindad. El costo de la función C es la función objetivo del problema a resolver y permite la evaluación de la calidad de la solución. La aceptación de los movimientos (nueva solución) es controlada por el

algoritmo de Metrópolis que involucra la función de Boltzmman. En cada ciclo de T, Metrópolis repite MCS ciclos. Para los diferentes valores de T los cuales pasan a través de una secuencia de enfriamiento, un proceso explora las mejores soluciones. El criterio de paro es considerado por FROZEN el cual es un valor mínimo al cual *T* llega y es controlado por *r*.

5. Metodología de Solución.

Para solucionar el problema $Rm \mid\mid \Sigma kC_j$ usando programación paralela con memoria distribuida, la siguiente metodología es propuesta basado en el grafo bipartita (Figura 1) y consiste en los siguientes pasos:

1. Representación simbólica del grafo bipartita.
2. Determinar la estructura de vecindad que obtenga soluciones factibles.
3. Sintonizar los parámetros de RS. *T, r, MCS y FROZEN.*
4. Aplicar la meta-heurística de RS secuencial y paralelizado.
5. Aplicar mecanismo generador de soluciones.
6. Aplicar mecanismo para recolectar soluciones en el cluster.

5.1. Representación Simbólica.

Para la representación simbólica del grafo bipartita, una estructura de datos unidimensional es usado para representar las *mn* posiciones que se dan entre las máquinas ($i=0,..., m-1$) y las k-ésimas posiciones que un trabajo *j* ($j=0,...,n-1$) puede tomar dentro de una máquina *i*. Con este estructura tendremos *n* trabajos calendarizados sobre el arreglo de tamaño *mn* y tendremos $mn - 1$ posiciones libres hacia donde se pueden mover los trabajos mediante un intercambio aleatorio. Una solución a la instancia presentada en la figura 1 es mostrada en la figura 3 usando la representación simbólica.

La estructura representa $m=2$ máquinas *i* en serie y $n=3$ trabajos *j*, lo que da un total de 6 posiciones posibles para los 3 trabajos *j*. Cada máquina *i* ocupa tres posiciones para los tres posibles trabajos *j*, por tanto la estructura representa a las máquinas 0 y 1. Las seis posibles

posiciones son referenciadas mediante un índice que va desde la posición 0 hasta la posición 5. La secuencia va desde *k=0* hasta *k=2* para representar las tres posibles posiciones dentro de la cola de la máquina *i*.

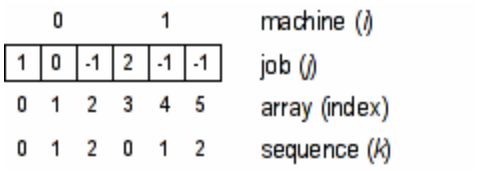

Fig. 3. Representación simbólica del problema

La interpretación de la figura 3 se da como sigue: El trabajo *j = 1* se procesa en primer lugar *k = 0* en la primera máquina *i = 0* y con una posición real *índice = 0*. El trabajo *j = 0* se procesa en segundo lugar *k=1* en la primera máquina *i = 0* y con una posición real *índice = 1* y así sucesivamente. Las posiciones dentro del arreglo que contengan *j = -1* son posiciones disponibles.

5.2. Mecanismo para Generar Soluciones en RS.

La estructura de vecindad esta basada en la representación simbólica de las posiciones que el grafo bipartita de la figura 1 representa. En este trabajo, se desarrollo un método de búsqueda local en la estructura de vecindad el cual incluye los siguientes pasos:

1. Generación de la calendarización inicial aleatoria.
2. Aplicar mapeo indirecto y directo.
3. Generar mapeo rápido.
4. Cambiar de posición un trabajo hacia una posición libre aleatoria.

5. Intercambiar dos trabajos aleatorios.

Para generar la solución inicial aleatoria, un trabajo *j* y una máquina *i* es seleccionada. El trabajo *j* es posicionado sobre la máquina seleccionada. La figura 4 representa el algoritmo que genera la solución inicial. Un conjunto de trabajos *J* y un conjunto de máquinas *M* son usados para calendarizar la solución inicial. Un trabajo *j* y una máquina *i* son seleccionados aleatoriamente del conjunto *J* y *M* respectivamente.

La posición k_i es asignada en la máquina donde no tiene un trabajo asignado (posición libre), entonces el trabajo j es almacenado en el Array (figura 3) en la posición k_i, la posición k_i es actualizada y el trabajo j es eliminado del conjunto J. El procedimiento es repetido mientras J no sea un conjunto vació.

```
function initial_solution( )
Begin
  k = {k1=0,...,km-1=0}
  M = {m1 = 0,...,mm=m-1}
  J= {0,...,n-1}
  Repeat
    j = random(J)
    i = random(M)
    Array[i * n + ki] = j
    ki=ki + 1
    J = J - {j}
  Until j0 ≠ ϕ
End
```

Fig. 4. Generación de la solución inicial.

El mapeo indirecto busca los elementos *ikj* a partir de la posición real dentro de la estructura de vecindad, la pojsición real es obtenida del índice que se utiliza para acceder al Array, las funciones son:

$$j = \text{Array}(\ index\ ) \tag{5}$$

$$i = \text{integer\_truncate}(index / n\ ) \tag{6}$$

$$k = \text{operator\_ module}\ (index,\ n) \tag{7}$$

Por ejemplo en la figura 3, usando (5), para *índice* = 3 se obtiene el trabajo 2. Con el mismo *índice* usando (6) se obtiene la máquina 1. Usando (7) la posición $k = 0$ es encontrada.

El mapeo directo encuentra la posición real dentro de la estructura de vecindad denotada por *índice* a partir de los elementos *ikj*. Las funciones son:

$$index = i * j + k \tag{8}$$

En la figura 5 presentamos un mapeo rápido a partir de la representación simbólica (figura 3), que genera una calendarización de trabajos en una estructura de tamaño *2n*.

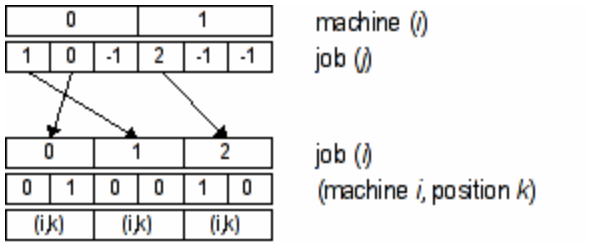

Fig. 5. Mapeo rápido

El arreglo corresponde a una calendarización (trabajos, máquinas y posiciones) menor en tamaño que la estructura de vecindad de tamaño *mn*. Esta estructura permite seleccionar un trabajo aleatoriamente y localizar su posición dentro de la estructura de vecindad en base a las funciones de mapeo descritas anteriormente. Esta estructura aumenta la eficiencia en la búsqueda de una nueva solución debido a que los mapeos directos e indirectos representan la indexación de la estructura de vecindad y agilizan el acceso y modificación de las nuevas soluciones, de la misma manera que lo haría una base de datos indexada. Como se observa en la figura 5 el ordenamiento de la estructura de vecindad es en base al número de máquinas y la estructura de mapeo rápido es en base al número de trabajos.

El movimiento de un trabajo a una posición libre del arreglo genera un vecino. Se selecciona un trabajo de forma aleatoria de la estructura de vecindad usando el mapeo rápido que contiene todos los trabajos calendarizados como se muestra en la figura 6a. Entonces se realiza la operación de transformación descrita en (8), para localizar el índice que apunta a ese trabajo en la estructura de vecindad figura 6b. Paso seguido selecciona una posición dentro de la estructura de vecindad de forma aleatoria que es apuntada por *índice* y representa el lugar hacia donde se moverá el trabajo previamente seleccionado, la figura 6c

muestra una posición libre (-1). En la figura 6d se muestra el movimiento del trabajo j = 0 de la posición 1 a la posición 4. Por ultimo en base a las operaciones de transformación descritas en (5), (6) y (7), actualizamos las posiciones del trabajo j = 0 como se muestra en la figura 6e.

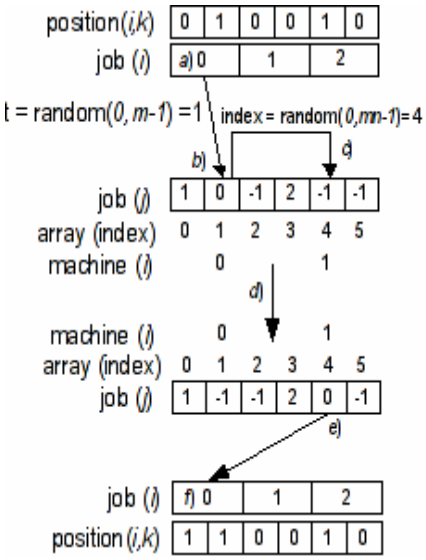

Fig. 6. Operación mueve.

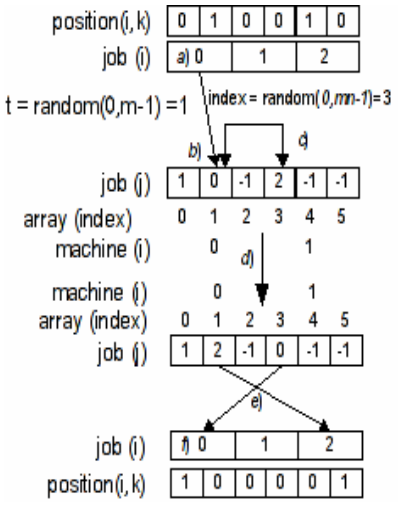

Fig. 7 Operación intercambia.

El intercambio de un par de trabajos en la solución genera un nuevo vecino. En la figura 7d se muestra el intercambio de trabajos 0 y 2 contenidos en las posiciones 1 y 3. La selección del par de trabajos es aleatoria y las operaciones de transformación son las mismas aplicadas anteriormente. Finalmente la figura 7e muestra que es necesario actualizar las posiciones de los trabajos 0 y 2 para una correcta referencia.

5.3. Mecanismo para recolectar soluciones en el cluster.

Se desarrolló una modificación al algoritmo de RS para permitir su ejecución en paralelo. Esta modificación consiste en agregar funciones para el pase de mensajes entre el proceso de control (llamado proceso

maestro) y los diferentes procesos de los nodos del cluster (llamados procesos esclavos). El número de procesos que se ejecutan en cada nodo es igual al número de procesadores que tenga cada nodo en particular, esto es, a cada proceso le corresponde un solo procesador. Conocido como MPI (Interfase de Pase de Mensajes) este estándar permite agregar paralelismo a las aplicaciones con memoria distribuida.

El algoritmo de RS paralelo maestro recolecta las soluciones obtenidas por los procesos de RS esclavos. Esto significa que cada proceso de RS esclavo envía el óptimo encontrado al proceso maestro. De esta manera el óptimo global de los RS esclavos puede ser obtenido por el RS maestro. Cuando se encuentran más de un óptimo Global igual se toma el que tiene menos tiempo de procesamiento.

En algoritmo propuesto denominado RSPCJ esta compuesto por la parte de control (figura 8a) y la parte de procesamiento numérico (figura 8b).

```
function RSProcesoMaestro ( )
Begin
   s:= s0, t:= 0
   cycle p := 1 to Procesadores
      recibe_del_procesador(p,s',t')
       if C(s') <  C(s) then  s := s, t:=t'
       if C(s') == C(s) and t > t' then  t:=t'
  end cycle
End
```

Fig. 8A Algoritmo RSPCJ parte de control.

```
function RSProcesoEsclavo( )
Begin
   s:= s0, ti:= t:=  tiempo_inicial()
  Repeat
    cycle := 1 to MCS
      ...
    end cycle
    T := r T
  Until FROZEN or s ==SIMPLEX
  t':= tiempo_final() - t
  envia_al_maestro(s,t')
End
```

Fig. 8b. Algoritmo RSPCJ procesamiento número.

Por ejemplo. En la figura 9a los *p* procesos esclavos de los *n* nodos del cluster ejecutan el RS cada uno con su propia solución inicial, en cada ciclo MCS los *p* procesos esclavos generan soluciones aleatorias y al término descienden la temperatura. Este proceso se repite hasta llegar al punto de congelación FROZEN y encontrar el óptimo dado por SIMPLEX.

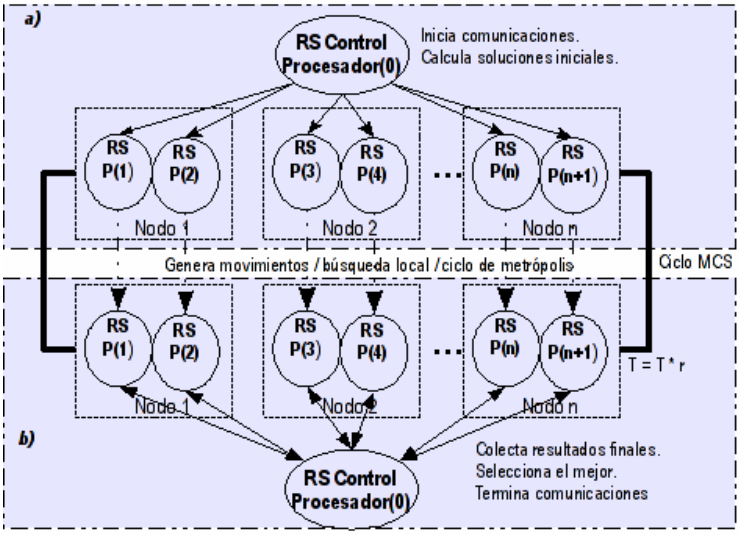

Fig.9 Algoritmo RSPCJ recolecta las soluciones

Finalmente los *p* procesadores esclavos envían el resultado final encontrado al nodo maestro. Entonces el nodo maestro escoge la mejor solución (figura 9b). En caso de encontrar resultados iguales en más de un procesador, se toma el que tenga un tiempo menor de búsqueda.

6. Escenario de pruebas

Cinco problemas de prueba fueron usados, estos benchmarks son propuestos en [15]. Cada benchmark contiene 60 trabajos y 4 máquinas paralelas no relacionadas. La información referente a los tiempos de iniciado no son tomados en cuenta ya que el modelo usado para este trabajo no los usa. Para probar la eficiencia y eficacia del algoritmo RS, un algoritmo exacto (SIMPLEX) fue desarrollado en [16] para resolver las instancias con un máximo de 60 trabajos y 4 máquinas. Con este algoritmo la solución óptima global fue obtenida. Después de obtener la solución, los algoritmos de RS secuencial y RS paralelo propuestos fueron usados para encontrar el óptimo basado en 30 ejecuciones para cada uno de los cinco benchmark propuestos.

La infraestructura utilizada fue un cluster de alto rendimiento compuesto de un nodo maestro con procesador Intel x86 de 1GB de memoria. 9 nodos de procesamiento numérico cada uno con procesador de doble núcleo Intel Celeron de 2.0 GHz y 2 Gb Memoria, haciendo un total 18 núcleos de procesamiento numérico. El software utilizado es Sistema Operativo Scientific Linux 4.7, compilador GCC 3.4.3, librerías OpenMPI 1.3.6 y sistema de archivos de red NFS.

7. Resultados Experimentales

Para todas las pruebas, el sintonizado de los valores de los parámetros del algoritmo RS fueron: $T = 25$, $r = 0.985$, $FROZEN = 0.0001$ y MCS es seis veces el tamaño de la vecindad. El sintonizado de r = 0.985 fue en el rango de ($0.8 <= r <= 0.99$) de acuerdo con [6]. El criterio de paro fue llegar al punto de congelamiento FROZEN o encontrar el óptimo dado por SIMPLEX.

En la tabla 1 se observa que para los cinco benchmark, el algoritmo de RS secuencial siempre encuentra el óptimo y el número de veces que lo encuentra es de al menos 3 veces. Muestra una desviación estándar aceptable y un error relativo por debajo del 0.02% en el peor de los casos para el promedio de 30 pruebas por cada problema. El mejor resultado obtenido por el RS secuencial fue para el benchmark 60on4Rp50Rs50_1, debido a que de las 30 pruebas el peor resultado fue de 62659 con una desviación estándar de 3.8201 y un promedio de error relativo de 0.0073%.

Tabla 1. Eficacia del algoritmo RS secuencial.

Benchmark	NP	# Pruebas	Opt.	# Veces	Mejor	Peor	σ	ER Average
60on4Rp50Rs50_1	1	30	62646	3	62646	62659	3.8201	0.0073
60on4Rp50Rs50_2	1	30	62185	6	62185	62246	13.5190	0.0190
60on4Rp50Rs50_3	1	30	62637	12	62637	62698	14.0426	0.0170
60on4Rp50Rs50_4	1	30	62973	3	62973	63036	11.9474	0.0188
60on4Rp50Rs50_5	1	30	63032	6	63032	63075	12.2476	0.0160

Para el algoritmo RS paralelo, en la tabla 2 se observa que al igual que el RS secuencial siempre encuentra el óptimo, pero el número de veces que lo encuentra es del al menos 28 veces. Muestra una desviación estándar mejor que el RS secuencial y por tanto un error relativo por debajo del 0.0003% en el peor de los casos para el promedio de 30

pruebas por cada problema. Los mejores resultados obtenidos por el RS paralelo fueron para los benchmark 60on4Rp50Rs50_2, 60on4Rp50Rs50_3 y 60on4Rp50Rs50_5.

La eficacia aumenta considerablemente debido a que el paralelismo permite evaluar un mayor número de RS. Así para una prueba del algoritmo RS paralelo se requiere 1 proceso maestro y 18 procesos esclavos, aumentando la eficiencia en 18 veces con respecto al RS secuencial.

Tabla 2. Eficacia del algoritmo RS paralelo.

Benchmark	# Proc	# Pruebas	Opt.	Mejor	# Veces	Peor	σ	ER Average
60on4Rp50Rs50_1	18	30	62646	62646	28	62648	0.4026	0.0002
60on4Rp50Rs50_2	18	30	62185	62185	30	62185	0.0	0.0
60on4Rp50Rs50_3	18	30	62637	62637	30	62637	0.0	0.0
60on4Rp50Rs50_4	18	30	62973	62973	28	62975	0.5074	0.0002
60on4Rp50Rs50_5	18	30	63032	63032	30	63032	0.0	0.00

La tabla 3 muestra los resultados de eficiencia. Para el ultimo benchmark, el algoritmo de RS paralelo muestra que el mejor resultado de las 30 pruebas reduce el tiempo de ejecución en un 27.38% con respecto al secuencial. En general se observa que el algoritmo de RS paralelo reduce el tiempo de ejecución en la búsqueda del óptimo en un 24.43% en promedio más rápido que el RS secuencial. La eficiencia del RS paralelo se ve aumentada cuando una de los 18 RS encuentra el óptimo antes del llegar al punto de congelación FROZEN, en este caso el tiempo de procesamiento se reduce y de los 18 procesos se toma el menor.

Tabla 3. Comparativa de eficiencia RS secuencial vs. RS paralelo

Benchmark	Promedio RS Secuencial 18 Proc.	Promedio RS Paralelo 18 Proc.	Optimiza Tiempo
	CPU segundos	CPU segundos	% reduce
60on4Rp50Rs50_1	70.52	2.87	26.03
60on4Rp50Rs50_2	67.02	2.90	23.07
60on4Rp50Rs50_3	64.07	2.83	22.25
60on4Rp50Rs50_4	69.75	2.97	23.45
60on4Rp50Rs50_5	68.51	2.89	27.38

El Speedup calculado para el RS paralelo es muy cercano al óptimo y corresponde a un Speedup sub-lineal como se muestra en la tabla 4. La

eficiencia calculada es en promedio del 0.999 muy próximo al ideal. El algoritmo paralelo minimiza el uso de comunicaciones lo que le permite alcanzar un excelente desempeño en eficiencia paralela.

Tabla 4. del Speedup y eficiencia del RS Paralelo.

Número de Procesadores	Tiempo uso CPU Segundos.	Speedup Óptimo	Speedup Paralelo $S_p = T_1 / T_p$	Eficiencia $E_p = S_p / P$
1	123.3	1	1	1
2	61.6	2	2	1
4	30.77	4	3.99	0.99
8	15.41	8	7.99	0.99
16	7.72	16	15.95	0.99

8. Conclusiones

La indexación de la estructura de vecindad representado por un mapeo rápido en el algoritmo de RS causa que la búsqueda del óptimo en el espacio de soluciones sea más eficiente que el método SIMPLEX en los 5 benchmark analizados. La eficacia del algoritmo RS paralelo y RS secuencial es del 100% para los benchmark analizados debido a que encuentran siempre el óptimo en todos los benchmarks. El RS paralelo muestra una mejor eficacia debido a que hace una mayor exploración y explotación del espacio de soluciones. Esto se debe a que para cada prueba se requiere 1 proceso maestro y 18 procesos esclavos ejecutándose en paralelo, lo que aumenta las posibilidades de encontrar el óptimo en 18 veces con respecto al RS secuencial. Así para las 30 pruebas existen 30x18 posibilidades de encontrar el óptimo. Las posibilidades de encontrar el óptimo en un cluster de alto rendimiento aumentan en función al número de procesadores disponibles en el.

Cuando comparamos la eficiencia de RS secuencial vs. RS paralelo se observa que el RS paralelo mejora el tiempo secuencial en un promedio de 24.43%. Esto se debe principalmente a que los 30x18 RS aumentan considerablemente las posibilidades de encontrar el óptimo antes de llegar al criterio de paro FROZEN, lo que lleva a una reducción considerable en el tiempo de búsqueda.

El algoritmo presenta un excelente desempeño en eficiencia del 0.99% y en Speedup muy próximo al óptimo debido principalmente al diseño que minimiza el uso de las comunicaciones entre los procesos del RS. Trabajos futuros se centraran en un uso más extenso de las comunicaciones a fin de hacer una mejor exploración del espacio de soluciones con el uso de pase de mensajes con MPI.

9. Referencias

1. R.L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math.,pages 287–326, 1979.
2. A.M. Garey and D.S. Johnson, Computers and intractability: A Guide to the theory of NP-completeness, Freeman, San Francisco, 1979.
3. M. Pinedo, Scheduling Theory, Algorithms and Systems, Prentice Hall, ISBN:0130281387, USA., Aug. p. 586, 2001.
4. F.S. Hillier, G.J. Lieberman, Introduction to Operation Research, 8th ed.,Mc Graw Hill , ISBN: 0073017795, 2008.
5. S. Kirkpatrick, C. Gelatt, and M. Vecchi, Optimization by Simulated Annealing. Science Vol. 220(4598): pp. 671-680, 1983.
6. R, David, R. Sixto and M. Jacinto. Simulación, alfaomega grupo editor., p. 388, 2009.
7. S. Koranne, Formulating SoC Test Scheduling as a Network Transportation Problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, december 2002.
8. H. Zhou, Z. Li and X. Wu, Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 132-136, August 18 - 21, Jinan, China, 2007.
9. L. Mönch, Heuristics to Minimize Total Weighted Tardiness of Jobs on Unrelated Parallel Machines, 4th IEEE Conference on Automation Science and Engineering Key Bridge Marriott, Washington DC, USA, pp. 572-577, August 23-26, ISBN: 978-1-4244-2022-3, 2008.
10. C. Chun.-Lung, An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2044-2047, Qingdao, China September, ISBN: 978-1-4244-2502-0, 2008.
11. C. Pessan, J.L. Bouquard, E. Néron, An Unrelated Parallel Machines Model For Production Resetting Optimization, International Conference on Service Systems and Service Management IEEE, Vol 2, ISBN: 1-4244-0450-9, pp. 1178-1182, 2007.
12. C. Chun.-Lung, C. Chuen-Lung, A Heuristic Method for a Flexible Flow Line with Unrelated Parallel Machines Problem, Conference on Robotics, Automation and Mechatronics, pp. 1-4, IEEE, 2006.
13. K. Anastasova, M. Dror, Intelligent Scheduler for Processing Help Requests on Unrelated Parallel Machines in a Computer Support Administration System, Conference on Systems, Man, and Cybernetics, IEEE, pp. 372-377, ISBN: 0-7803-4778-1, 1998.
14. V.S. Anil Kumar, M. V. Marathe, Approximation Algorithms for Scheduling on Multiple Machines, Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), pp. 254-263, ISBN: 0-7695-2468-0/05, 2005.

15. . P. Arnaut, R. Musa, G. Rabadi, Ant Colony Optimization Algorithm to Parallel Machine Scheduling Problem with Setups, 4th IEEE Conference on Automation Science and Engineering, Washington DC, USA, August 23-26, ISBN: 978-1-4244-2023-0, pp. 578-582, 2008.
16. C. Marco, J. Fredy, Y. Erika, M. Alina, Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem, The Electronics, Robotics and Automotive Mechanics Conference, 2009.