Paralelización de algoritmos de estimación de distribuciones

Alexander Mendiburu, Endika Bengoetxea, José Miguel

Resumen — **Se realiza una breve introducción a los algoritmos de estimación de distribuciones, dentro del ámbito de la computación evolutiva. Al aplicar estos algoritmos a un problema de concordancia inexacta de grafos, se observan largos tiempos de ejecución (más de 37 horas para un problema de tamaño significativo). Se busca la reducción de estos tiempos mediante la paralelización de la operación más costosa, usando una aproximación maestro / trabajadores. Se implementa esta estrategia con MPI sobre un cluster de PCs con Myrinet como red de interconexión. El resultado es una drástica reducción de los tiempos de ejecución, con un *speedup* máximo observado de 4,53.**

Palabras clave — **Algoritmos de estimación de distribuciones (EDAs), computación evolutiva, MPI.**

I. Introducción

DURANTE esta última década se ha progresado con acierto en el estudio y uso de técnicas heurísticas aplicadas a la optimización. Entre dichas técnicas, la referencia se ha centrado en la computación evolutiva (algoritmos genéticos, estrategias evolutivas, programación evolutiva y programación genética). Basado en los algoritmos genéticos, ha surgido un nuevo tipo de computación evolutiva conocido como EDAs (Estimation of Distribution Algorithms, algoritmos de estimación de distribuciones) [13] donde se generalizan los algoritmos genéticos sustituyendo los operadores de cruce y mutación por el aprendizaje y muestreo de distribuciones de probabilidad de los mejores individuos de la población en cada iteración del algoritmo.

Mientras que en los heurísticos de computación evolutiva las interdependecias entre las variables que representan a los individuos son tenidas en cuenta de manera implícita, en los EDA las interrelaciones son expresadas explícitamente a través de las distribuciones de probabilidad asociadas al conjunto de individuos seleccionados en cada iteración.

La primera población se genera de forma aleatoria y, a partir de esta, se muestrean los nuevos individuos, comenzando por una distribución de probabilidad estimada de la base de datos que contiene únicamente los individuos seleccionados en la generación anterior. De hecho, estimar la distribución de probabilidades requiere la adaptación de métodos para el aprendizaje de modelos a partir de datos, que han sido desarrollados por investigadores en el dominio de los modelos gráficos probabilísticos. El aprendizaje es, generalmente, el paso más costoso en términos de cómputo.

Departamento de Arquitectura y Tecnología de Computadores. Universidad del País Vasco / Euskal Herriko Unibertstitatea. Apdo. 649, 20080 San Sebastián. E-mail: amendiburu@si.ehu.es.

Los algoritmos EDA cumplen los siguientes pasos:

1. Se genera la primera población D_0 de M individuos. Habitualmente, dicha generación se lleva a cabo suponiendo una distribución uniforme en cada variable y posteriormente cada individuo es evaluado.
2. Se selecciona un número N ($N \leq M$) de individuos en base a un determinado criterio (generalmente son seleccionados aquellos con mejor valor).
3. Se induce el modelo probabilístico de dimensión n que mejor refleja las interdependencias entre las n variables que conforman el individuo.
4. Finalmente, se constituye la nueva población con los M nuevos individuos a partir de la simulación de la distribución de probabilidad aprendida en el paso previo.

Los pasos 2, 3 y 4 se repiten hasta alcanzar una determinada condición de parada. Como ejemplos de condiciones de parada podríamos citar las siguientes: alcanzar un determinado número de generaciones o de individuos evaluados, homogeneidad en la población generada o no conseguir tras un cierto número de generaciones un individuo con mejor valor. Es importante reseñar que, en el caso de tratarse de individuos discretos, el modelo probabilístico gráfico es una Red Bayesiana, mientras que para individuos continuos sería una red Gaussiana.

El modelo probabilístico seleccionado (dependiendo del tipo de problema a tratar) para aprender las interdependencias entre variables influirá notablemente en el comportamiento del algoritmo EDA, tanto en tiempo de ejecución como en resultados obtenidos. Por ello, creemos interesante exponer a continuación una clasificación de los modelos en función de su complejidad:

- Sin dependencias entre variables. La distribución de probabilidad se factorizará como el producto de n distribuciones de probabilidad univariables e independientes. Como ejemplo, podemos citar UMDA (Univariate Marginal Distribution Algorithm) en el supuesto discreto [12] y $UMDA_C$ en el continuo [10].
- Dependencias por parejas. La estimación se puede realizar de forma sencilla teniendo en cuenta las dependencias entre parejas de variables. En este caso, basta con estadísticos de segundo orden. A modo de ejemplo cabe citar MIMIC (Mutual Information Maximization for Input Clustering) para el caso discreto [6] y $MIMIC_C$ para el continuo [10].

- Dependencias múltiples. Se consideran en este caso todas las posibles dependencias entre variables sin importar la complejidad que ello conlleve. Ejemplos en esta categoría son EBNA (Estimation of Bayesian Networks Algorithm) en el dominio discreto [7] y EGNA (Estimation of Gaussian Networks Algorithm) en el continuo [10].

Haciendo un análisis de los porcentajes de ejecución requeridos en cada uno de los pasos del algoritmo EDA, queda probado que en los dos últimos grupos el coste derivado del aprendizaje es lo suficientemente importante como para plantear su paralelización [1]. Por lo tanto, para nuestros experimentos vamos a seleccionar una aproximación en el dominio discreto dentro del modelo de dependencias múltiples: EBNA_{BIC}. De todos modos, el lector interesado puede conseguir información completa de todas las aproximaciones en [9].

II. Algoritmo EBNA_{BIC}

Dentro de esta sección se explicará cómo se realiza la fase de aprendizaje que devuelve la estructura probabilística gráfica (Red Bayesiana) que mejor representa a los individuos. En [7] los autores utilizan el criterio BIC (Bayesian Information Criterion) [15] como índice para evaluar la bondad de cada estructura encontrada durante la búsqueda. Siguiendo dicho criterio, el valor *BIC(S,D)* para una estructura de red Bayesiana *S* construida a partir de una base de datos *D* y conteniendo *N* casos puede ser definida como:

$$BIC\ (S,D) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\ \log\ \frac{N_{ijk}}{N_{ij}} - \frac{\log N}{2}\sum_{i=1}^{n}(r_i-1)q_i$$

donde:

- n es el número de variables de la red Bayesiana.
- r_i es el número de valores diferentes que puede tomar la variable X_i.
- qi es el número de valores diferentes que los padres de X_i, Pa_i^S, pueden tomar.
- N_{ij} es el número de individuos en D en los que las variables Pa_i^S toman su j-ésimo valor.
- N_{ijk} indica el numero de casos en D en los que la variable X_i toman su valor k-ésimo y las variables Pa_i^S toman su valor j-ésimo.

Para poder obtener el mejor modelo se deberá realizar una búsqueda a través de todas las posibles estructuras, pero esta demostrado que se trataría de un problema NP-completo [5], por lo que nos vemos en la obligación de utilizar algoritmos sencillos para poder mantener un coste computacional factible.

El aprendizaje del modelo probabilístico gráfico se realiza comenzando con una estructura sin arcos y añadiendo o eliminando en cada paso la arista con la que mejor valor BIC se obtiene. Este proceso se repetirá hasta que se cumpla una condición de parada. Por lo tanto, EBNA_{BIC} se basa en una aproximación de *score+search*.

Una propiedad importante del criterio *BIC* es la posibilidad de descomponerlo: podemos calcular de manera separada el valor BIC para cada variable y posteriormente sumarlos todos. Por consiguiente, cada variable X_i tendrá asociado su *BIC(i,S,D)*.

$$BIC\ (S,D) = \sum_{i=1}^{n} BIC\ (i,S,D)$$

$$BIC\ (i,S,D) = \sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk}\ \log\ \frac{N_{ijk}}{N_{ij}} - \frac{1}{2}(r_i-1)q_i$$

El algoritmo de búsqueda de estructura usado en EBNA_{BIC} es generalmente del tipo *hill-climbing*. En cada paso, se realiza una búsqueda exhaustiva a través del conjunto de modificaciones posibles de arcos. La modificación que maximiza la ganancia del criterio *BIC* será utilizada para actualizar *S*, siempre y cuando la estructura se mantenga como un grafo dirigido acíclico (es conveniente recordar que la estructura de una Red Bayesiana debe ser un grafo dirigido acíclico). Este bucle continuará hasta que ninguna modificación de arcos supere el valor máximo alcanzado hasta el momento. Es importante resaltar que si se actualiza la estructura *S* con la modificación del arco (j,i) sólo será necesario recalcular *BIC(i,S,D)*.

El algoritmo de aprendizaje implica una secuencia de acciones que difiere entre el primer paso y los siguientes. En dicho primer paso, tomando una estructura *S* y una base de datos *D*, se calcula el cambio de *BIC* para cada una de las posibles modificaciones de arcos. Por lo tanto, tendremos que calcular tantos términos como modificaciones sean posibles, esto es, $n(n\text{-}1)$. La modificación que maximiza el criterio *BIC* manteniendo la estructura de grafo dirigido acíclico es añadida a *S*. En los pasos sucesivos, sólo se tendrán en cuenta los cambios de *BIC* debidos a modificaciones de arcos relacionadas con la variable X_i (se entiende que en el paso previo, *S* se actualizó con el arco (j,i)). Otras modificaciones no alteran el valor gracias a la posibilidad de descomponer el criterio *BIC*. En este caso, el número de términos a calcular será n-2.

III. El programa secuencial

Como se ha comentado, el valor BIC global se obtendrá a partir de la suma de los BIC locales. Aunque, recordemos, lo interesante no es dicho valor, sino la red Bayesiana que lo hace óptimo. Utilizamos cuatro estructuras de datos para el presente algoritmo:

- Un vector *BIC* de tamaño n, donde almacenaremos en $BIC[i]$ el valor local de la estructura actual asociada a la variable X_i.
- Una estructura $S[i]$ con i desde 1 hasta n donde los grafos dirigidos acíclicos serán representados como listas de adyacencia, esto es, cada S[i] define la lista de los inmediatos sucesores del vértice X_i.
- Una matriz $G[n\times n]$, donde cada entrada (j,i) representa la ganancia o perdida asociada con la modificación del arco (j,i)
- Finalmente, una matriz $paths[n\times n]$ que representa el número de caminos entre cada par de vértices (variables). Esta última estructura se utiliza para comprobar si la modificación de un arco va a producir un gráfico dirigido acíclico.

El algoritmo secuencial queda así:

Paso 1. Desde i=1 hasta n calcular *BIC*[i]
Paso 2. Desde i=1 hasta n, j=1 hasta n *G*[j,i]=0 /* Inicializar *G* */
Paso 3. Desde i=1 hasta n, j=1 hasta n
Si i≠j calcular *G*[j,i] /* El cambio en *BIC* producido por la modificación del arco (j,i) */
Paso 4. Buscar (j,i) tal que *paths*[i,j]=0 y *G*[j,i]≥*G*[r,s] para todos los r,s=1 hasta n tales que *paths*[r,s]=0
Paso 5. Si *G*[j,i]>0
Actualizar S con la modificación del arco (j,i)
Actualizar *paths*
Si no, parar
Paso 6. Desde k=1 hasta n
Si (k≠i o k≠j) calcular *G*[k,i]
Paso 7. Volver al Paso 4

IV. Estrategia de paralelización

A la hora de desarrollar el programa paralelo no se ha realizado ningún cambio en el algoritmo aparte del necesario para aprovechar la ventaja que supone descomponer el cálculo de $BIC(S,D)$ en subcálculos de $BIC(i,S,D)$. Por lo tanto, los resultados serán los mismos que en la versión secuencial pero con una notable ventaja respecto al tiempo de ejecución.

Se ha utilizado una estrategia maestro / trabajadores para descomponer el trabajo. El maestro realiza toda la parte secuencial, y reparte trabajo entre una colección de trabajadores. Como se ha dicho, la parte que se ha paralelizado es el cálculo de los $BIC(i,S,D)$ que serán ejecutados de forma independientemente en los trabajadores para finalmente devolver los resultados al maestro y continuar con la ejecución secuencial. La comunicación y sincronización entre estos elementos se realiza mediante funciones de MPI

Los experimentos se han llevado a cabo en un cluster de 5 máquinas biprocesador Pentium II a 350 Mhz, con 512KB de memoria caché por procesador y 128MB de memoria RAM. El sistema operativo utilizado ha sido GNU-Linux. Como implementación de MPI se ha elegido MPICH. Las máquinas se han interconectado mediante dos tipos de red distintos: Fast Ethernet y Myrinet. Los procesos (entre 2 y 10) se distribuyen en los recursos disponibles según el orden (M1P1, M2P1, ... , M5P1, M1P2, M2P2, ... , M5P2), donde MiPj significa “máquina i, procesador j”.

V. Resultados

El problema real utilizado para realizar los experimentos ha sido el *inexact graph matching* (concordancia inexacta de grafos) aplicado al reconocimiento de imágenes. Es habitual en problemas reales que los grafos entre los que se busca concordancia sean bastante grandes, debido a imprecisiones en las técnicas de adquisición de imágenes. En consecuencia, el mencionado EDA puede tardar días en devolver un resultado [2,3,4]. El tiempo de ejecución es directamente proporcional al número de dependencias, es decir, número de padres que el algoritmo de aprendizaje ha de tener en cuenta.

Antes de aplicar técnicas de programación paralela para reducir los tiempos de ejecución, se ha procedido a localizar las partes susceptibles de ser paralelizadas así como las que han de ejecutarse de forma secuencial. En [1] se explica el proceso seguido para obtener el porcentaje que supone en tiempo de ejecución cada parte del algoritmo $EBNA_{BIC.}$ A modo de resumen podríamos mencionar que, en función de la complejidad de los grafos que se vayan a utilizar, el porcentaje de tiempo necesario para realizar el aprendizaje del modelo probabilístico estará entre un 45% y un 85% del total, por lo que centraremos nuestro esfuerzo en paralelizar dicho aprendizaje. En la literatura ya se han realizado propuestas para la paralelización de distintos aprendizajes (redes posibilistas, Markov, etc.) [8,14,16], y más concretamente para el caso que nos ocupa, en [11] se presentan varias propuestas para la paralelización del criterio $EBNA_{BIC}$ utilizando una arquitectura MIMD con memoria compartida.

Para la evaluación, las pruebas se han realizado tomando como ejemplo dos problemas de concordancia inexacta de grafos con grafos generados aleatoriamente. Los tamaños de los problemas podrían clasificarse como mediano (grafo G_M de 30 nodos y 39 aristas, grafo G_D de 100 vértices y 297 aristas) y grande (grafo G_M de 50 nodos y 88 aristas, grafo G_D de 250 vértices y 1681 aristas). Para cada uno de ellos se ha ejecutado el programa cambiando el número de trabajadores (procesos) obteniendo los resultados resumidos en las Tablas 1 y 2.

En estas tablas, el tiempo de referencia (1 proceso) corresponde a la ejecución de la versión secuencial del programa, no a la versión paralela con un solo trabajador.

TABLA 1

PROGRAMA PARELO EJECUTADO SOBRE FAST ETHERNET.

NP = NÚM. PROCESOS, TE = TIEMPO DE EJECUCIÓN, SU = SPEEDUP.

	Problema mediano		Problema grande	
NP	TE	SU	TE	SU
1	1h 28’ 7”	1	37h 0’ 9”	1
2	1h 53’ 3”	0,78	34h 24’ 55”	1,08
4	1h 26’ 4”	1,02	21h 16’ 22”	1,74
5	1h 28’ 55”	0,99	19h 50’ 30”	1,86
6	1h 23’ 47”	1,05	18h 15’ 10”	2,03
8	1h 17’ 57”	1,13	15h 52’ 58”	2,33
10	1h 34’ 20”	0,93	17h 22’ 47”	2,13

TABLA 2

PROGRAMA PARELO EJECUTADO SOBRE MYRINET

	Problema mediano		Problema grande	
NP	TE	SU	TE	SU
1	1h 28’ 7”	1	37h 0’ 9”	1
2	1h 4’ 25”	1,37	20h 21’ 1”	1,82
4	47’ 19”	1,86	11h 50’ 2”	3,13
5	44’ 47”	1,97	11h 6’ 42”	3,33
6	45’ 13”	1,95	10h 20’ 6”	3,58
8	41’ 2”	2,15	8h 42’ 15”	4,25
10	42’ 2”	2,10	**8h 10’ 7”**	**4,53**

Utilizando las herramientas suministradas con MPICH, generamos y ejecutamos una versión de nuestro programa capaz de generar un fichero con las trazas de las operaciones realizadas. Estas trazas se pueden visualizar con Upshot para analizar el comportamiento en tiempo de ejecución de nuestro programa.

Los gráficos generados nos permitieron observar cómo hay una etapa inicial y una final (que suponen aprox. el 15% del tiempo en el problema grande) en la que no se explota nada de paralelismo. La etapa central consta de una sucesión de fases <comunicación, cómputo, comunicación>, donde la proporción cómputo / comunicación depende de la red de interconexión empleada. En Fast Ethernet cada procesador pasa más tiempo intercambiando datos que realizando trabajo efectivo (Figura 1). En Myrinet, dadas las mejores características de esta red, el tiempo efectivo de cómputo es aproximadamente 2/3 del tiempo total de esta etapa central (Figura 2).

VI. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

Tal y como se ha comentado a lo largo del artículo, el porcentaje que supone el cálculo de $BIC(S,D)$ varía en función del tamaño del problema y es por ello que tendremos que tener en cuenta este aspecto junto con el número de procesos y el tipo de red utilizado.

Ejecutando el problema mediano en Fast Ethernet, el coste que supone la comunicación entre procesos anula la mejora obtenida por la paralelización de la función BIC. En cambio, en el problema grande, se puede observar un *speedup* de 1,74 cuando ejecutamos el programa utilizando cuatro procesos mientras que a partir de este número el *speedup* se va estancando e incluso empeora (para el caso de 10 procesos).

En cambio, utilizando la red Myrinet, al proporcionar ésta mayor velocidad (1,2 Gbps) y tener un *overhead* menor, se observa mejoría en ambos problemas. En el mediano, en la ejecución realizada con cinco procesos, el tiempo de ejecución se ha reducido prácticamente a la mitad (*speedup* de 1,97) y a partir de ahí (6, 8 ó 10 procesos) la ganancia es mucho menos significativa. Con respecto al problema grande, el estancamiento se va produciendo cuando el número de procesos supera 8 (*speedup* de 4,25) aunque con un número de procesos menor, por ejemplo la mitad (4), el *speedup* ya es bastante razonable (3,13).

Aún cuando nuestro trabajo se ha centrado en el problema de la concordancia inexacta de grafos y la aproximación $EBNA_{BIC}$, los EDA han sido aplicados con éxito en multitud de problemas diferentes utilizando las distintas aproximaciones existentes (UMDA, MIMIC, $EBNA_{K2+pen}$, $EGNA_{BIC}$, EMNA, etc.). Por lo tanto, en nuestros planes está estudiar y paralelizar (si procede) las distintas aproximaciones de forma paulatina, para lograr, al igual que en el caso que nos ocupa, reducir los tiempos de ejecución necesarios para la resolución de problemas. De esta forma, podremos sustituir los EDA por una versión paralelizada que permita a todos los investigadores que trabajan con éstos algoritmos reducir los tiempos de ejecución de sus respectivos experimentos.

VII. AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por el proyecto MCYT TIC2001-0591-C02-02.

VIII. REFERENCIAS

[1] Bengoetxea, E., Miguel, J., Larrañaga, P., Bloch, I. (2002). Model-based recognition of brain structures in 3D magnetic resonance images using graph matching and parallel estimation of distribution algorithms. Submitted for Publication in Cybernetics & Systems; Special Issue on Pattern Recognition and Image Analysis in Cybernetic Applications.

[2] Bengoetxea, E., Larrañaga, P., Bloch, I., and Perchant, A. (2001a). Solving graph matching with EDAs using a permutation-based representation. In Larrañaga, P. y Lozano, J. A., editors, Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, pages 243-265. Kluwer Academic Publishers.

[3] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2000). Inexact graph matching using learning and simulation of Bayesian networks. An empirical comparison between different approaches with synthetic data. In Proceedings of CaNew workshop, ECAI 2000 Conference, ECCAI, Berlin.

[4] Bengoetxea, E., Larrañaga, P., Bloch, I., Perchant, A., and Boeres, C. (2001b). Learning and simulation of Bayesian networks applied to inexact graph matching. Pattern Recognition. (submitted).

[5] Chickering, D. M., Geiger, D., and Heckerman, D. (1994). Learning Bayesian networks is NP-hard. Technical report, Technical Report MSR-TR-94-17, Microsoft Research, Redmond, WA.

[6] De Bonet, J. S., Isbell, C. L., and Viola, P. (1997). MIMIC: Finding optima by estimating probability densities. In Advances in Neural Information Processing Systems, volume 9. M. Mozer, M. Jordan and Th. Petsche eds.

[7] Etxeberria, R. and Larrañaga, P. (1999). Global optimization with Bayesian networks. In Special Session on Distributions and Evolutionary Optimization, pages 332-339. II Symposium on Articial Intelligence, CIMAF99.

[8] Freitas, A. A. and Lavington, S. H. (1999). Mining very large databases with parallel processing. Kluwer Academic Publishers, London.

[9] Larrañaga, P. and Lozano, J. A., editors, Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers 2001.

[10] Larrañaga, P., Etxeberria, R., Lozano, J., and Pe~na, J. (2000). Optimization in continuous domains by learning and simulation of Gaussian networks. In Proceedings of the Workshop in Optimization by Building and using Probabilistic Models. A Workshop within the 2000 Genetic and Evolutionary Computation Conference, GECCO 2000, pages 201-204, Las Vegas,Nevada, USA.

[11] Lozano, J. A., Sagarna, R., and Larrañaga, P. (2001). Parallel estimation of distribution algorithms. In Larrañaga, P. and Lozano, J. A., editors, Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers.

[12] Mühlenbein, H. (1998). The equation for response to selection and its use for prediction. Evolutionary Computation, 5:303-346.

[13] Mühlenbein, H. and Paaβ, G. (1996). From recombination of genes to the estimation of distributions i. Binary parameters. In Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature - PPSN IV, pages 178-187.

[14] Sangüesa, R., Cortés, U., and Gisol, A. (1998). A parallel algorithm for building possibilistic causal networks. International Journal of Approximate Reasoning, 18:251-270.

[15] Schwarz, G. (1978). Estimating the dimension of a model. Annals of Statistics, 7(2):461-464.

[16] Xiang, Y. and Chu, T. (1999). Parallel learning of belief networks in large and difficult domains. Data Mining and Knowledge Discovery, 3:315-339.

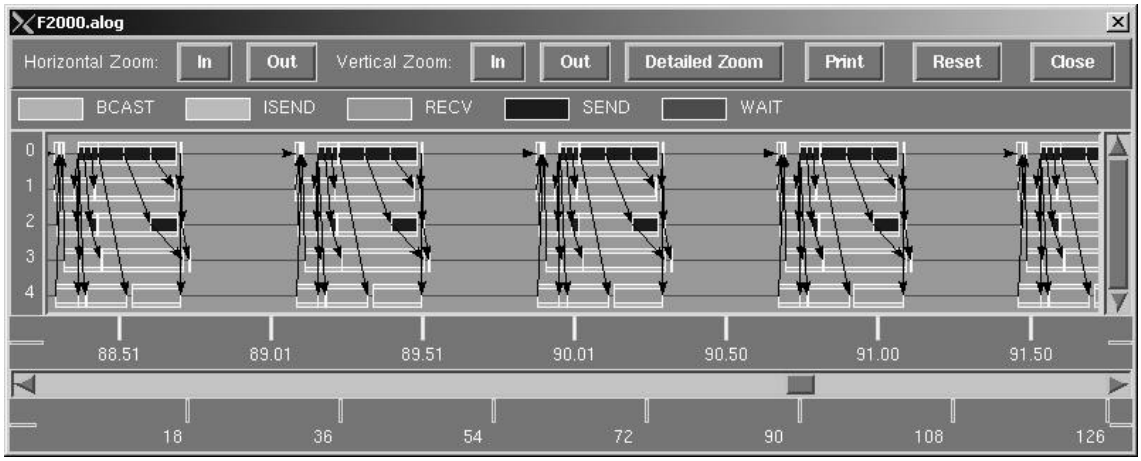

Fig. 1. Instantánea de la ejecución del programa sobre Fast Ethernet.

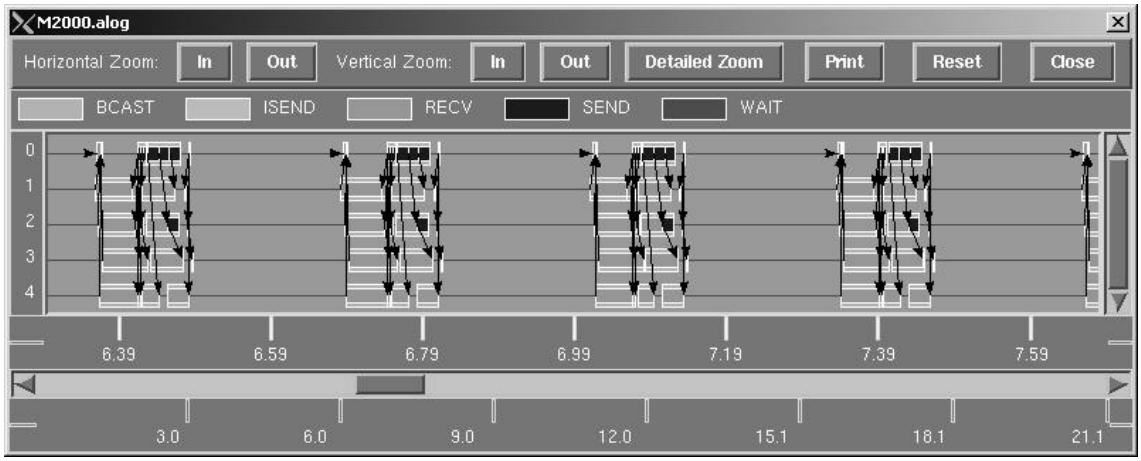

Fig. 2. Instantánea de la ejecución del programa sobre Myrinet.