Mapping en Arquitecturas Heterogéneas

1. Datos de la Tesis Doctoral

La autora de la Tesis Doctoral es De Giusti Laura Cristina, bajo la dirección del Dr. Luque Emilio quien trabaja en la Universidad Autónoma de Barcelona (España) y la co dirección de los Dres. Simari Guillermo quien se desempeña en la Universidad Nacional del Sur (Argentina) y Marcelo Naiouf quien trabaja en la Universidad Nacional de La Plata.

Esta Tesis se desarrolló en el Instituto de Investigación en Informática LIDI, perteneciente a la Facultad de Informática, la cual pertenece a la Universidad Nacional de La Plata.

La exposición de la Tesis se realizó en la Facultad de Informática el día 18 de Noviembre de 2008. Los miembros del jurado fueron el Ms. Ardenghi Jorge (UNS), la Dra. Printista Marcela (UNSL) y el Dr. Margalef Tomás (UAB). La nota de la exposición fue 10 (diez).

2. Objetivo y contenido de la Tesis

En esta sección se desarrolla el objetivo principal de la Tesis y un breve resumen del contenido de la misma.

2.1 Objetivo

El objetivo principal de la Tesis es: Estudio de modelado de aplicaciones paralelas sobre arquitecturas distribuidas heterogéneas (clusters), y su utilización para el mapping estático de tareas a procesadores buscando optimizar el tiempo de ejecución de las aplicaciones.

2.2 Contenido

En cuanto al contenido primero se planteará el contexto de la Tesis, luego los modelos y algoritmos desarrollados y por último los resultados obtenidos de las pruebas realizadas.

2.2.1 Contexto

En general, toda solución a un problema paralelo requiere del desarrollo de las siguientes fases: detección de paralelismo, definición del grafo de tareas y asignación de tareas a procesadores. En la figura 1 se muestran dichas fases.

La fase de detección del paralelismo consiste en poder determinar y descomponer las actividades secuenciales que pueden ser realizadas en paralelo, con sus interdependencias. La segunda fase es la definición del grafo de tareas de la aplicación, el cual utiliza las tareas identificadas en la fase anterior. En esta etapa se deben identificar las interacciones entre las tareas, los cuales se llevan a cabo mediante un mecanismo de comunicación. Esto se hace por medio de la definición del grafo de la aplicación y en el mismo deben considerarse problemas importantes como: la granularidad de las tareas, descomposición de los datos a tratar por cada una y organización de las comunicaciones

entre tareas. El grafo resultante de la aplicación estará formado por un conjunto de tareas que se comunican y que realizan iguales o distintas funciones según el modelo de programación al que pertenecen (aplicaciones con paralelismo de datos, aplicaciones con paralelismo funcional y aplicaciones mixtas). La tercera fase es la asignación de tareas a procesadores. En esta fase se busca realizar la asignación de manera de minimizar el tiempo de ejecución final. Dicha asignación se lleva a cabo mediante un proceso de scheduling, el cual determina dónde y cuándo debe ejecutarse cada tarea, es decir en qué procesador y en qué orden se ejecutarán las tareas de la aplicación. En este trabajo se utilizó la asignación estática.

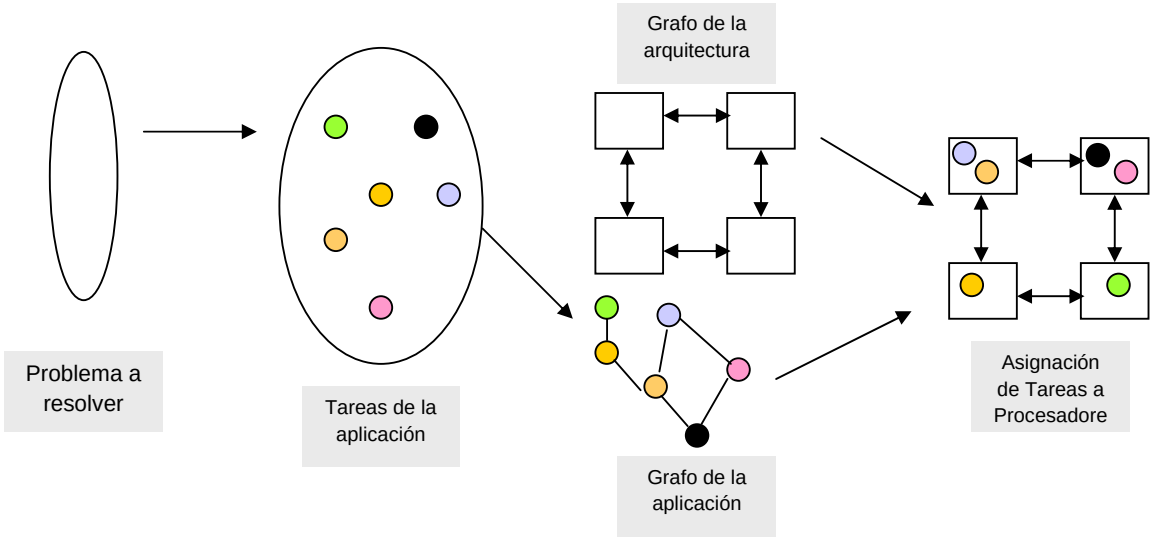

Figura 1 – Fases del desarrollo de la aplicación

Existen numerosos modelos para representar una aplicación paralela, cada uno tiene sus ventajas y desventajas. Entre los modelos basados en grafos más conocidos que se utilizan para representar aplicaciones paralelas se puede mencionar: Grafo de Precedencia de Tareas (Task Precedence Graph TPG), Grafo de Interacción de Tareas (Task Interaction Graph TIG) y Temporal Task Interaction Graph (TTIG).

El problema que presentan los tres modelos mencionados anteriormente es que asumen que los nodos de procesamiento son idénticos y en la actualidad, la mayoría de las aplicaciones paralelas son ejecutadas sobre clusters / multiclusters de procesadores heterogéneos, ya que es difícil formar una arquitectura cuyos procesadores tengan exactamente las mismas características. Dado que dichos modelos no consideran la posible heterogeneidad de la arquitectura (en cuanto a los procesadores y a la red de interconexión) nace la necesidad de definir un nuevo modelo que permita tener en cuenta esta situación.

2.2.2 Desarrollo de la Tesis

En esta Tesis Doctoral se definieron dos modelos basados en grafos para la representación de aplicaciones paralelas (TTIGHA, MPAHA) las cuales pueden ser ejecutadas en arquitecturas homogéneas o heterogéneas. A su vez, en esta Tesis también se implementaro dos algoritmos de mapping (MATEHA, MATEHAIB) basados en el primer modelo y otro algoritmo de mapping (AMTHA) basado en el segundo modelo.

2.2.2.1 Modelo TTGIHA y Algoritmos MATEHA y MATEHAIB

El modelo TTGIHa (Temporal Task Interaction Graph in Heterogeneous Architecture), permite representar aplicaciones paralelas considerando la heterogeneidad de la arquitectura, tanto en lo que respecta a los procesadores como a las comunicaciones. Se basa en la construcción de un grafo *G(V,E,B)* donde:

- *V*, es el conjunto de nodos donde cada uno representa una tarea T_i del programa paralelo.
- *E*, es el conjunto de aristas que representan la comunicación entre los nodos del grafo.
- *B* es un conjunto de identificadores que indica cuales son los nodos (o tareas) básicos (o iniciales) en la aplicación, es decir aquellos que pueden comenzar su ejecución sin necesidad de interacción con otro nodo.

En el primer parámetro del grafo (V) cada nodo representa una tarea T_i del programa paralelo. Dada la posibilidad de contar con una arquitectura heterogénea se deben tener en cuenta los tiempos de cómputo en cada uno de los tipos de procesadores que la componen. Es decir, el nodo *i* ($V_i \in V$) almacena el tiempo de cómputo de la tarea T_i en cada uno de los tipos diferentes de procesador. Por lo tanto, $V_i\ (p)$ = tiempo de ejecución de la tarea T_i en el tipo de procesador *p*.

En el segundo parámetro del grafo (*E*), las aristas representan las comunicaciones que existen entre cada par de tareas. En este conjunto una arista *A* entre dos tareas T_i y T_j contiene el volumen de comunicación en bytes y el "grado de concurrencia" entre ambas.

El "grado de concurrencia" es una matriz que representa el máximo porcentaje de cómputo entre ambas tareas en los distintos tipos de máquinas h_{ij}, donde $h_{ij}(s,d)$ representa el grado de concurrencia entre las tarea T_i en un procesador de tipo *s* y la tarea T_j en un procesador de tipo *d*.

La estrategia del algoritmo de schedulling MATEHA (Mapping Algorithm based on Task dEpendencies in Heterogeneous Architectures) consiste en determinar para cada una de las tareas del grafo *G* formado por el modelo TTIGHA, a qué procesador debe ser asignada para lograr el mayor rendimiento de la aplicación en la arquitectura utilizada. En primer lugar, para cada nodo del grafo del modelo TTIGHA se define el nivel; este valor será utilizado para realizar la asignación de las tareas del grafo con cierta prioridad. Luego, para cada tarea T_i (aun no asignada) se calcula el procesador que genera la máxima ganancia, teniendo en cuenta las tareas adyacentes ya asignadas; es decir, para cada adyacente ya asignada T_j se calcula la diferencia entre ejecutar la tarea T_i y T_j en el mismo o en diferentes procesadores. Para esto se consideran los tiempos de ejecución de las tareas T_i y T_j, los tiempos de comunicación y el grado de concurrencia entre ambas.

Si bien los resultados de la asignación generada por MATEHA eran razonables, este algoritmo requería un paso extra para determinar el orden en que debían ser ejecutadas las tareas dentro de un procesador, y además la asignación generaba "*huecos*" entre la ejecución de las tareas de un procesador que podrían ser aprovechados para ejecutar otras tareas.

Por lo tanto se implementó el algoritmo MATEHAIB, el cual también utilizaba el modelo TTIGHA pero se diferencia del algoritmo anterior en la manera de calcular la ganancia de una tarea. Esta diferencia radica en que ahora cada procesador maneja dos listas: *LU* (lista de subtareas que componen las tareas asignadas, que ya fueron ubicadas en el procesador y en el instante en que cada una debe ejecutarse) y *LNU* (lista de subtareas que componen las tareas asignadas, que pertenecen al procesador pero aun no pueden ubicarse en un intervalo de tiempo dado que alguna de sus adyacentes no ha sido ubicada). Luego, al momento de elegir el procesador, para cada procesador

disponible en la arquitectura se calcula el instante de tiempo que la tarea que se quiere asignar va a terminar su ejecución. Este tiempo se obtiene en base a los huecos que dejan las subtareas ya ubicadas en el procesador y las restricciones de ejecución que cada una de las subtareas tienen. Al final se elige el procesador que minimice el instante en el cual la tarea terminará su ejecución.

Una vez que el algoritmo MATEHAIB terminó su ejecución, todas las tareas han sido asignadas a algún procesador de los que componen la arquitectura. Esta asignación además determina en que instante debe ejecutarse cada subtarea dentro de un procesador.

Si bien MATEHAIB aprovecha en gran medida los huecos libres de cada procesador, el orden de ejecución de este algoritmo es bastante elevado.

2.2.2.2 Modelo MPAHA y Algoritmo AMTHA

Teniendo en cuenta las características mencionadas para los algoritmos anteriores se implementó un nuevo algoritmo de mapping llamado AMTHA (Automatic Mapping Task on Heterogeneous Architectures). Este algoritmo no utiliza toda la información incluida en el modelo TTIGHA y por este motivo también se implementó un nuevo modelo llamado MPAHA (Model for Parallel Algorithms on Heterogeneous Architectures) en el cual se basa dicho algoritmo. Este modelo es una variante del modelo TTIGHA, en el cual no se representa el grado de concurrencia entre pares de tareas.

El algoritmo AMTHA se aplica al modelo MPAHA. El mismo permite determinar la asignación de tareas a los procesadores de la arquitectura a utilizar, buscando minimizar los tiempos de ejecución de la aplicación en dicha arquitectura. Además este algoritmo debe indicar el orden en el cual deben ejecutarse las subtareas (que componen las tareas) asignadas en cada procesador. Al igual que los algortimos anteriores, AMTHA considera la heterogeneidad de la arquitectura con respecto a la red de interconexión y al conjunto de procesadores.

AMTHA utiliza los valores del grafo G generado por MPAHA; estos valores son el tiempo de cómputo de una subtarea en cada tipo de procesador, el volumen de comunicación con sus adyacentes y la relación de pertenencia de subtareas a tareas. AMTHA asigna una tarea por vez, hasta que todas han sido asignadas. El siguiente pseudocódigo describe los pasos fundamentales de este algoritmo:

```
Calcular el rank de cada tarea.
Mientras (no se asignaron todas las tareas)
    Seleccionar la próxima tarea t a asignar.
    Elegir el procesador p al cual debe asignarse la tarea t.
    Asignar la tarea t (elegida en el paso 2) al procesador p (elegido en el paso 3).
    Actualizar el rank de las tareas involucradas.
```

Al finalizar el algoritmo todas las tareas han sido asignadas a algún procesador y además se ha determinado el orden en que las subtareas que componen las tareas asignadas al procesador deben ser ejecutadas dentro del mismo.

Dado un grafo *G*, el rank de una tarea *Rk(T)* se define como la suma de los tiempos promedios de las subtareas que la componen y que se encuentran listas para su ejecución (todas sus predecesoras ya han sido asignadas y ubicadas en un procesador). La Fórmula 1 expresa la definición antes mencionada:

$$Rk\ (T) = \sum_{i \in L(T)} W_{prom}(St_i) \qquad \text{Fórmula 1}$$

donde:

L(T) es el conjunto de subtareas listas para la tarea *T*.

$W_{prom}(St)$ es el tiempo promedio de la subtarea *St*. El tiempo promedio se calcula como se muestra en la Fórmula 2

$$W_{prom}(ST_i) = \frac{\sum_{p \in P} V_{st_i}(tipo\ de\ procesador\ de\ p)}{\#\ p} \qquad \text{Fórmula 2}$$

donde:

P es el conjunto de procesadores que componen la arquitectura.

#p es la cardinalidad de *p*.

Una vez obtenido el rank de cada tarea que compone la aplicación **se elige la tarea** que lo máximo rank. En caso que ocurriese que dos o más tareas poseen el mismo valor máximo el algoritmo rompe este empate eligiendo aquella tarea que minimice el promedio del tiempo total de ejecución de la tarea. La Fórmula 3 muestra este cálculo:

$$Tprom\ (T) = \sum_{i \in T} W_{prom}(St_i) \qquad \text{Fórmula 3}$$

La **selección del procesador** consiste en elegir aquel procesador, perteneciente a la arquitectura, que minimice el tiempo de ejecución al asignar la tarea elegida al procesador.

Para poder comprender cómo se calcula el tiempo de un procesador p, debe tenerse en cuenta que cada procesador mantiene una lista de subtareas que ya le fueron asignadas LU_p y que pueden ejecutarse (todas sus predecesoras ya fueron ubicadas) y otra lista que contiene aquellas subtareas que fueron asignadas a *p* pero aún no pueden ejecutarse LNU_p (alguna de sus predecesoras aun no ha sido ubicada).

Por lo tanto para calcular el procesador *p* que será elegido se tiene en cuenta dos casos:

1. Todas las subtareas de la tarea *t* pueden ser ubicadas en *p* (es decir todas sus predecesoras han sido ubicadas).
2. Alguna de las subtareas de *t* no puede ser ubicada en *p* (esto ocurre cuando alguna predecesora de alguna subtarea de *t* no ha sido ubicada).

Para el primer caso el tiempo *Tp* del procesador *p*, está dado por el instante en *p* en el cual termina la ejecución de la última subtarea de *t*. En cambio, para el segundo caso, el tiempo *Tp* del procesador *p*, está dado por el tiempo en que la última subtarea de LU_p finalizará, más la suma de los tiempos de ejecución en *p* para cada una de las subtareas de LNU_p.

Al asignar una tarea *t* a un procesador *p*, cada subtarea St_k perteneciente a *t* se intenta ubicar en el procesador en un instante de tiempo en el cual todas las subtareas adyacentes a ella han finalizado (inclusive su predecesora dentro de *t* en caso de que posea). Esto puede ser en un intervalo libre entre dos subtareas ya ubicadas en *p*, o bien en un intervalo al final de estas. Si la subtarea St_k no puede ser ubicada se agrega a la lista *LNU* de *p*. Cada vez que una subtarea St_k es agregada a la lista *LU* de algún procesador, se intenta ubicar todas sus subtareas predecesoras que pertenezcan a tareas ya asignadas.

Como primera acción en este paso se le asigna -1 como valor de rank a la tarea *t* que fue asignada al procesador *p*. Esto se realiza para que la tarea *t* no vuelva a ser elegida para asignarse.

Además en este paso se considera la siguiente situación: para cada subtarea St_k, ubicada en el paso anterior se analiza la necesidad de **actualizar el rank** de las tareas a las cuales pertenecen las sucesoras St_{succ} de St_k, es decir, si St_{succ} tiene todas sus predecesoras ubicadas entonces el rank de la tarea a la cual pertenece St_{succ} se actualiza incrementándolo con $W_{prom}(St_{succ})$.

2.2.3 Resultados

Para analizar el comportamiento de los algoritmos propuestos se han generado un conjunto de aplicaciones. Las mismas varían en términos de cantidad de tareas que las componen; cantidad de subtareas que forman las tareas; tamaño de las subtareas y el volumen de comunicación entre las subtareas. En todas las aplicaciones, el tiempo de cómputo supera al de comunicaciones (aplicaciones de grano grueso).

Además en cada una de las pruebas realizadas debe ser indicada previamente la configuración de la arquitectura a utilizar. Por lo tanto se debe especificar la cantidad de tipos diferentes de procesadores, la cantidad de máquinas de cada tipo, y las características de comunicación entre los procesadores (tiempo de startup de cada uno, y el tiempo de transferencia entre cada par de ellos).

Con cada una de estas pruebas creadas (aplicación y configuración de la arquitectura), se generaron los grafos correspondientes a los modelos TTIGHA y MPAHA respectivamente, y a partir de estos grafos se realizó el mapping por medio de los algoritmos MATEHA, MATEHAIB y AMTHA. Al mismo tiempo, se realizó el mapping con dos algoritmos conocidos: MATE (para arquitecturas homogéneas) y HEFT (para arquitecturas heterogéneas).

Se analizó el comportamiento de los tres algoritmos desarrollados en esta Tesis, comparando el tiempo final de cada una de las aplicaciones, involucradas en cada prueba, de acuerdo al scheduling generado por los algoritmos y los generados por MATE y HEFT. En cuanto a los resultados se puede mencionar que:

- AMTHA obtiene mejores resultados en el 100% de las pruebas que los algoritmos MATEHA y MATEHAIB.
- Los algoritmos MATEHA y MATEHAIB obtienen en el 98% de las pruebas mejores resultados que el algoritmo MATE, mientras que AMTHA lo hace en el 100% de las pruebas.
- Los algoritmos MATEHA y MATEHAIB obtienen en el 28% de las pruebas mejores resultados que el algoritmo HEFT, mientras que AMTHA lo hace en el 89% de las pruebas.

3. Aporte

A partir del desarrollo de la Tesis se pueden mencionar entre los aportes más relevantes a:

- El diseño de dos modelos para la representación de aplicaciones paralelas sobre arquitecturas heterogéneas: Temporal Task Interaction Graph in Heterogeneous Architecture (TTIGHA) y Model for Parallel Algorithms on Heterogeneous Architecturas (MPAHA).
- La implementación de tres algoritmos de mapping, dos de ellos Mapping Algorithm on Task dEpendecies in Heterogeneous Architectures (MATEHA) y Mapping Algorithm on Task dEpendecies in Heterogeneous Architectures using the policity Inserted Based (MATEHAIB) basados en el primer modelo y un tercero Automatic Mapping Task on Heterogeneous Architecture (AMTHA) basado en el segundo modelo.
- El análisis comparativo de los tres algoritmos mencionados anteriormente con algoritmos preexistentes.

4. Conclusiones

Entre las conclusiones más significativas de la Tesis se pueden mencionar:

- El desarrollo de un modelo, TTIGHA, para representar aplicaciones paralelas en arquitecturas heterogéneas y luego la implementación de dos algoritmos de mapping, MATEHA y MATEHAIB, basados en dicho modelo.
- El desarrollo de un nuevo modelo, MPAHA, y la implementación de un algoritmo de mapping AMTHA basado en el mismo modelo, que mejora los resultados de los algoritmos anteriores y reduce el overhead generado por el mapping.
- La comparación de los resultados obtenidos por los algoritmos desarrollados (MATEHA, MATEHAIB y AMTHA) frente a los algoritmos MATE (usado para arquitecturas homogéneas) y HEFT (uno de los más usados para arquitecturas heterogéneas).
- De las pruebas aplicadas a los algoritmos se deduce que:
  - ✓ Todos los algoritmos propuestos mejoran en general los resultados obtenidos por MATE.
  - ✓ Respecto a los resultados del algoritmo HEFT se fueron mejorando con los distintos algoritmos hasta obtener que el algoritmo AMTHA, el cual sobre 32 grupos de pruebas realizados obtiene mejores resultados que HEFT en 28 grupos e iguala los resultados obtenidos por HEFT en 2 grupos.

5. Líneas Futuras

A partir de la realización de la Tesis se pueden desprender las siguientes líneas futuras:

- Analizar la posibilidad de utilizar los modelos propuestos en arquitecturas tipo GRID y Cluster de Multicore, y en caso de ser necesario, realizar las modificaciones.

- Analizar el comportamiento de los algoritmos de mapping propuestos en los tipos de arquitecturas mencionados anteriormente, y; en caso de ser necesario, adaptarlos.

6. Bibliografía Básica de la Tesis

1. Akl S., "Parallel Computation. Models and Methods", Prentice-Hall, Inc., 1997.

2. Andrews G. "Foundations of Multithreaded, Parallel and Distributed Programing" Addison Wesley Higher Education 2000. ISBN-13: 9780201357523

3. Chaudhary V. Aggarwal J. "A generalized scheme for Mapping Parallel Algorithms". IEEE Transactions on Parallel and Distributed Systems. Vol 4, nro 3. Pags 328-346. 1993.

4. De Giusti L., Chichizola F, Naiouf M., De Giusti A. Análisis de la Robustez del Método de Asignación MATEHa". XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007), Corrientes – Resistencia, Argentina.

5. England D, Weissman J, and Sadagopan J, "A New Metric for Robustness with Application to Job Scheduling", IEEE International Symposium on High Performance Distributed Computing 2005 (HPDC-14), Research Triangle Park, NC. pags 24-27. 2005.

6. Fernandez E. Busseli B. "Bounds on the Number of Processors and Time for Multiprocessors Optimal Schedules". IEEE Transactions on Computers, 22(8):745-751, 1973.

7. Grama A., Gupta A., Karypis G., Kumar V., "An Introduction to Parallel Computing. Design and Analysis of Algorithms", Pearson Addison Wesley, 2nd Edition, 2003.

8. Hwang J., Chow Y., Anger F., Lee C. "Scheduling Precedence Graphs in Systems with Interprocecessor Communication Times". SIAM Journal of Computing, 18(2): 244-257, Abril 1989.

9. Leopold C., "Parallel and Distributed Computing. A Survey of Models, Paradigms and Approaches". Wiley, 2001. ISBN: 0471358312.

10. Roig C. Ripoll A. Guirado F "A New Task Graph Model for Mapping Message Passing Applications" IEEE Transactions on Parallel and Distributed Systems. Vol 18 nro 12, 2007.

11. Roig C., Ripoll A., Senar M.A., Guirado F., Luque E. "Modelling Message Passing Programs for Static Mapping". In Euromicro Worshop on Parallel and Distributed Processing (PDP'00). IEEE CS Press. USA, pag 229-236, 1999.

12. Roig Concepcio. "Algoritmos de asignación basados en un nuevo modelo de representación de programas paralelos" Tesis Doctoral – 2005 Unidad Autónoma de Barcelona -España

13. Topcuoglu H., Wu M. "Performance- Effective and Low-Compexity Task Scheduling for Heterogeneous Computing". IEEE Transactions on Parallel and Distributed Systems, Vol 13 nro 3. 2002.

14. Uppaluri S., Izadi B. and Radhakrishnan D. “Low-Power Dynamic Scheduling in Heterogeneous Systems” In Proceedings of the International Conference on Embedded Systems and Applications, (ESA ’03) ISBN: 1-932415-05-X Pags 261-267. 2003.

15. Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 7(5):506-521, May 1996.

16. Zoltan J., Kacsuk P., Kranzlmuller D., “Distributed and Parallel Systems: Cluster and Grid Computing”. (The International Series in Engineering and Computer Science). Springer; 1st edition, 2004.