

# Incorporando conceptos en la enseñanza de Concurrencia y Paralelismo utilizando el entorno CMRE

Laura De Giusti<sup>1</sup>, Fabiana Leibovich<sup>1</sup>, Franco Chichizola<sup>1</sup>, Marcelo Naiouf<sup>1</sup>,  
Armando De Giusti<sup>1,2</sup>

<sup>1</sup> Instituto de Investigación en Informática LIDI (III-LIDI) – Facultad de Informática –UNLP

<sup>2</sup> Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)

Argentina

{ldgiusti, fleibovich, francoch, mnaiouf, degiusti}@lidi.info.unlp.edu.ar

**Abstract.** Se analiza el modo de incorporar los conceptos de heterogeneidad de los procesadores y su impacto sobre el speedup y la eficiencia de un sistema paralelo, así como el estudio de la eficiencia energética de algoritmos paralelos, en función de la potencia de los procesadores.

El entorno CMRE (Concurrent Multi Robot Environment) se extiende para poder considerar relojes virtuales diferentes por robot (procesador) y también el costo en tiempo y en consumo de energía de las operaciones que realizan los robots (Move, Depositar / Recoger / Comunicarse por Mensajes / Informar).

En el trabajo se analizan algunos casos ejemplo para mostrar el modo de presentar los conceptos para el alumno.

**Keywords:** Concurrencia, Paralelismo, Procesadores heterogéneos, Algoritmos paralelos, Consumo energético.

## 1 Introducción

La Concurrencia ha sido un tema central en el desarrollo de la Informática y los mecanismos de expresión de procesos concurrentes que cooperan y compiten por recursos ha estado en el núcleo curricular de los estudios de Informática desde la década del 70, en particular a partir de los trabajos fundacionales de Hoare, Dijkstra y Hansen [HOA85][DIJ78][HAN77]. Los conceptos se enseñaron tradicionalmente asumiendo la disponibilidad de un único procesador que podía explotar parcialmente la concurrencia del algoritmo, en función de la arquitectura física disponible (incluso con hardware específico como coprocesadores, controladores de periféricos o esquemas vectoriales que replicaban unidades aritmético-lógicas).

El paralelismo, entendido como “concurrencia real” en la que múltiples procesadores pueden operar simultáneamente sobre varios threads o hilos de control en el mismo instante, resultó durante muchos años una posibilidad limitada por la tecnología de hardware disponible [DAS89]. Las currículas informáticas clásicas [ACM68][ACM78][ACM99] contenían conceptos de concurrencia en diferentes áreas (Lenguajes, Paradigmas, Sistemas Operativos) pero omitían casi totalmente el tratamiento del paralelismo, salvo al plantear los conceptos de sistemas distribuidos.

Las actuales arquitecturas de procesadores, que integran múltiples “cores” o núcleos en un procesador físico han producido un notorio impacto, obligando a replantear el “modelo base” de un procesador. Esto ha llevado a reemplazar el formato de “máquina de Von Neuman” con un solo hilo de control, por un esquema que integra múltiples “cores” cada uno con uno o más hilos de control y varios niveles de memoria accesible en forma diferenciada [GEP06][MCC08].

Al mismo tiempo, los cambios tecnológicos han producido una evolución de los temas de mayor interés en Informática, fundamentalmente por las nuevas aplicaciones que se desarrollan a partir de disponer de arquitecturas y redes de comunicación de mayor potencia y menor costo [HOO13]. Por este motivo, las recomendaciones curriculares internacionales mencionan la necesidad de tratar los temas de concurrencia y paralelismo desde las etapas tempranas de la formación del alumno, dado que todas las arquitecturas y sistemas reales con los que trabajará son esencialmente paralelos [ACM13]. Sin embargo, la programación paralela (y los conceptos fundamentales de concurrencia) resulta más compleja para un alumno en las etapas iniciales de su formación, y es necesario contar con nuevas estrategias que permitan abordar.

Dados los estímulos que los alumnos reciben desde temprana edad, ya sea mediante juegos electrónicos, computadoras, celulares, tablets o cualquier otro dispositivo electrónico, la utilización de herramientas interactivas para la enseñanza de conceptos fundamentales a alumnos desde un curso CS1 [ACM04][ACM08][ACM13] se ha vuelto una herramienta fundamental [DEG14a]. En este sentido, la posibilidad de dar los primeros pasos en el mundo de la programación mediante un entorno gráfico e interactivo permite reducir la brecha que tradicionalmente existió entre la abstracción y la posibilidad de ver gráficamente la aplicación de los conceptos estudiados en un entorno que conceptualmente es similar a los utilizados en la vida cotidiana [AMD09][HOO13].

El entorno gráfico CMRE, donde se cuenta con un conjunto de robots que se mueven en una ciudad, ha permitido incorporar la enseñanza de los conceptos básicos de concurrencia y paralelismo en un curso inicial de Informática. Sin embargo, existen características avanzadas (tales como heterogeneidad, costo en tiempo, balance de carga, consumo de energía, etc) que resulta de interés introducir a fin de utilizar el entorno en cursos de años superiores. Este trabajo presenta la extensión de CMRE para tener en cuenta los conceptos mencionados.

El trabajo está estructurado de la siguiente manera: la Sección 2 presenta el entorno CMRE actual. En la Sección 3 se aborda el problema de la heterogeneidad, mientras la Sección 4 se centra en el consumo energético. En la Sección 5 se detallan las conclusiones y líneas de trabajo futuro.

## **2 El entorno CMRE actual**

Las características principales del entorno CMRE pueden resumirse de la siguiente manera [GEP06][DEG14b]:

- Existen múltiples procesadores (robots) que realizan tareas y que pueden cooperar y/o competir. Los mismos representan los “cores” de una arquitectura multiprocesador real. Estos robots virtuales pueden tener un reloj propio y diferentes tiempos para la ejecución de sus tareas específicas.

- El modelo de ambiente (“ciudad”) en la que desarrollan sus tareas admite áreas privadas, parcialmente compartidas y totalmente compartidas. En una área privada sólo puede moverse un único robot, en un área parcialmente compartida se especifica el conjunto de robots que pueden moverse en ella y en un área totalmente compartida todos los robots definidos en el programa pueden moverse dentro de ella.
- Si se instancia a un sólo robot en un área que abarque toda la ciudad, se repite el esquema del Visual Da Vinci.
- Cuando dos o más robots están en un área compartida (parcial o totalmente), compiten por el acceso a las esquinas del recorrido y a los recursos que allí existan. Para esto deben sincronizar.
- Cuando dos o más robots (en un área común o no) desean intercambiar información (datos o control) deben hacerlo por mensajes explícitos.
- La sincronización se da por un mecanismo equivalente a un semáforo binario.
- La exclusión mutua puede generarse con la declaración de las áreas alcanzadas por cada robot. Acceder a otras áreas de la ciudad, así como salir de ellas no está permitido.
- Todo el modelo de ejecución es sincrónico y permite la existencia de un reloj virtual de ciclos, que a su vez permite asignar tiempos específicos a las operaciones, simulando la existencia de una arquitectura heterogénea.
- El entorno permite ejecutar el programa de manera tradicional, o paso a paso por instrucciones, dando al usuario un control detallado sobre la ejecución del programa, de manera de poder controlar situaciones típicas de concurrencia tales como conflictos (colisiones) o deadlocks.
- En la ejecución paso a paso, el efecto de las operaciones se puede reflejar en los robots físicos, comunicados vía Wi-fi. Los robots físicos poseen un sistema operativo Linux que permite ejecutar un servidor http implementado en NodeJS [NJS15]. De esta manera el entorno se comunica con los robots (cada robot físico se corresponderá con uno virtual en el ambiente). La comunicación entre ellos es punto a punto, y bidireccional, es decir, el entorno envía las instrucciones al robot físico y luego este último envía su respuesta al entorno indicando la finalización de la instrucción indicada.

### **3 Heterogeneidad en arquitecturas paralelas**

#### 3.1- Conceptos generales

Históricamente se ha buscado incrementar el poder computacional de los sistemas de computación. Sin embargo, se ha llegado a la situación de que resulta difícil acelerar la velocidad de los procesadores incrementando la frecuencia de reloj de los mismos. Son dos los problemas que los arquitectos de hardware deben enfrentar: la generación de calor y el consumo de energía. La solución que presentaron los diseñadores a estos problemas ha sido integrar dos o más núcleos computacionales dentro de un mismo chip, lo cual se conoce como procesador multicore o multinúcleo. Los procesadores multicore mejoran el rendimiento de una aplicación al distribuir el trabajo entre los núcleos disponibles [GEP06] [MCC07].

En una clasificación general actualmente existen dos tipos de multicores de acuerdo a las características de sus núcleos:

- Arquitecturas Multicores Homogéneas: todos sus cores poseen las mismas características.
- Arquitecturas Multicores Heterogéneas: poseen cores con diferentes características en cuanto a rendimiento y consumo de energía, pudiendo utilizar o no distintos ISA (instruction set architecture).

Actualmente la investigación se está enfocando en este segundo tipo de Multicores, dado que tener cores de diferentes tipos permite optimizar el rendimiento, y al realizar una correcta distribución de las tareas entre los núcleos, se logra una mayor eficiencia en la relación rendimiento/energía.

En este tipo de arquitectura, la heterogeneidad se da en diferentes aspectos, los más importantes son la potencia de cómputo (velocidad de cómputo) de los cores; el tiempo de acceso a memoria; la velocidad de comunicación entre cores. Estos tres aspectos definen el Tiempo de Ejecución de las instrucciones en cada core, por lo que una misma sentencia ejecutada en dos núcleos diferentes puede insumir tiempos distintos. Por otro lado, al existir un cierto grado de independencia entre las características que producen la heterogeneidad, no todas las instrucciones se ven influenciadas en la misma proporción. Es decir, una operación de punto flotante ejecutada en el core A, puede tardar la cuarta parte del tiempo que en el core B, mientras que en el caso de una operación de escritura puede tardar la mitad al ejecutarse.

Para analizar el rendimiento logrado por las aplicaciones paralelas sobre estas arquitecturas, se utilizan métricas tradicionales: speedup y eficiencia [DON02].

El speedup (S) es una medida que permite cuantificar el beneficio relativo de resolver un problema en paralelo, es decir cuan “rápido” ejecuta el algoritmo paralelo respecto al secuencial. La función de speedup se define como la relación entre el tiempo del mejor algoritmo secuencial sobre un simple núcleo (Ts) y el tiempo requerido para resolver el mismo problema en una arquitectura paralela [GRA03].

$$S = \frac{TS}{TP}$$

La eficiencia (E) es una medida de la fracción de tiempo para la cual los cores son usados útilmente en la aplicación paralela. Esta métrica se define como la relación entre el speedup logrado y el speedup óptimo ( $S_{opt}$ ) que se puede conseguir en la arquitectura [GRA03].

$$E = \frac{S}{S_{opt}}$$

Tradicionalmente, en las arquitecturas homogéneas, el speedup óptimo está dado por la cantidad de cores utilizados (p). Sin embargo, en el caso de las arquitecturas heterogéneas, se deben tener en cuenta la potencia de cómputo de los diferentes núcleos que la componen, por lo cual se redefine el speedup óptimo como:

$$S_{opt} = \frac{p-1}{\sum_{i=0}^{p-1} \frac{Potencia(Core_i)}{Potencia(Mejor\ Core)}}$$

### 3.2- Tratamiento de la heterogeneidad en CMRE.

La potencia de cómputo de un core está dada por el clock del procesador, en el caso

del multicore heterogéneo, se requiere que cada núcleo tenga uno propio. Esta situación se modela en el CMRE teniendo un clock general del sistema para la simulación (que es el más rápido) y múltiplos de ese clock para los robots. Esto se puede definir como relojes virtuales de cada robot/procesador.

Las operaciones de Depositar y Recoger se pueden asimilar a las de Write y Read en procesadores reales. Naturalmente una arquitectura heterogénea puede tener tiempos diferentes, que en el caso de CMRE serán múltiplos del clock general.

Asimismo hay que considerar los tiempos de la comunicación (Send y Receive) e incluso de operaciones tales como Informar que pueden tener distinto número de ciclos de reloj por procesador/robot.

Dentro del entorno CMRE, se le podrá configurar a cada robot una magnitud de procesamiento relativa a la velocidad patrón: “n veces más lento que”. Además permitirá definir cuantos ciclos de reloj consumirá cada una de sus instrucciones, es decir, configurar la heterogeneidad según la necesidad del ejercicio y del concepto que sea necesario brindar.

Es de hacer notar que para la introducción del concepto, se pueden definir diferencias simples en los relojes virtuales y posteriormente en el costo de ciclos de reloj en las operaciones por robot. Separar los dos aspectos ayuda al alumno a comprender que los tiempos tienen que ver no sólo con el procesamiento, sino también con los accesos a memoria y la comunicación.

### 3.3- Heterogeneidad y Balance de carga.

El balance de carga de una aplicación incide directamente en el rendimiento de la misma. Esto se debe a que si el sistema se encuentra desbalanceado, significa que hay cores que se encuentran ociosos esperando a que otros terminen de trabajar, incrementando el tiempo final de ejecución, y por consiguiente disminuyendo el speedup y la eficiencia. Esto depende de las características de la aplicación y de la arquitectura paralela que se esté utilizando.

Desde el punto de vista de la aplicación, en el caso del entorno CMRE, dependerá mucho de la actividad que cada robot desempeñe en su algoritmo, dado que, según las instrucciones que ejecute y también de la configuración particular de la ciudad, el trabajo podrá o no estar balanceado. Por ejemplo, si los robots deben juntar los papeles de un área privada (de igual tamaño para todos los robots), el balance de carga dependerá de la cantidad de papeles que exista en cada área.

Teniendo en cuenta la arquitectura paralela, la heterogeneidad de los cores es un factor fundamental que incide en el balance de carga. Por ejemplo, si dos robots deben recorrer una avenida de principio a fin, pero tienen diferentes velocidades, es indudable que el más rápido quedará ocioso hasta que el otro termina, lo cual afecta al balance de carga.

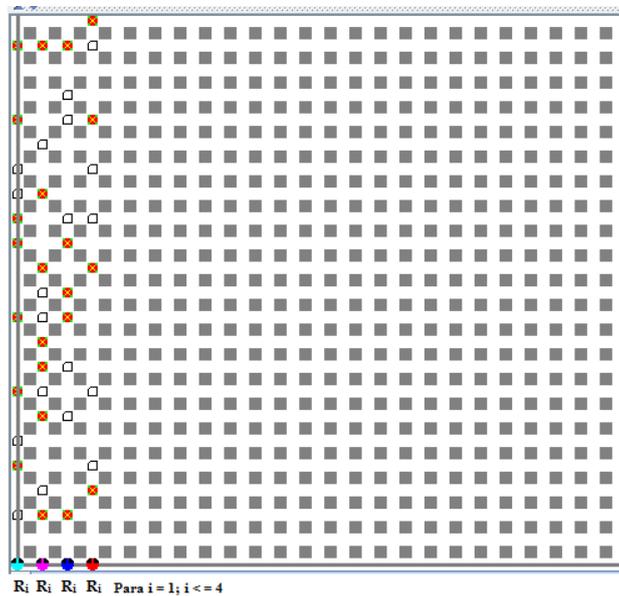
Si combinamos ambos aspectos, el problema del balance de carga se vuelve aún más complejo pero también más desafiante para el alumno a la hora de resolver un algoritmo que sea eficiente, lo que implica una motivación extra a la hora de implementarlo.

3.4- Ejemplo de un caso experimental sencillo para mostrar el impacto de la heterogeneidad.

Se tienen 4 robots con velocidades relativas de 1 paso por cuadra, 2 pasos por cuadra, 3 pasos por cuadra y 4 pasos por cuadra. Esto significa que el robot más rápido (R1) hará por ejemplo 100 cuadras de una Avenida en 100 unidades de tiempo y el más lento (R4) en 400 unidades de tiempo.

Si se tienen 4 Avenidas (Av1, Av2, Av3 y Av4) a recorrer totalmente (1 por robot), con una distribución de objetos a recoger diferente en cada una, por ejemplo 10, 20, 40 y 80 objetos respectivamente.

El tiempo de recoger un objeto  $3T$  (donde  $T$  es el tiempo de dar un paso cada robot).



**Fig. 1.** Esquema del ejemplo a desarrollar.

La asignación del trabajo (en este caso la Avenida a recorrer) será significativa para el tiempo total del programa y para el desbalance de carga:

a- Si los robots los asignamos en forma directa relacionando la velocidad de los mismos con las Avenidas, según el trabajo esperado, se tiene:

$$R1 \text{ recorre la Av4 y tarda } 100 + 80 \times 3 = 340 T$$

$$R2 \text{ recorre la Av3 y tarda } 200 + 40 \times 3 = 320 T$$

$$R3 \text{ recorre la Av2 y tarda } 300 + 20 \times 3 = 360 T$$

$$R4 \text{ recorre la Av1 y tarda } 400 + 10 \times 3 = 430 T$$

El programa tardaría 430 T y el máximo tiempo ocioso sería para R2 (110 T).

La ociosidad total (que es una medida del desbalance de carga) sería de 270 T.

b- Si asignáramos los robots en forma inversa relacionando la velocidad de los mismos con las Avenidas, según el trabajo esperado, se tiene:

$$R1 \text{ recorre la Av1 y tarda } 100 + 10 \times 3 = 130 T$$

R2 recorre la Av2 y tarda  $200 + 20 \times 3 = 260$  T

R3 recorre la Av3 y tarda  $300 + 40 \times 3 = 420$  T

R4 recorre la Av4 y tarda  $400 + 80 \times 3 = 640$  T

Notar que en este caso el programa tardaría 640 T y el máximo tiempo ocioso sería para R1 (510 T).

La ociosidad total crecería a 1110 T → Más de 4 veces el desbalance de la distribución anterior.

#### 4 Consumo energético en algoritmos paralelos

El consumo energético es un punto clave en los procesadores actuales. En general la performance de un algoritmo paralelo no se mide sólo por el tiempo de ejecución del mismo, sino también por la energía consumida. Así aparecen índices que relacionan Flops/Watt o Flops/Joule según se relacione el cómputo con potencia instantánea o energía total [BAL13][BRO10].

Resulta importante en la formación de los alumnos de Informática hacer hincapié en las métricas de consumo como un indicador de calidad de los algoritmos. Asimismo, comprender los mecanismos automáticos que desarrollan los procesadores en función de la temperatura que alcanzan (que es una función directa de la energía consumida en un intervalo de tiempo) [BAL13].

##### 4.2- La limitación del clock de los procesadores y su modelización en CMRE

En general todos los procesadores modernos tienen una curva de ajuste automático del clock en función del consumo (o de la temperatura interna que alcanzan) que sigue la forma de la Figura 2.

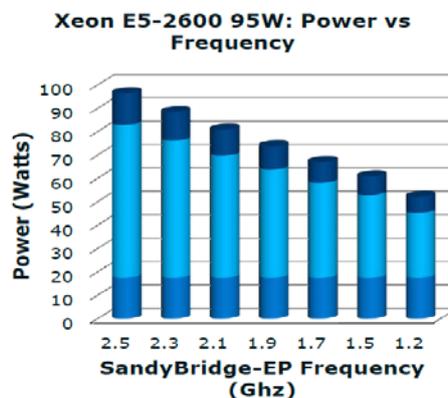
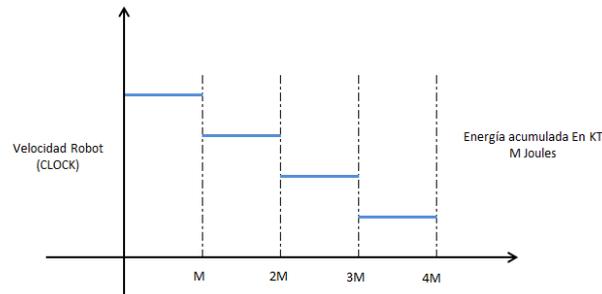


Fig. 2. Gráfico representativo de la curva de ajuste automático del clock.

En determinado punto el reloj se “enlentece” para disminuir el trabajo realizable por el procesador en cada unidad de tiempo y de ese modo alcanzar un punto de equilibrio con menor temperatura. En algunos casos es posible manejar la reasignación de tareas del procesador (por ejemplo controlando información de los contadores de hardware o mediante monitoreo directo), para evitar el cambio de la frecuencia de clock o para adaptarlo al cambio de frecuencia [BAL13].

Modelizar esto en CMRE requiere tener una medida de energía consumida en las operaciones (Mover / Recoger / Depositar / Comunicarse / Informar) y contabilizar la acumulación de energía de cada robot/procesador para ajustar su “velocidad” de acuerdo a un modelo previamente definido.

El esquema se asimila al de la Figura 2 donde se muestra una modelización discreta de la curva mostrada para procesadores reales.



**Fig. 3.** Curva de ajuste automático de la “velocidad” de los robots.

Básicamente en CMRE esto significa “cambiarle” el reloj virtual al robot ajustando la velocidad de movimiento (y por ende la capacidad de trabajo por unidad de tiempo) de acuerdo a una métrica de energía gastada en las últimas K unidades de tiempo.

A los fines didácticos, se han desarrollado ejemplos con  $K=10$  simplemente para considerar un período de acumulación de trabajo que genere cambios en la temperatura interna del procesador.

Con estas ideas se define un consumo energético por operación, partiendo de que MOVER requiere 1 Joule y se le da un costo energético a DEPOSITAR, RECOGER, ENVIAR, RECIBIR e INFORMAR. Con esto es sencillo computar la energía total de cada robot/procesador para un dado algoritmo, y también se pueden comparar algoritmos desde el punto de vista energético.

Es más complejo en el simulador adoptar una estrategia de “cambio de clock” en función del consumo. Actualmente se ha definido que si la energía consumida en los últimos 10 T de un robot cualquiera supera los M Joules (donde M es un parámetro que se fija según la curva indicada en la Figura 3, a ese robot se le reduce un escalón la velocidad (clock) por un mínimo de R ciclos (en este momento se utiliza  $R=3$ ) y luego se reinicia el cálculo. Si el robot pasa a estar por debajo de los M Joules en los últimos 10 T, recupera su clock anterior.

Naturalmente esto implica diferente speedup, diferente eficiencia y diferente consumo máximo por intervalo de 10 T, mirando el algoritmo paralelo que se esté ejecutando. Asimismo, se tiene una medida (virtual) del consumo total de un dado algoritmo.

#### 4.3- Ejemplo de ejercitación experimental

Es relativamente fácil extender el ejemplo relacionado con heterogeneidad (sección 3.4) al caso de consumo y considerar el cambio de clock en función del mismo. Al ejemplo mencionado se agrega que el consumo de una operación de RECOGER es de 5 Joules y la de MOVER de 1 Joule.

- a- Con la primera asignación de trabajo planteada en el ejemplo del punto 3.4 (a) se tiene:

R1 recorre la Av4 y tarda  $100+80 \times 3 = 340$  T y consume  $100+80 \times 5 = 500$  Joules.  
R2 recorre la Av3 y tarda  $200+40 \times 3 = 320$  T y consume  $200+40 \times 5 = 400$  Joules.  
R3 recorre la Av2 y tarda  $300+20 \times 3 = 360$  T y consume  $300+20 \times 5 = 400$  Joules.  
R4 recorre la Av1 y tarda  $400+10 \times 3 = 430$  T y consume  $400+10 \times 5 = 450$  Joules.

El programa tarda 430 T y consume en total 1750 Joules.

Es posible que si se aplica la restricción del reloj, posiblemente a R1 o R2, alguno de los robots DEBA disminuir su velocidad (según la distribución de la carga) y en ese caso será *constante el consumo total, pero el programa tardará más y puede cambiar el desbalance de carga.*

- b- En la asignación inversa del ejemplo del punto 3.4 (b) el consumo sería:

R1 recorre la Av1 y tarda  $100+10 \times 3 = 130$  T y consume  $100+10 \times 5 = 150$  Joules.  
R2 recorre la Av2 y tarda  $200+20 \times 3 = 260$  T y consume  $200+20 \times 5 = 300$  Joules.  
R3 recorre la Av3 y tarda  $300+40 \times 3 = 420$  T y consume  $300+40 \times 5 = 500$  Joules.  
R4 recorre la Av4 y tarda  $400+80 \times 3 = 640$  T y consume  $400+80 \times 5 = 800$  Joules.

Notar que en este caso el programa tardaría 640 T y consumiría en total la misma energía (1750 Joules). Esto es razonable porque hay un trabajo total igual en operaciones de MOVER y DEPOSITAR.

*Sin embargo es menos probable que haya que reducir el clock de cualquiera de los robots...*

## 5 Conclusiones y Líneas de Trabajo Futuro

Los temas de heterogeneidad y consumo energético en arquitecturas paralelas son de gran importancia y se ha presentado la modelización de los mismos sobre el entorno CMRE.

CMRE aparece como una herramienta muy útil para la introducción de estos conceptos, a partir de su empleo desde 2013 con robots/procesadores homogéneos.

Si bien hay modificaciones que se deben realizar en el entorno, resultan transparentes al alumno y la realización de trabajos experimentales orientados a estos temas es muy natural.

Actualmente se está trabajando en la implementación para 2016 de estas extensiones, considerando que al habilitar diferentes relojes para los diferentes robots, aparecen recorridos en los que los conflictos de concurrencia se pueden dar no sólo en las esquinas de la ciudad, sino también en puntos intermedios entre dos esquinas. De esta manera, el nivel de complejidad de los posibles escenarios aumenta, planteando un desafío mucho más ambicioso, que responde a la realidad tecnológica de los procesadores actuales.

## 6 Bibliografía

[ACM04] ACM/IEEE-CS Joint Task Force on Computing Curricula. “Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering”. Report in the Computing Curricula Series. 2004.

- [ACM08] ACM/IEEE-CS Joint Interim Review Task Force. "Computer Science Curriculum 2008: An Interim Revision of CS 2001". Report from the Interim Review Task Force. 2008.
- [ACM13] ACM/IEEE-CS Joint Task Force on Computing Curricula. "Computer Science Curricula 2013". Report from the Task Force. 2013.
- [ACM68] ACM Curriculum Committee on Computer Science. "Curriculum '68: Recommendations for the undergraduate program in computer science". Communications of the ACM, 11(3):151-197. 1968.
- [ACM78] ACM Curriculum Committee on Computer Science. "Curriculum '78: Recommendations for the undergraduate program in computer science". Communications of the ACM, 22(3):147-166. 1979.
- [ACM99] ACM Two-Year College Education Committee. "Guidelines for associate-degree and certificate programs to support computing in a networked environment". New York: The Association for Computing Machinery. 1999.
- [AMD09] AMD. "Evolución de la tecnología de múltiple núcleo". <http://multicore.amd.com/es-ES/AMD-Multi-Core/resources/Technology-Evolution>. 2009.
- [AND00] Andrews G. "Foundations of Multithreaded, Parallel, and Distributed Programming". Addison Wrsley, 2000.
- [BAL13] Balladini J., Rucci E., De Giusti A., Naiouf M., Suppi R., Rexachs D., Luque E. "Power Characterisation of Shared-Memory HPC Systems". Computer Science & Technology Series – XVIII Argentine Congress of Computer Science Selected Papers. Págs. 53-65. 2013.
- [BRO10] Brown D. J., "Toward Energy-Efficient Computing", Magazine Communications of the ACM Volume 53 Issue 3, March 2010
- [DAS89] Dasgupta S. "Computer Architecture. A Moder Synthesis. Volume 2: Advanced Topics". Jhon Wilet & Sons. 1989.
- [DEG14a] De Giusti, L., Leibovich, F., Sanchez, M., Chichizola, F., Naiouf, M., De Giusti, A. "Desafíos y herramientas para la enseñanza temprana de Concurrencia y Paralelismo". Congreso Argentino de Ciencias de la Computación (CACIC), 2014.
- [DEG14b] De Giusti, A., De Giusti L., Leibovich, F., Sanchez, M., Rodriguez Eguren, S. "Entorno interactivo multirrobot para el aprendizaje de conceptos de Concurrencia y Paralelismo". Congreso Tecnología en Educación, Educación en Tecnología. 2014.
- [DIJ78] Dijkstra E. W. "Finding the Correctness Proof of a Concurrent Program". In Program Construction, International Summer School, Friedrich L. Bauer and Manfred Broy (Eds.). Springer-Verlag, 24-34, 1978.
- [DON02] Dongarra J. , Foster I., Fox G., Gropp W., Kennedy K., Torzcon L., White A. "Sourcebook of Parallel computing". Morgan Kaufmann Publishers 2002. ISBN 1558608710.
- [GEP06] Gepner P., Kowalik M.F. "Multi-Core Processors: New Way to Achieve High System Performance". In: Proceeding of International Symposium on Parallel Computing in Electrical Engineering 2006 (PAR ELEC 2006). Pags. 9-13. 2006.
- [GRA03] Grama A., Gupta A., Karypis G., Kumar V. "Introduction to Parallel Computing". Pearson – Addison Wesley 2003. ISBN: 0201648652. Segunda Edición (Capítulo 3).
- [HAN77] Hansen P. B. "The Architecture of Concurrent Processes". Prentice Hall, 1977.
- [HOA85] Hoare C. "Communicating Sequential Processes". Prentice Hall, 1985.
- [HOO13] Hoonlor A., Szymanski B. K., Zaki M. J., Thompson J. "An Evolution of Computer Science Research". Communications of the ACM. 2013.
- [MCC07] Mc Cool M, "Programming models for scalable multicore programming", 2007, <http://www.hpcwire.com/features/17902939.html>.
- [MCC08] McCool M. "Scalable Programming Models for Massively Parallel Multicores". Proceedings of the IEEE, 96(5): 816–831, 2008.
- [NJS15] <https://nodejs.org/api/http.html>