INNOVACIÓN Y COMPUTACIÓN PARALELA

Marcos Hernández Marcelino[1]
E.T.S.I.T. de la Universidad de Málaga – UMA
marcosherma@gmail.com

RESUMEN

En el presente artículo se aborda el estudio de la computación paralela ligado a la innovación. Se repasan conceptos básicos sobre las teorías de la innovación para después centrarnos en el estudio de la programación paralela. Justificaremos su existencia y por qué puede considerarse innovación. Haremos un repaso breve de las realizaciones más relevantes y los múltiples usos que tiene éste tipo de computación.

PALABRAS CLAVE: innovación – computación – paralela – arquitectura de computadores – programación paralela.

CLASIFICACIÓN JEL: O39 (Cambio tecnológico; Investigación y Desarrollo; Otros).

ABSTRACT

In this paper, we focus on the study of parallel computing linked to innovation. We are going to review basic concepts of innovation. Then, we pay attention on theoretical concepts about serial and parallel computing. Besides, we are going to justify why parallel computation is considered as innovation. We are going to see the most useful realizations of parallel computing. Finally, some investigation lines are purposed.

KEY WORDS: innovation – computing – parallel – computer architecture – parallel programming.

JEL CLASSIFICATION: O39 (Technological Change; Research and Development; Other).

1. INTRODUCCIÓN

En las últimas décadas, hemos sido testigos de numerosos cambios en el mundo. El surgir de nuevas potencias económicas que han sabido adaptarse al mundo global en el que vivimos, como Japón y Corea del Sur. O, por contra, el paulatino decaimiento de otras regiones, como España. Por supuesto, nada es fruto de la casualidad.

Vivimos en un mundo global, dominado por las Tecnologías de la Información y la Comunicación (TIC). Sin duda, la Investigación, Desarrollo e Innovación (I+D+i) es fundamental en éste campo, ya que contribuye al avance tecnológico que posibilitará el desarrollo de una región en el panorama mundial.

En el siguiente cuadro se muestra la evolución del gasto en I+D+i[2] de algunos países de la Unión Europea, Japón y Estados Unidos

Tabla 1: Evolución del gasto en I+D (porcentaje del PIB).

	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
UE 27	1.84	1.86	1.87	1.88	1.87	1.83	1.82	1.85	1.85	1.92	2.02	2.01	2.03
Alemania	2.41	2.47	2.47	2.5	2.54	2.5	2.51	2.54	2.53	2.69	2.82	2.8	2.84
Dinamarca	2.18	2.24	2.39	2.51	2.58	2.48	2.46	2.48	2.58	2.85	3.16	3.07	3.09
Finlandia	3.17	3.35	3.32	3.36	3.44	3.45	3.48	3.48	3.47	3.7	3.94	3.9	3.78
Francia	2.16	2.15	2.2	2.24	2.18	2.16	2.11	2.11	2.08	2.12	2.27	2.24	2.25
Suecia	3.58		4.13		3.8	3.58	3.56	3.68	3.4	3.7	3.6	3.39	3.37
EE.UU.	2.63	2.69	2.71	2.6	2.6	2.53	2.58	2.62	2.69	2.82	2.87		
Japón	2.98	3	3.07	3.12	3.14	3.13	3.31	3.41	3.46	3.47	3.36		
España	0.86	0.91	0.92	0.99	1.05	1.06	1.12	1.2	1.27	1.35	1.39	1.39	1.31
Eslovaquia	0.66	0.65	0.63	0.57	0.57	0.51	0.51	0.49	0.46	0.47	0.48	0.63	0.68
Letonia	0.36	0.45	0.41	0.42	0.38	0.42	0.56	0.7	0.6	0.62	0.46	0.6	0.7
Italia	1.02	1.04	1.08	1.12	1.1	1.09	1.09	1.13	1.17	1.21	1.26	1.26	1.25
Rumanía	0.4	0.37	0.39	0.38	0.39	0.39	0.41	0.45	0.52	0.58	0.47	0.46	0.48

[1] Estudiante de la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universidad de Málaga, especialidad Telemática.

[2] Léase inversión en I+D+i, ya que produce beneficios a corto y medio plazo.

A simple vista, podemos ver que los países punteros en la actualidad son los que más gasto han tenido y tienen en I+D+i. Por tanto, parece existir una relación directa entre dicha inversión y el crecimiento económico. Si utilizamos ésta cifra como referencia para medir el crecimiento económico de los países, aquellos que más inviertan[3] en I+D+i están llamados a ser las potencias mundiales en un futuro próximo.

Aunque parezca novedoso, éste cambio se produjo tiempo atrás. Entre los años 80 y 90, empezaron a evidenciarse las limitaciones teóricas del modelo convencional neoclásico. Se empieza a buscar qué papel juega la innovación y el cambio tecnológico en los factores de producción.

Nuevos modelos de crecimiento económico empiezan a surgir. En ellos, el cambio tecnológico tiene un papel fundamental. Es aquí donde comienza el interés por la inversión en un factor endógeno llamado *innovación tecnológica* (Romer, 1990). Durante el mismo periodo de tiempo, comienzan a tomar importancia las teorías basadas en la globalización y aquellas centradas en las empresas.

Uno de los pilares del nuevo modelo económico pasó a ser el capital humano. No se puede concebir una economía que apueste claramente por I+D+i sin la inversión correspondiente en educación. Educación es invertir en capital humano, que es el eje fundamental del desarrollo tecnológico y las actividades de I+D+i. Sin duda, esta educación deberá tener su parte de escolarización (*schooling*) y su aprendizaje práctico (*learning-by-doing*) (Lucas, 1988).

Surge así un nuevo pensamiento económico denominado *economía de la innovación y del cambio tecnológico*. Una de sus ideas fundamentales es la fuerte relación entre el crecimiento de un país y su nivel de desarrollo tecnológico (medido por el número de científicos e ingenieros, el gasto en I+D+i, el número de patentes otorgadas y el número de artículos publicados).

En lo competente a innovación, no podemos obviar la aportación de J. A. Schumpeter (Schumpeter, 1939). Destacado por sus teorías sobre la innovación unida al aumento de la prosperidad, planteó el proceso innovador (interconexión entre invención, innovación y difusión) como una secuencia holística. Centra su interés en el resultado del proceso, y no en el proceso en sí.

Define la innovación como producir otras cosas, o las mismas por métodos distintos. Normalmente, asocia la innovación con tecnología. Sin embargo, dicha innovación ha de entrar al mercado para considerarla como tal. La fuente de la innovación serán las nuevas empresas y los empresarios innovadores. Para Schumpeter, los protagonistas del sistema económico. Los empresarios tendrán que ser elevados a los estratos más altos de la sociedad.

El objetivo de la innovación busca lo que él mismo denomina como proceso de *destrucción creadora*. Se trata de irrumpir en el mercado con una innovación que obligue a la reordenación del tejido empresarial, llegando incluso a destruir lo anterior. Éste proceso lo toma como esencia del capitalismo.

Su visión de la economía basada en el proceso innovador no está muy alejada de la realidad actual. La Organización para la Cooperación y el Desarrollo Económico (OCDE) amplía las ideas de Schumpeter en su obra más importante, el Manual de Oslo. En ella, se toman en cuenta los denominados *intangibles* y se amplía en concepto de innovación a aspectos no tecnológicos. Define cuatro tipos de innovación: de producto, de proceso, de organización y de *marketing* (Manual de Oslo, 2006). Además, considera la innovación en empresas de servicios.

Así mismo, se concibe la innovación tanto en el ámbito privado como público. Ya que las innovaciones producen crecimiento económico, la inversión por parte de organismos públicos en actividades relacionadas con ello (como educación o I+D+i) debe ser importante. Así, atraerá también inversiones de organismos privados. Gracias a dicho crecimiento económico, el bienestar de la población se verá recompensado gratamente, y la productividad económica aumentará proporcionalmente.

La evidencia empírica demuestra que, en casos exitosos de inversión en I+D+i (como Finlandia o Japón), se produce crecimiento económico. La inversión, tanto pública como privada, implica la aparición de innovaciones orientadas al mercado. Ellas conducen a beneficios, los cuales se vuelven a invertir en I+D+i. Así se han generado los sistemas virtuosos de innovación en la actualidad.

Como el I+D+i es una actividad transversal, repercutirá en la mayor parte las actividades de una región.

[3] Se considera buena inversión entre un 3% y un 4% del Producto Interior Bruto (PIB).

Implícitamente, estas actividades generan avance y difusión del conocimiento, pieza fundamental de la innovación. Ello repercute en el avance y bienestar social, ya que por medio del conocimiento y la investigación, se pueden dar soluciones a muchos problemas presentes en la sociedad.

En el ámbito puramente empresarial, la apertura de nuevas líneas de investigación permite diversificar la producción, pudiendo entrar a nuevos mercados. Además, se aumentará la eficiencia y disminuirá los costes de los procesos productivos. Empresas con alto grado de I+D+i serán las punteras en sus respectivos campos, aumentando cada vez más su prestigio y credibilidad.

La importancia de las actividades de I+D+i en el mundo actual es clara. Actualmente, es decisiva en el devenir de una región en un futuro a corto o medio plazo.

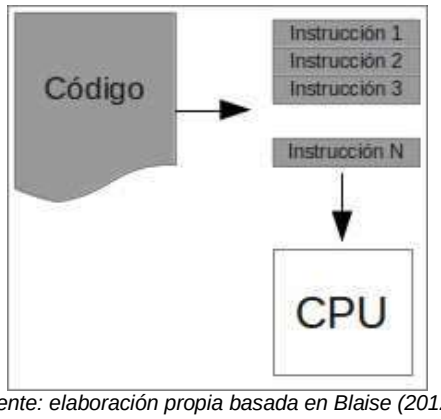
Dentro del campo de la innovación, la presente contribución se va a centrar en la computación paralela. Para ello se abordará un breve estudio sobre la computación secuencial y los factores que la limitan. Así se justificará la importancia creciente de la computación paralela para resolver las limitaciones del modelo secuencial. Estudiaremos los modelos de computación paralela y las realizaciones de dichos modelos. Daremos un breve repaso de las aplicaciones, y finalmente, se propondrán futuras líneas de trabajo.

2. MARCO TEÓRICO

2.1. COMPUTACIÓN SECUENCIAL

Tradicionalmente, el código se ha escrito para ser ejecutado secuencialmente (Blaise, 2012). Es decir, las instrucciones que componen el código, las ejecuta de manera ordenada la Unidad Central de Proceso (*Central Processing Unit, CPU*). Una única instrucción puede ser ejecutada en el mismo instante de tiempo.

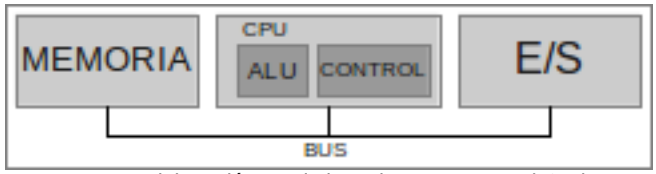
Ilustración 1. Resolución secuencial de un problema



Fuente: elaboración propia basada en Blaise (2012).

En la actualidad, dichos programas son ejecutados por computadoras. Es decir, máquinas de propósito general. De una u otra forma, aunque la tecnología cambia, la mayoría de ellas se implementan siguiendo la arquitectura de John von Neumann (Neumann, 1945; Godfrey, Hendry, 1993). Dicha arquitectura se compone de cinco partes básicas: memoria, unidad aritmético-lógica (*Arithmetic Logic Unit, ALU*), unidad de control (control), dispositivo de entrada y salida (E/S) y el bus.

Ilustración 2. Arquitectura de John von Neumann



Fuente: elaboración propia basada en Neumann (1945).

La característica principal de dicha arquitectura es que tanto el programa como los datos, son almacenados en el mismo medio físico (concepto de programa almacenado, desarrollado también por John Presper Eckert y John William Mauchly entre otros). Anteriormente, el programa era una parte de la máquina[4].

[4] Como podía ser las tarjetas perforadas.

2.2. FACTORES LIMITANTES

Sin embargo, la ejecución secuencial de código tiene sus limitaciones. Principalmente, el rendimiento de una máquina se aumentaba subiendo su frecuencia de funcionamiento[5]. Pero, el consumo de potencia de los circuitos viene dada por,

$$P = C \times V^2 \times F$$

Donde P es la potencia consumida, C el cambio de capacitancia de los transistores presentes en los *chips*, V la tensión de trabajo y F la frecuencia de procesamiento. Podemos ver que la potencia consumida es directamente proporcional a la frecuencia de procesamiento. Por ello, al subir la frecuencia, estamos subiendo el consumo de potencia.

Otro problema es el límite de la distancia de integración. Disminuirla significa poder aumentar la frecuencia de funcionamiento de los transistores, disminuir el voltaje de alimentación, aumentar el número de transistores en el mismo área de integración y abaratar costes de fabricación del microprocesador (Ujaldón, 2001: 68-78). Pero existe un límite teórico del cual no se puede bajar[6].

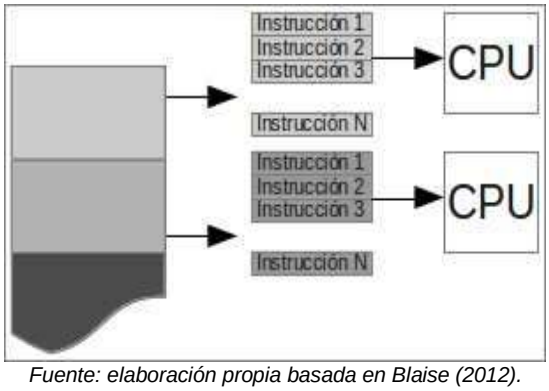
Sin embargo, existen relaciones entre los anteriores parámetros que no hacen la solución tan sencilla. Subir la frecuencia de conmutación sin bajar la distancia de integración, acarreará consumos de potencia poco deseados. Además, el hecho de disminuirla, hará que los transistores pidan más voltaje. Adicionalmente, al aumentar el número de transistores (Moore, 1965; Moore 1975) podremos usarlos para aumentar memoria u otras unidades funcionales[7], haciendo más largo en camino crítico (Ujaldón, 2003: 14-15), condicionando la frecuencia máxima de funcionamiento.

Existe otro factor físico limitante, la transmisión de los datos entre memoria y *CPU*. La velocidad de procesado ha aumentado considerablemente a lo largo del tiempo. Sin embargo, la velocidad a la que memoria comunica los datos no ha seguido su evolución. Por tanto, puede ocurrir que la *CPU* quede ociosa esperando datos, lo que disminuye la eficiencia.

2.3. LA COMPUTACIÓN PARALELA

Por ello, es de especial relevancia el estudio de la programación paralela. En ella, el problema es dividido en varias partes. Cada parte podrá ser ejecutada simultáneamente usando distintos recursos de la computadora. Dichos recursos podrán ser: una única computadora con múltiples procesadores (puede ser la arquitectura von Neumann repetida), un número aleatorio de computadores conectados en una misma red, o una combinación de ambos (Blaise, 2012).

Ilustración 3. Resolución paralela de un problema



Fuente: elaboración propia basada en Blaise (2012).

El problema que queremos resolver de forma paralela, se tiene que poder dividir en pos de trabajar sobre

---

[5] Desde 1MHz hasta velocidades comprendidas entre 1GHz y 4GHz en apenas 30 años. Unas 1000 veces más rápido.

[6] El límite en cuanto a frecuencia de funcionamiento es de 10Ghz, que corresponde a tecnologías de fabricación de 0.02 micras. Por ello, se están estudiando nuevas formas de computación como la computación cuántica.

[7] Aumentar el número de unidades funcionales y dividirlas en etapas conllevó la aparición de procesadores que explotaban el paralelismo a nivel de instrucción (procesadores segmentados, supersegmentados y superescalares).

las partes simultáneamente. Como resultado, deberá tardar menos en resolverse que secuencialmente. Con ésta técnica, se pueden resolver problemas muy complejos en tiempo muy reducido.

La innovación en éste ámbito radica en resolver problemas de una forma más eficiente. Incluso permite enfrentarse a nuevos problemas de mayor magnitud, inimaginables de abordar mediante la computación secuencial.

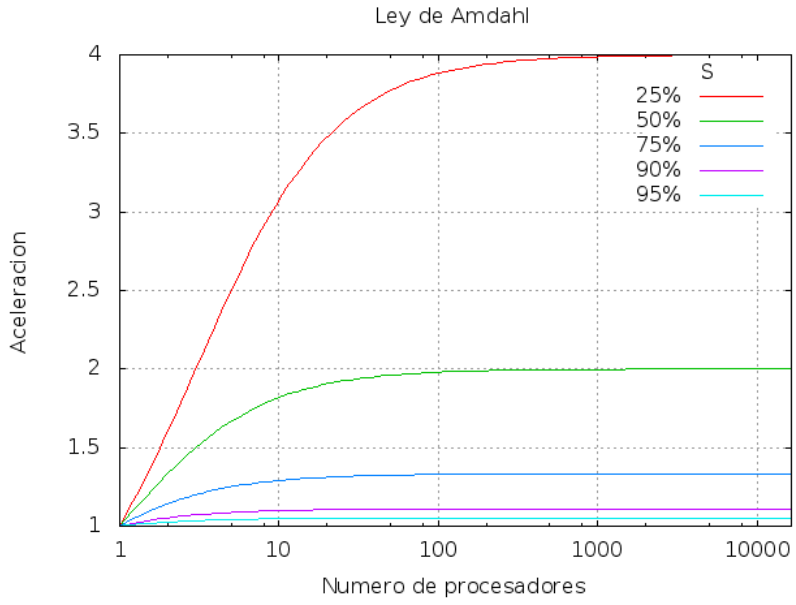
Existen tres niveles básicos de paralelismo. Dado que un programa es una secuencia de instrucciones, éstas pueden ser cambiadas de orden y ejecutarse varias a la vez, siempre y cuando el resultado final no se vea alterado. Se conoce como paralelismo a nivel de instrucción. Cuando hay que tratar un gran volumen de datos o hacer las mismas operaciones una y otra vez, tenemos paralelismo de datos. Finalmente, cuando cada procesador realiza una tarea diferente (distintas operaciones en cada procesador), tendremos paralelismo a nivel de tareas.

Por medio de la Ley de Amdhal, se puede calcular la mejora en velocidad de ejecución de un programa utilizando múltiples procesadores. Siguiendo la fórmula siguiente, siendo A la aceleración obtenida, S la parte secuencial del código (por lo que $P = 1 - S$ será la parte paralela) y N el número de procesadores.

$$A = \frac{1}{S + \frac{1-S}{N}}$$

Representado gráficamente la anterior Ley, para diferentes porciones de código paralelo,

Ilustración 4. Ley de Amdahl para diferentes porciones de código secuencial



Fuente: elaboración propia utilizando GNUPlot.

A menor cantidad de código secuencial, se consigue una mayor aceleración. Se observa que existen límites teóricos para la mejora que se puede conseguir haciendo el código cada vez más paralelo. Sin embargo, ciertos problemas mejoran su rendimiento incrementando su tamaño.

Mediante la computación paralela se resuelven algunos problemas derivados del uso de un único procesador. El más importante de ellos es el consumo de potencia. Parece claro que seguir aumentando la frecuencia de funcionamiento y el número de transistores de un único procesador (sabiendo que se llegará al límite de distancia de integración) no es una solución. Por ello, cada vez es más clara la tendencia de aumentar rendimiento incrementando el número de procesadores.

Sin embargo, aparecerán otros problemas antes inexistentes. Muchos de ellos derivan directamente de que el código paralelo es más complejo de escribir que el secuencial. Aparecen problemas de comunicación y

sincronización entre diferentes tareas, dependencias de datos, mayor complejidad en el acceso a datos, desarrollo de compiladores y entornos de programación para los sistemas paralelos, entre otros.

2.4. CLASIFICACIÓN DE COMPUTADORAS

A la hora de clasificar la arquitectura de una máquina, existen diversas formas de hacerlo. La que más se usa hoy en día es la taxonomía de Flynn (Flynn, 1972), que será la usada en el presente artículo. En ella, se usa el concepto de flujo de datos o instrucciones que serán ejecutados u operados en el procesador. Además, ambos flujos podrán ser de un único elemento (instrucción o dato) o múltiples elementos. Así se obtiene directamente que la clasificación es: *SISD* (*Single Instruction, Single Data*), *SIMD* (*Single Instruction, Multiple Data*), *MISD* (*Multiple Instruction, Single Data*) y *MIMD* (*Multiple Instruction, Multiple Data*).

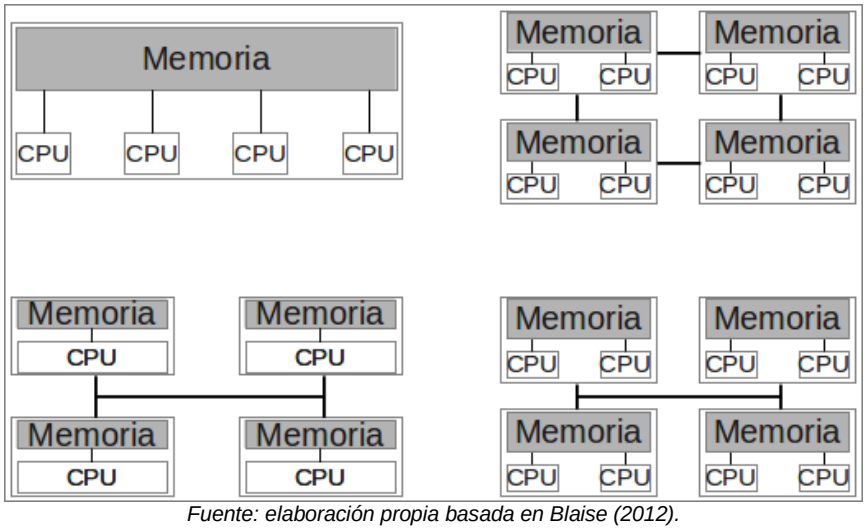
La anterior clasificación no tiene en cuenta cómo se distribuye la memoria. Si nos atenemos a éste criterio, distinguimos entre: memoria compartida, memoria distribuida e híbrido.

- **Memoria compartida.** Todos los procesadores tendrán acceso a toda la memoria con un espacio de direcciones global. Una ventaja es la rápida comunicación entre procesadores a través de la memoria común. Sin embargo, tendremos problemas de sincronización a la hora de acceder a los datos, pues son comunes para todos. Según los tiempos de acceso a memoria, también podemos tener arquitecturas con acceso a memoria uniforme (*UMA*, *Uniform Memory Access*) y arquitecturas de acceso a memoria no uniforme (*NUMA*, *Non-Uniform Memory Access*). En las primeras, los procesadores tendrán la misma latencia en el acceso a memoria, mientras que en las segundas, cada procesador podrá tener tiempos de acceso diferentes. Tendrá problemas de escalabilidad, ya que al aumentar el número de procesadores, aumenta el tráfico de datos entre CPU y memoria.

- **Memoria distribuida.** Cada procesador tendrá su propia memoria local, con su propio espacio de direcciones. Operarán de forma independiente en su propia memoria. Sin embargo, se presenta la dificultad para el programador de sincronizar y comunicar un procesador que necesite datos de una memoria que no es la suya. Para ello, se necesita una red de interconexión entre los procesadores. En éste caso, sólo tenemos arquitecturas NUMA. Ésta arquitectura solventará problemas de escalabilidad, pues incrementos de procesadores obtienen un incremento proporcional de memoria. No se necesita nada en la memoria de un mismo procesador. Es más difícil de programar.

- **Híbrido (distribuida-compartida).** Mezcla las dos arquitecturas anteriores. Existirán varios procesadores que compartan una misma memoria local (aspecto de memoria compartida). A su vez, estarán conectados con otro grupo de procesadores que comparten otra memoria local diferente (aspecto de memoria distribuida) mediante una red de interconexión. Actualmente, es la arquitectura más usada.

Ilustración 5. De derecha a izquierda y de arriba a abajo: Memoria compartida UMA, memoria compartida NUMA,memoria distribuida e híbrida.



Fuente: elaboración propia basada en Blaise (2012).

2.5. MODELOS DE COMPUTACIÓN PARALELA

Al igual que en la computación secuencial, existen varios modelos para la computación paralela. Éstos son

una mera abstracción entre el *software* y el *hardware*. Sin embargo, no existe un modelo único como lo es el modelo de von Neumann para la computación secuencial. A continuación se describen éstos modelos.

Modelo de memoria compartida. Los hilos de ejecución o tareas comparten un mismo espacio de direcciones. Sin embargo, se puede simular en memoria distribuida, teniendo en cuenta la penalización por acceder a un espacio de memoria que no corresponda al procesador donde se ejecuta la tarea o hilo. Los hilos o tareas accederán a la memoria de forma concurrente. Por ello se tendrán que habilitar mecanismos de control de acceso a la memoria[8].

Paso de mensajes. Cuando varios procesos se están ejecutando y necesitan intercambiar información, una forma de hacerlo es mediante paso de mensajes. Esta comunicación puede ser tanto síncrona como asíncrona, y puede ser utilizado también como método de sincronización. En éste modelo, el programador es el encargado de llevar a cabo todo el paralelismo. Éste se suele usar para arquitecturas de memoria distribuida.

Modelo de paralelismo de datos. En éste modelo, se trabaja con un conjunto de datos muy grande. Suele estar organizado de una forma conocida (por ejemplo, *arrays* de dimensiones múltiples). Los procesos trabajan sobre zonas diferentes de los datos. Normalmente, se aplica la misma operación a las porciones de datos. En arquitecturas de memoria compartida, los procesos podrán tener acceso a todos los datos. Para memoria distribuida, habrá que partir la estructura de datos y repartirla[9]. Éste modelo se puede clasificar como *SIMD*.

Modelo híbrido. Es el resultado de combinar al menos dos de los modelos anteriores.

Si vamos a un nivel de abstracción mayor, encontramos otros dos modelos para la computación paralela: *SPMD* y *MPMD*.

SPMD (Single Program Multiple Data, Programa Único Múltiples Datos). Se trata de múltiples procesos ejecutando las mismas instrucciones sobre un conjunto de datos diferente.

MPMD (Multiple Program Multiple Data, Múltiples Programas Múltiples Datos). Cada proceso ejecuta un programa diferente operando sobre datos diferentes. Es menos usado que *SPMD*.

Tanto el modelo SPMD como MPMD pueden ser realizados por cualquier combinación de modelos anteriormente explicados (memoria compartida, paso de mensajes, paralelismo de datos o híbrido).

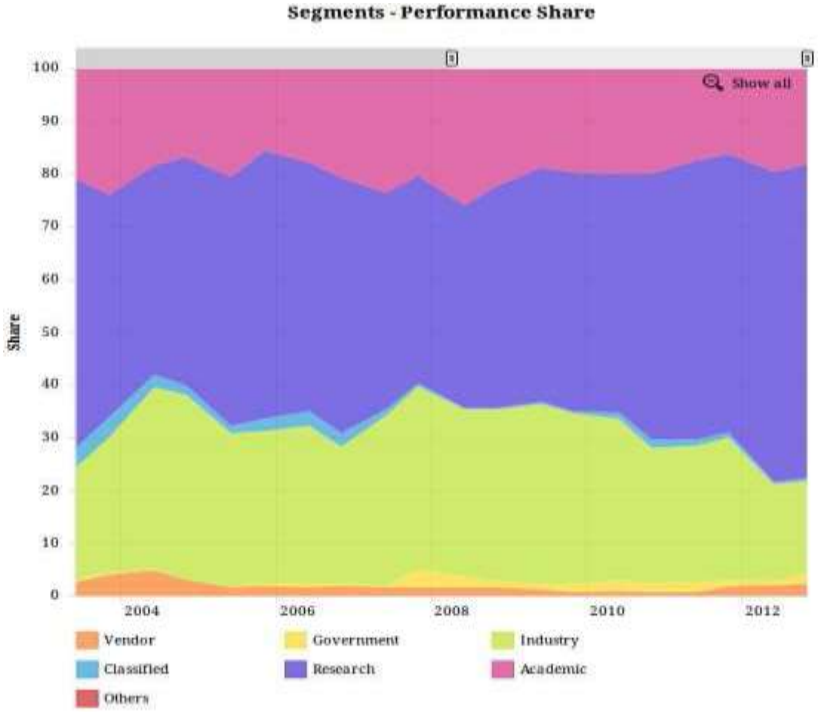
2.6. SITUACIÓN ACTUAL

En sus inicios, la computación paralela parecía destinada a un campo específico y cerrado de la ingeniería. En la Ilustración 6 (top500, 2012), vemos el uso de la computación paralela en varios segmentos productivos. En general, se observa que no es algo que podamos obviar. En el área de la investigación (*Research*), cada vez está cobrando una mayor importancia.

[8] Por ejemplo semáforos, monitores o “locks”.

[9] Una de las dificultades a la hora de programar estriba en saber repartir los datos entre los diferentes procesos.

Ilustración 6. Uso de computación paralela en distintos segmentos



Fuente: top500 (2012).

Hoy en día, la computación paralela es una realidad. Cualquier dispositivo electrónico (teléfonos móviles, ordenadores personales, procesadores gráficos,…) incorpora múltiples núcleos. Encontrar un ordenador personal con un único núcleo es hoy un desafío. La programación en arquitecturas paralelas ya no es algo opcional. Se ha transformado en un asunto de interés para todos aquellos segmentos relacionados con la mejora de rendimiento en la programación.

3. EVIDENCIA EMPÍRICA

En ésta sección, se hará una visión global de las realizaciones de los modelos explicados (Asenjo, 2012: 4). Se describirán cualitativamente y proporcionaremos referencias de cada una para que el lector interesado pueda profundizar en su estudio.

Modelo de memoria compartida

Aunque existen varias formas de realizar éste modelo, nos vamos a centrar en el uso de *threads* (hilos de ejecución) y en el concepto de *programación basada en tareas*.

La instancia de un programa (proceso) puede tener varios hilos (*threads*) ejecutándose de manera concurrente. Los hilos, comparados con los procesos, son más rápidos en cuanto a tiempo de conmutación entre ellos y creación y terminación de cada uno. Además, comparten el mismo espacio de direcciones.

El uso de hilos ofrece algunas ventajas. La comunicación entre hilos es más simple que entre procesos. Permite solapar trabajo de la CPU con operaciones de entrada y salida. Disminuye el número de cambios de contexto de los procesos. Por tanto, la programación con hilos será más sencilla en términos generales.

El concepto de hilo de ejecución es antiguo, y cada fabricante tenía su propia implementación. Entre toda esa variedad, el paso del tiempo y los esfuerzos realizados por llegar a un estándar común, han hecho que las más utilizadas sean *POSIX Threads* (comúnmente conocido como *Pthreads*) y *OpenMP*.

La primera de ellas es un estándar de *POSIX*, implementado para lenguaje C mediante un fichero de cabecera y una biblioteca. Se puede usar tanto en arquitectura de memoria *UMA*, las cuales representan multiproceso simétrico (*SMP, Symmetric Multi-Processing*) como en computación secuencial para simular procesamiento paralelo.

La segunda es *OpenMP* (OpenMP, 2012), una interfaz de programación de aplicaciones (*API, Application Programming Interface*). En éste caso, está disponible para lenguaje C, C++ y Fortran. Se basa en el

modelo “fork-join”. Es decir, un hilo principal se ejecuta secuencialmente hasta la primera región marcada como paralela. Mediante un “fork” se crean los hilos que se precisen. Una vez terminado el trabajo en la región paralela, los hilos se sincronizan y terminan, dejando solitario al hilo central. Así hasta completar el programa.

Con la aparición del concepto de programación basada en tareas, surgieron nuevas formas de programar en arquitecturas de memoria compartida. Las tareas son tratadas de forma diferente. A la hora de planificarlas, no se hará de forma justa. Se sacrifica la justicia del planificador en pos de conseguir mayor eficiencia. Además, las tareas no necesitarán copiar los datos del proceso, como pasa en las operaciones *fork*. Por ello, su creación y destrucción será mucho más ligera que en los procesos.

Éste nuevo concepto fue ampliamente aceptado, y se desarrollaron muchos métodos para explotar las tareas. El objetivo que se persigue en la mayoría de éstas realizaciones es abstraer al programador de la arquitectura subyacente, aumentando así su productividad.

Intel concibe una biblioteca para C++ basada en plantillas llamada *Intel Threading Building Blocks (Intel TBB)* (Intel TBB, 2012) e *Intel Concurrents Collections (Intel CnC)*, también para C++. En el segundo caso, no es necesario especificar qué operaciones se harán en paralelo. Únicamente se centra en qué orden se hacen dichas operaciones. A partir de la versión 3.0 de *OpenMP*, también se introduce el concepto de tareas. En *Microsoft* se desarrolla *Task Parallel Library (TPL)*.

Paso de mensajes

Éste mecanismo es necesario para comunicar procesos o hilos tanto en memoria compartida como distribuida. En el primer caso, podemos no querer compartir el espacio de direcciones con el resto de los hilos. En el segundo caso, queremos comunicar procesos que estén en un sistema distribuido, por lo que éste mecanismo nos permitirá ponerlos en contacto. Además del propio intercambio de información, podemos estar interesados en usar el paso de mensajes como otro método de sincronización.

Sin duda, hablar de paso de mensajes hace recordar *Message Passing Interface (MPI)* (MPI, 2012). Dadas la múltiples implementaciones de éste modelo, MPI fue creado con la idea de estandarizar. De nuevo, el objetivo es conseguir eficiencia a la hora de programar, además de conseguir portabilidad y flexibilidad al estandarizarlo. La interfaz está especificada para los lenguajes C, C++ y Fortran en forma de biblioteca.

Existen muchas más realizaciones de éste modelo. *Parallel Virtual Machine (PVM)* dejó de ser usado tras aparecer MPI. *Global Address Space Networking (GASNet)* (GASNet, 2012) o *SHMEM* son otras de las muchas realizaciones que podemos encontrar.

Modelo híbrido

Cada vez están cobrado una mayor importancia. Un caso de especial interés es el estudio de la evolución de los procesadores gráficos. En ésta clase de procesadores, se optó por no aumentar en frecuencia, sino, multiplicar en número de unidades funcionales[10]. El uso de procesadores gráficos para cómputo de propósito general (*General-Purpose Computing on Graphics Processing Units*, *GPGPU*) está cobrando un gran interés.

En concreto, para las tarjetas gráficas de *nVIDIA* (nVIDIA, 2012), se trabaja con CUDA (*Compute Unified Device Architecture*), un conjunto de herramientas y un compilador creados por ellos mismos. Permite programar en los procesadores gráficos *nVIDIA* en una modificación del lenguaje C.

OpenCL (Open Computing Language) (OpenCL, 2012) también está adquiriendo importancia. Basado en el lenguaje C, consta de un lenguaje de programación y una *API* para programar todo tipo de procesadores (CPUs, GPUs,...). Está orientado tanto al modelo de paralelismo de datos como a la programación basada en tareas.

Aparte de los modelos comentados anteriormente, existen lenguajes de programación paralela propiamente dichos. A su vez, se pueden dividir en dos tipos diferentes.

Partitioned Global Address Space (PGAS)

[10] Notar la diferencia entre multi-core (en los ordenadores personales con 4 y 8 núcleos) y many-core (como en el caso de los procesadores gráficos, con más de 1000 núcleos).

Tenemos un conjunto de lenguajes de programación que simulan una memoria global compartida. Se suele utilizar para arquitecturas de memoria compartida. Como en casos anteriores, se pretende simplificar la tarea del programador abstrayéndolo de detalles de la arquitectura a más bajo nivel.

Sin embargo, todo abstracción conlleva una penalización. Por ello, para usar éstos lenguajes, hay que tener en cuenta a qué posiciones de memoria accedemos. Como realmente no hay una memoria global compartida, acceder a una posición que no nos pertenezca será penalizado con pérdida de tiempo, es decir, la eficiencia se verá mermada.

Las implementaciones de los PGAS son extensiones de lenguajes ya existentes. Algunos de ellos son *UPC (Unified Parallel C)*, que es una extensión de C (UPC, 2012), *CAF (Co-Array Fortran)*, una extensión de Fortran (CAF, 2012) y *Titanium*, una forma de hacer código paralelo en Java (Titanium, 2012).

High Performance Computing Systems (HPCS)

Darpa (*Defense Advanced Research Projects Agency*) lanzó un programa para investigar sobre los sistemas de computación de altas prestaciones (HPCS, 2002). En dicho programa participaron importantes empresas como son *Cray*, *IBM* y *Sun*. Los objetivos son mejorar el rendimiento de las aplicaciones, reducir el tiempo y esfuerzo dedicado al desarrollo, portabilidad y robustez de las mismas.

Para ello, las empresas antes citadas desarrollaron su propio lenguaje de programación. Por parte de Cray tenemos *Chapel* (*Cascade High Productivity Language*) (Chapel, 2002). IBM trabaja en X10 (X10, 2002). Finalmente, Sun trabaja *Fortress* (Fortress, 2012). X10 se basa en Java, mientras que Chapel y Fortress han creado su propio lenguaje de programación basado en objetos.

Como podemos ver, la tendencia es reducir el esfuerzo que conlleva programar en arquitecturas paralelas. Sin embargo, por cada nivel de abstracción que se sube, conlleva un coste en cuanto a rendimiento y eficiencia. Es por eso que se buscan realizaciones que minimicen dichas pérdidas.

4. APLIACIONES DE LA PROGRAMACIÓN PARALELA

El mundo es masivamente paralelo. Ocurren multitud de cosas de forma simultánea. Por tanto, si queremos simular la realidad, debemos usar métodos que permitan hacer multitud de cosas de forma concurrente. Por ello, en el mundo de la computación, es necesario acudir al paralelismo. Como venimos justificando, la computación paralela se usa para resolver problemas en menor tiempo. Ya sea por complejidad de operaciones a realizar o necesidades de memoria para almacenar los datps. A continuación se recopilan algunas de las posibles aplicaciones.

Por ello, los usos de éste tipo de computación son múltiples. Se puede usar en ámbitos muy dispares, desde la biomedicina hasta la simulación de modelos económicos. Los problemas que se abordan suelen tener en común que son operaciones complicadas, operaciones no tan complicadas pero con un gran volumen de datos o bien, problemas que requieren de un mejor tiempo de respuesta.

En el entorno de las matemáticas, uno de los usos más populares es el álgebra lineal. Un problema popular a resolver es la multiplicación de matrices. En éste caso, podemos llegar a tener un volumen de datos muy elevado (órdenes de matrices muy elevados). Sobre éste volumen de datos se aplican operaciones como la factorización de matrices y cálculos de valores propios, entre otras. Mediante el álgebra lineal numérica, se resuelven los algoritmos que realizan cálculos del álgebra lineal.

Las ciencias de la salud también se han beneficiado enormemente del auge de la computación paralela. El diseño de nuevos fármacos, la mejora en tiempo y calidad de las imágenes médicas usadas para la diagnosis de enfermedades y los notables avances de la biomedicina son algunas de las aplicaciones donde se puede usar. Sin duda, uno de los resultados más importantes de la medicina moderna ha sido la decodificación del genoma humano, donde también ha tenido un papel importante.

Además de los restringidos a la ciencia pura, la computación paralela puede ser usada en muchos otros campos. Los dedicados al ocio están abarcando una parte importante del mercado. Más aún con la ya comentada programación de propósito general en procesadores gráficos. La mejora en gráficos de videojuegos y la renderización son algunos ejemplos.

Para el modelado de circuitos eléctricos, nuevos componentes electrónicos, fenómenos atmosféricos, físicos y moleculares hasta modelos económicos, se puede aprovechar éste tipo de computación. Además

de aplicaciones como defensa y seguridad, procesamiento digital de señales o ingeniería aeroespacial entre otros.

Otro campo que está adquiriendo gran importancia es la computación distribuida. Es decir, computación en procesadores geográficamente distribuidos. Íntimamente relacionado a éste concepto está la *computación en GRID*.

Un nuevo campo de investigación y desarrollo se abre con la aparición de la computación en la nube (C*loud Computing*), la computación cuántica y la computación biológica. Campos que están surgiendo ahora.

Finalmente, los centros de supercomputación y computación de altas prestaciones también hacen uso de computación paralela. Supercomputadores hay alrededor de todo el mundo (top500, 2012). En España existe la Red Española de Supercomputación.

5. CONCLUSIONES

En éste artículo hemos descrito cómo la innovación influye de manera notable sobre el desarrollo de regiones. Mostramos la computación paralela como una forma de innovación, no sin antes explicar los factores limitantes de la computación secuencial que propiciaron su desarrollo. Explicamos la teoría asociada a ella para desembocar en la explicación de modelos existentes para tratarla. Finalmente, damos una descripción cualitativa de las realizaciones de la misma y en qué campos se utiliza.

Nos hemos cerciorado de que sus usos son múltiples. En sus inicios, parecía destinada a círculos cerrados de la ciencia pura. Sin embargo, con el gran avance tecnológico de las últimas décadas, se ha convertido en un asunto de actualidad.

Una parte importante del desarrollo de la computación paralela se está enfocando en reducir la dificultad de programar en arquitecturas paralelas. Por ello, es un interesante campo donde seguir investigando e innovando. Además, la computación cuántica, biológica y en la nube, son campos donde queda mucho por hacer.

6. REFERENCIAS

Romer, P. (1990): “Endogenous technological change”. En revista *Journal of Political Economy*, Vol. 98, N. 5, octubre 1990, p. S71-S102.

Lucas, R. E. (1988): “On the mechanics of economic development”. En revista *Journal of Economic Monetary*, N. 22, febrero 1988, p. 3-42.

Schumpeter, J. A. (1939): “Business cycles: A theorical, historical and statistical análisis of the capitalista process”. Editorial McGraw-Hill, Nueva York.

OCDE y Eurostat (2006): “Manual de Oslo: guía para la recogida e interpretación de datos sobre innovación”. Editorial Grupo Tragsa, Madrid.

Blaise, L. (2012): “Introduction to parallel computing”. Disponible en: https://computing.llnl.gov/tutorials/parallel_comp/. Consultado en 09/02/2013 a 12:20.

Neumann, J. (1945): “First draft of a reporto m the EDVAC”. Disponible en http://virtualtravelog.net.s115267.gridserver.com/wp/wp-content/media/2003-08-TheFirstDraft.pdf. Consultado en 10/02/2013 a 14:26.

Godfrey, M. D., Hendry, D. F. (1993): “The computer as von Neumann planned it”. En revista *IEE Annals of the History of Computing*, Vol. 15, N. 1, junio 1993, p. 11-21.

Ujaldón, M. (2001): “Arquitectura del PC”. Editorial Ciencia-3, Madrid.

Moore, G. (1965): “Cramming more components onto integrated circuits”. En revista *Electronics*, Vol. 38, N. 8, abril 1965, p. 114-117.

Moore, G. (1975): “Excerots from a conversation with Gordon Moore: Moore’s Law”. Disponible en http://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1975_Speech.pdf. Consultado en 10/02/2013 a 16:17.

Ujaldón, M. (2003): “Arquitectura del PC. Volumen V: los chips”. Editorial Ciencia-3, Madrid.

Flynn, M. (1972): “Some computer organizations and their effectiveness”. En revista *IEEE Transactions on Computers*, Vol. c-21, N. 9, septiembre 1972, p. 948-960.

Asenjo, R. (2012): “Main Benefits of Task Parallel Frameworks”. Disponible en http://www.hpcadvisorycouncil.com/events/2012/Spain-Workshop/pres/16_UofMalaga.pdf. Consultado en 19/02/2013 a 15.50.

Meuer, H.; Strohmaier, E.; Dongarra, J.; Simon, H. (1993): “Top 500 supercomputers sites”. Disponible en http://top500.org/. Consultado en 10/02/2013 a 17:14.

OpenMP (2012): “The OpenMP API specification for parallel programming”. Disponible en http://openmp.org/wp/. Consultado en 23/12/2012 a 10:21.

Intel Threading Building Blocks (2012): “Intel Threading Building Blocks (Intel TBB)”. Disponible en http://threadingbuildingblocks.org/. Consultado en 23/12/2012 a 11:40.

Message Passing Interface (2012): “The Message Passing Interface (MPI) standard”. Disponible en http://www.mcs.anl.gov/research/projects/mpi/. Consultado en 23/12/2012 a 12:43.

Global-Address Space Networking (GASNet) (2012): “GASNet Communications System”. Disponible en http://gasnet.cs.berkeley.edu/. Consultado en 26/12/2012 a 11:34.

nVIDIA (2013): “NVIDIA”. Disponible en http://www.nvidia.es/page/home.html. Consultado en 12/02/2013 a 11:30.

OpenCL (2012): “OpenCL – The open standard for parallel programming of heterogeneus systems”. Disponible en http://www.khronos.org/opencl/. Consultado en 12/02/2013 a 12:02.

UPC (2013): “Berkeley Unified Parallel C (UPC)”. Disponible en http://upc.lbl.gov/. Consultado 11/02/2013 a 10:41.

CAF (2012): “Co-Array Fortran”. Disponible en http://www.co-array.org/. Consultado en 29/12/2012 a 16:14.

Titanium (2012): “Titanium Project home page”. Disponible en http://titanium.cs.berkeley.edu/. Consultado en 29/12/2012 a 17:40.

HPCS Program (2002): “High Producivity Computing Systems (HPCS)”. Disponible en http://www.darpa.mil/Our_Work/MTO/Programs/High_Productivity_Computing_Systems_(HPCS).aspx. Consultado en 20/01/2013 a 19:31.

Chapel (2002): “The Chapel Pararallel Programming Language”. Disponible en http://chapel.cray.com/. Consultado en 12/02/2013 a 12:18.

X10 (2002): “X10: performance and productivity at scale”. Disponible en http://x10-lang.org/. Consultado en 12/02/2013 a 13:17.

Fortress (2002): “Fortress”. Disponible en http://labs.oracle.com/projects/plrg/Fortress/overview.html. Consultado en 11/02/2013 a 14:31.