INSTITUTO POLITECNICO NACIONAL

CENTRO DE INVESTIGACION EN COMPUTACION

Planificación de Tareas de Tiempo Real con Restricciones de Precedencia y Potencia

T E S I S

QUE PARA OBTENER EL GRADO DE DOCTOR EN CIENCIAS DE LA COMPUTACION

P R E S E N T A

M. en C. Héctor Eduardo Silva López

DIRECTOR
DR. SERGIO SUAREZ GUERRA

MEXICO, D.F. **Diciembre de 2009**

AGRADECIMIENTOS

Al Instituto Politécnico Nacional por su apoyo económico a través del programa de becas PIFI e Institucional.

Un agradecimiento especial a mí asesor, el Dr. Sergio Suarez Guerra, por el tiempo y esfuerzo dedicado durante todo este tiempo como director de esta tesis doctoral. Sin duda alguna, sus consejos y motivación han sido determinantes para el buen fin del mismo.

A los doctores del Centro de Investigación en Computación por compartir su entusiasmo por el conocimiento y ser participes de mi vida académica, en especial a los doctores: Dr. Sergio Suarez y el Dr. Juan Carlos Chimal.

A los miembros del jurado integrado por los siguientes: Dr. Luis Pastor Sánchez Fernández, Dr. Carlos Aguilar Ibáñez, Dr. Sergio Suárez Guerra, Dr. Pedro Guevara López, Dr. Marcelino Becerril Silva y el Dr. José de Jesús Medel Juárez, por sus incontables sugerencias y sobre todo por el tiempo dedicado en llevar al termino la presente tesis.

A todas las personas que integran la UTE, en especial a la Sra. Silvia Arteaga por todos los tramites que tuvo que realizar para que fuera acreedor a la beca PIFI e Institucional. A la Sra. Carmen Pitayo por todos los trámites administrativos que realizo y finalmente a la Maestra Martha Duran por todo el apoyo recibido desde mi ingreso hasta el día de mi graduación.

A mi querida esposa María del Pilar López Cárdenas, por haber iniciado esta aventura junto a mí, por su paciencia, por su comprensión y sobre todo por estar ahí cuando la necesitaba.

A mis hijos Abril y Alan por pasar los fines de semana sin salir de casa sin ningún reclamo y apoyándome en todo momento.

INDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

El problema que se presenta en esta tesis es la planificación de tareas de tiempo real con restricciones de precedencia y potencia con el objetivo de obtener el máximo ahorro de energía del conjunto de tareas y las sumas de las utilizaciones serán menores o iguales a la capacidad total del sistema. Para resolver esta cuestión se presenta el mismo como un problema de control óptimo en tiempo discreto, la metodología y dos algoritmos para resolverlo son implementados: el primero se resuelve como un problema de cálculo de variaciones y el segundo utiliza el método de la programación dinámica. Posteriormente se implementó un algoritmo específico para un problema en donde se maximiza la función objetivo propuesta.

El resultado para el primer algoritmo es el (PD-N) que tiene un buen desempeño al alcanzar un ahorro de energía muy cercana a la holgura en el sistema, para un número de nodos que van de 6 a 16 su comportamiento es muy parecido y muy cercano al óptimo, por último el tiempo de ejecución tiende a decrecer en cuanto la carga del procesador aumenta, pero una carga entre 0.2% y 0.5% vemos que el método Max emplea más tiempo de ejecución y el método Min el menor tiempo. El segundo algoritmo (KP-B) que se presenta como un problema de programación lineal con restricciones discretas, en donde la solución está basada en la ecuación de Bellman. Estos dos algoritmos se implementaron y se compararon con el más cercano de la literatura. Para el algoritmo KP-B el ahorro de energía, cuando se varía el número de tareas, es muy cercano al Opt-Lu. Cuando se varía la carga del procesador tiene un mejor ahorro de energía que el Opt-Lu a partir del 60% de carga del procesador. Con respecto al tiempo de ejecución el KP-B es el que tiene menor tiempo, estando dentro de los 13 y 175 milisegundos.

Y finalmente, el método y algoritmo específico que se implementó en una imagen obteniendo un ahorro de energía del 44.19%. En seguida se obtuvo el ahorro de energía para diferentes tamaños de imágenes, en donde los tamaños pequeños son los que presentan un ahorro de energía mayor y para un tamaño mayor se aprecia un ahorro de energía menor. Y finalmente se ejecutó el algoritmo en una ambiente de memoria compartida y se obtuvo un ahorro de energía promedio del orden de 28.77%.

ABSTRACT

The problem presented in this thesis is real-time scheduling with precedence and power constraints in order that the energy saving of the system is maximized without having the utilization sum exceed the capacity of the system. To resolve this issue a discrete time optimal control problem is presented. A methodology and two algorithms for solving it are implemented: the first is solved as a calculus of variations problem and the second using the dynamic programming method. A specific algorithm was implemented where the proposed objective function is maximized.

The result of the first algorithm (PD-N) produced a good performance by achieving energy saving very close tolerance level in the system for a number of nodes ranging from 6 to 16; their behavior is very similar and very close to optimal. Finally the execution time tends to decrease as processor load increases, but a charge between 0.2% and 0.5% showed that the Max method uses more runtime and the Min method the shortest time. The second algorithm (KP-B) is presented as a discrete constraints linear programming problem, where the solution is based on the Bellman equation. The algorithms presented in this work were compared to those proposed in literature for solving this kind of problem. When the number of the tasks varied for the KP-B algorithm, the execution time was very close to that of Opt-Lu, and when the processor load varied, from 60% of processor load and upwards it had better energy saving that Opt-Lu. Regarding the execution time, the KP-B had the shortest, being between 13 and 175 milliseconds.

Finally, for the specific method an energy saving of the 44.19% is obtained for an image. Then energy saving for different image sizes was obtained, thus concluding that a grater energy saving occurs for small image sizes, whereas for bigger size, less energy is saved. Later, when executing the algorithm in a shared memory environment, an extra saving to the order of 28.77% on average was obtained.

CAPITULO 1

INTRODUCCION

En este capítulo como antecedentes, se presentan los conceptos básicos de un sistema de tiempo real, se explica el concepto de factor de utilización, así como se presentan los principales algoritmos de planificación. El modelo del sistema es en seguida mostrado, después lo que motivó el desarrollo de la presente tesis, luego se plantea el problema a resolver y por último los objetivos a desarrollar en este trabajo de tesis y su justificación, así como las limitaciones de la tesis y las contribuciones del trabajo.

El *tiempo de respuesta* de un sistema de software es el que transcurre entre que se ponen un conjunto de entradas al sistema y que aparecen todas las salidas asociadas del mismo.

Tradicionalmente se ha considerado que un sistema computacional funciona correctamente cuando la solución obtenida es correcta desde el punto de vista lógico. Esta definición, si bien es válida para sistemas de cálculo convencional, no es suficiente cuando estamos considerando sistemas que interactúan fuertemente con el entorno. Este tipo de sistemas, normalmente dedicados a tareas de monitoreo o control, suelen estar formados por un conjunto de recursos tales como sensores, unidades de cómputo y actuadores que deben trabajar de forma coordinada para realizar la labor encomendada. La cooperación entre diferentes recursos obliga, no sólo a que cada operación sea correcta, sino a que se realice en los instantes oportunos. Este tipo de sistemas a los que es preciso imponer restricciones temporales se les denominan sistemas de tiempo real [44, 61].

Existen las siguientes definiciones de un sistema de tiempo-real (STR).

- Laplante: Un *sistema de tiempo real* es un sistema cuyos tiempos de respuesta deben satisfacer requisitos explícitos, y que en caso de no responder dentro de acuerdo a los requisitos pueden producirse consecuencias graves, incluyendo el fracaso del sistema.
  - Un STR *fracasa* si no puede satisfacer uno o más de los requisitos establecidos en la especificación del sistema.
- Burns:

  Un *sistema de tiempo real* es un sistema de procesamiento de información que debe responder a estímulos generados externamente dentro de un período finito y especificado.
  - la correctitud depende no sólo del resultado lógico sino también del instante en que el sistema lo produce
  - el no respeto de los requisitos temporales es tan malo como una respuesta incorrecta.

Las tareas de tiempo real se pueden clasificar en base a las consecuencias provocadas por la pérdida de los plazos de respuesta en los siguientes tipos:

1. *Sistemas de Tiempo Real duros.* Sistemas en los que es absolutamente imperativo que las respuestas ocurran dentro de los períodos de tiempo especificados, a riesgo de causar un desastre. (Ej: Sistema de Control de Vuelo).
2. *Sistemas de Tiempo Real Suaves*. Sistemas en los cuales los requerimientos temporales son importantes, pero que igualmente funcionan correctamente si ocasionalmente los mismos no se respetan: el desempeño es degradado pero no se destruyen por el hecho de fracasar. (Ej: Sistema de Adquisición de Datos).

3. *Sistemas de Tiempo Real Firmes*. Son sistemas de tiempo real suaves, pero en caso de no respetar las restricciones de tiempo no sirve de nada el servicio que prestan.

Una Tarea de Tiempo Real es una entidad ejecutable J_i que al menos es caracterizada por un tiempo de arribo y una restricción temporal. Está formada por un conjunto de instancias J_k, tal que $J_i = \{j_k\}$ con $i,k \in Z^+$, [19].

Considerando la regularidad en la ejecución de las tareas de tiempo real, se pueden clasificar como:

1. *Tarea Periódica.-* Reiniciación periódica de tareas, cada instancia debe completar antes de su plazo.
2. *Tarea aperiódica.-* Se activan al producirse determinados eventos de forma imprevisible. Se ejecutan sólo durante una instancia de ejecución al término de la cual desaparecen. Las tareas tienen plazos suaves o no tienen plazos.
3. *Tarea esporádica.-* Son tareas aperiódicas con restricciones temporales críticas (o duras). Si se monitorea el arribo, es posible determinar una separación mínima entre activaciones consecutivas, lo cual podría permitir caracterizarlas como tareas periódicas.

Los sistemas de tiempo real son concurrentes por naturaleza. Un sistema concurrente se compone de un conjunto de procesos o *tareas* que se ejecutan de forma aparentemente paralela. Un sistema de tareas concurrentes en un solo procesador se puede ejecutar multiplexando el tiempo de varias formas:

- Ejecución síncrona: ejecutivo cíclico.
- Ejecución asíncrona:
    - Núcleo de multiprogramación separado (SO de tiempo real).

- Lenguaje de programación concurrente.

Cuando se usa un lenguaje concurrente, el multiplexado del procesador se realiza por medio de un núcleo de ejecución *(run-time system)*.

Las características principales de los sistemas de tiempo real son:

- Oportunidad (timeliness): un STR debe operar de forma tal que satisfaga determinados requisitos de tiempo, que pueden ser de dos tipos:
  - Requisitos de tiempo relativos: si una acción debe ocurrir dentro de un intervalo de tiempo dado relativo a la ocurrencia de un evento
  - Requisitos de tiempo absolutos: si una acción debe ocurrir en un punto fijo de tiempo.
- Simultaneidad: un STR debe responder dentro de intervalos de tiempo predeterminado a los diferentes estímulos (eventualmente simultáneos) que reciba desde el exterior. Para satisfacer este requisito, debe operar en forma paralela (ya sea a través de procesadores múltiples o cuasi-paralelismo).
- Predictibilidad: un STR debe comportarse dentro de los límites que se derivan de su especificación; los resultados que produce un STR frente a determinados estímulos deben ser completamente predecibles.
- Confiabilidad (*dependability*), que implica:
  - correctitud: el sistema se comporta de acuerdo a su especificación.
  - robustez: el sistema permanece en un estado predecible, aún cuando el entorno no corresponda a las especificaciones.
  - disponibilidad (*readiness*): los procesos de un STR consisten en un lazo infinito, a los efectos de interactuar continuamente con el entorno.

El estado de un proceso así como las restricciones de los sistemas de tiempo real, sé pueden consultar en el apéndice A.

1.1 Factor de utilización.

Se considera un conjunto $T = \{T_1,...,T_n\}$ de n tareas periódicas y desalojables de tiempo real, corriendo en un procesador. Cada tarea T_i está compuesta de restricciones de precedencia igual a un grafo. El periodo de T_i es representado por P_i (entero no negativo), el cual es igual al plazo de respuesta relativo de la tarea. C_i es un entero y representa el tiempo de ejecución requerido por T_i. El factor de utilización del procesador U es la fracción de tiempo consumido por el procesador en la ejecución del conjunto de tareas. Dado que C_i/P_i es la fracción de tiempo que consume el procesador en ejecutar la tarea T_i, el factor de utilización para n tareas está dado por:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \tag{1}$$

El factor de utilización provee una medición de la carga de trabajo ejecutándose en la CPU (Unidad Central de Procesamiento).

1.2 Planificación de tareas de tiempo real.

Para maximizar el número de tareas de tiempo-real que pueden ser procesadas sin violaciones, los sistemas computacionales de tiempo real usan sofisticados algoritmos de planificación para decidir el orden en el cual las tareas son ejecutadas. El desempeño de un algoritmo de planificación es medido por su habilidad para generar una planificación factible para un conjunto de tareas de tiempo real. Un planificador para asignar tareas a uno o más procesadores se dice que es factible si la ejecución de cada tarea puede ser completada antes del plazo de respuesta. Un planificador factible se dice ser mínimo si no hay una planificación factible utilizando menos procesadores. Un algoritmo de planificación

se dice ser óptimo, si para un conjunto de tareas, el algoritmo encuentra una planificación mínima.

Los algoritmos de planificación se clasifican en ocho clases principales y estos son:

- Desalojo: La tarea que está en ejecución puede ser interrumpida en cualquier momento para asignar al procesador otra tarea, siguiendo una política de planificación previamente establecida.
- Sin desalojo: Una vez asignada la tarea al procesador, ésta permanece en ejecución hasta que termina. Las decisiones de planificación son tomadas cuando la tarea termina.
- Estática: Las decisiones de planificación son tomadas antes que comiencen la activación de las tareas.
- Dinámica: Las decisiones de planificación son tomadas cuando las tareas se están ejecutando en forma dinámica.
- Fuera de línea: Las decisiones de planificación son tomadas antes de que se ejecuten las tareas. Estas decisiones son almacenadas en una tabla y después es ejecutada por un despachador.
- En línea: Las decisiones de planificación son tomadas en tiempo de ejecución cada vez que una tarea comience o termine su ejecución.
- Optima: Se dice óptimo si se minimiza alguna de las funciones de costo definidas sobre el conjunto de tareas. Cuando no es definida una función de costo, y lo único es alcanzar una planificación factible, entonces el algoritmo de planificación es óptimo si éste encuentra una solución de planificación que no pueda ser contradecida por ningún otro algoritmo de planificación. La solución indica si el conjunto de tareas es factible o no es factible (si cumple o no con sus plazos de respuesta).
- Heurístico: Encuentra una solución aproximada que intenta estar cerca de la solución óptima.

Los algoritmos de planificación pueden ser divididos dentro de: algoritmos de prioridad fija y algoritmos de prioridad dinámica. En los algoritmos de prioridad fija, la prioridad de una tarea permanece constante todo el tiempo, por otro lado en un algoritmo de prioridad dinámica, la prioridad de una tarea puede cambiar durante su ejecución. Estos algoritmos se pueden apreciar en la figura 1.1.

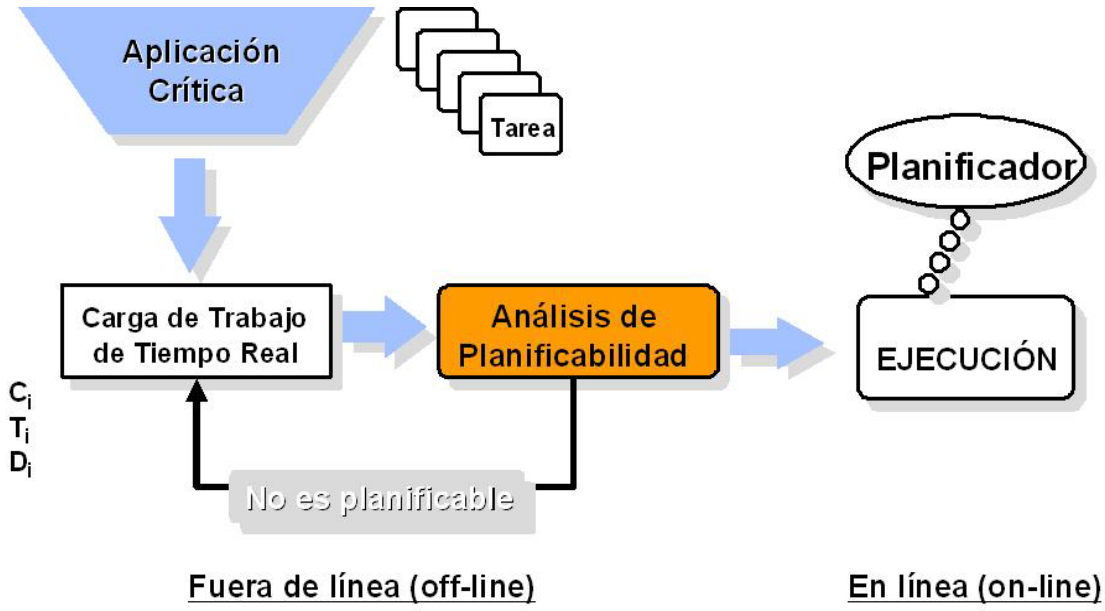


Figura 1.1. Proceso de planificación.

1.3 Algoritmos de planificación.

Monoprocesadores

Liu y Layland dedujeron la primera prueba de planificabilidad dentro del algoritmo RMA [18]. Esta prueba se aplica al caso de que las n tareas sean independientes en su ejecución, con plazos de finalización iguales a sus períodos. Los autores probaron que la asignación de prioridades óptima para este tipo de sistemas era el Rate Monotonic Scheduling (RMS), que asignan mayor prioridad al que tiene menor periodo de activación. Esta asignación es óptima en el sentido de que si un sistema con esta asignación no es planificable, entonces no hay ninguna otra

asignación de prioridades que haga que el sistema lo sea. En estas condiciones, el sistema será planificable si cumple:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq U(n) = n\left(2^{\frac{1}{n}} - 1\right) \tag{2}$$

Donde:

U - La utilización total del procesador.

$U(n)$ - El límite de utilización máximo del procesador para n tareas.

Esta prueba condiciona la planificabilidad del sistema a que la utilización total del procesador no sobrepase un valor máximo establecido, dependiendo del número total de tareas. Este límite de utilización varia (para n>1) entre el 83% y el 69%, lo cual apunta a sistemas planificables para utilizaciones relativamente bajas. Esta prueba es suficiente pero no necesaria, de forma que puede haber conjuntos de tareas que sobrepasen el límite de utilización establecido por la prueba y que, sin embargo, sean planificables.

Lehoczky, Sha y Ding [39] propusieron una prueba exacta para este mismo caso. Para ello, valiéndose de la definición de instante critico, definen una utilización del peor caso para una tarea T_i, en un intervalo de anchura t. Esta utilización del peor caso se obtiene como la cantidad total de tiempo de ejecución requerida por tareas de prioridad mayor o igual que la de T_i, dividido entre la anchura t, es decir:

$$U_i(t) = \frac{\sum_{j=1}^{i} \left(\left\lceil \frac{t}{T_j} \right\rceil C_j \right)}{t} \tag{3}$$

Donde:

$U_i(t)$ - La utilización del peor caso en el intervalo $(0,t)$.

$\lceil x \rceil$ - Función techo, definida como el menor entero mayor o igual que x.

Entonces, el sistema será planificable si se cumple:

$$\min_{0<t<T_i} U_i(t) \leq 1 \quad \forall i = 1...n \tag{4}$$

Cuando los plazos de finalización son menores que los períodos, las pruebas anteriores ya no son válidas. Además, la asignación de prioridades en función de los periodos ya no es óptima: Leung y Whitehead probaron que, en este caso, la asignación óptima es la basada en los plazos de ejecución (deadline monotonic, DM) [37], de forma que le corresponde la mayor prioridad a la tarea que tenga menor plazo de ejecución.

Audsley [4] desarrolló una prueba de planificabilidad suficiente, basado también en utilizaciones, para un conjunto de tareas con asignación de prioridades según sus plazos de ejecución. La mayor interferencia en la ejecución de una tarea T_i se produce en un instante crítico. Para una tarea que cumpla su plazo, esta interferencia está limitada por la expresión:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \tag{5}$$

En donde:

$hp(i)$.- Es el conjunto de tareas que pueden interferir la ejecución de T_i, formado por las tareas con prioridad mayor o igual que la de T_i, (exceptuando la propia T_i).

El sistema será planificable si cada tarea experimenta una utilización máxima, definida dentro de su plazo, menor del 100%. Esto es:

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} \leq 1 \quad \forall i = 1...n \tag{6}$$

Esta prueba no es exacta, ya que supone que las tareas de mayor prioridad del conjunto $hp(i)$ pueden interrumpir la ejecución de T_i en cualquier instante dentro

del plazo, sin tener en cuenta que la ejecución de T_i podría haber finalizado antes de que esa expulsión se hubiera producido.

Todas las pruebas que se han presentado se basan en la utilización del procesador y nos permiten estudiar la planificabilidad de sistemas con asignación de prioridad Rate Monotonic y Deadline Monotonic. Un método más potente, en el sentido de que permite estudiar conjuntos de tareas con plazos menores o iguales que el sentido de que permite estudiar conjuntos de tareas con plazos menores o iguales que el periodo y cualquier asignación arbitraria de prioridades, es el desarrollado por Joseph y Pandya [34]. Este método es equivalente al algoritmo de Dilatación Temporal (Time Dilation Algorithm) desarrollado por Harter [23] y que utiliza lógica temporal para obtener tiempos de respuesta. El método de Joseph y Pandya se basa en el cálculo de los tiempos de respuesta peor caso de cada tarea. Si Ri es el tiempo de respuesta del peor caso de una tarea T_i, la máxima interferencia producida a partir del instante crítico, debida a tareas de mayor prioridad viene dada por:

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{7}$$

El tiempo de respuesta del peor caso Ri vendrá dado por la suma de esta interferencia más el tiempo de ejecución de la propia tarea, esto es:

$$R_i = C_i + I_i \tag{8}$$

De forma que la planificación del sistema se condiciona a que se cumpla

$$R_i \leq D_i, \quad \forall i = 1...n \tag{9}$$

En sistemas de tiempo real estricto si es imprescindible que nunca se pierdan los plazos de ejecución; esto quiere decir, que una prueba válida para el análisis de planificabilidad debe ser siempre suficiente. Por otra parte, una prueba que no sea exacta será aceptable cuanto más se acerque para cumplir con los plazos de respuesta.

El algoritmo EDF (Earliest Deadline First) propuesto por [18, 44], asigna las prioridades a las tareas dependiendo de la cercanía del plazo de respuesta; la tarea con el plazo más cercano recibe la prioridad más alta. En cualquier instante, la tarea con más alta prioridad puede desalojar a la tarea que en ese momento utilice el procesador. La condición necesaria y suficiente para planificar un conjunto de tareas bajo EDF es:

$$\sum_{i=1}^{n} (C_i / P_i) \leq 1 \qquad (10)$$

Donde n es el número total de tareas.

El algoritmo EDF es óptimo en el sentido de que si un conjunto de tareas es factiblemente planificado por cualquier algoritmo de planificación de prioridades dinámicas, entonces también es planificado bajo EDF. Algoritmos con manejo de prioridad dinámica tienen cierta ventaja sobre los algoritmos de prioridad fija. Una ventaja de este algoritmo, radica en el hecho de que la cota límite es (máxima) 100% para cualquier conjunto de tareas. Esto significa que el CPU puede ser utilizado totalmente. La principal desventaja de EDF se encuentra en que no existe una manera de identificar que tareas perderán sus plazos de respuesta ante un evento de sobrecarga $(\sum_{i=1}^{n} (C_i / P_i > 1)$ ocurre.

Multiprocesadores.

En el apéndice B se presenta el estado en los sistemas multiprocesadores.

1.4 Modelo del sistema.

Generalmente un problema de planificación se define por cuatro parámetros: a) El ambiente del sistema, b) La caracterización de las tareas, c) El ambiente de planificación, y d) Los objetivos de planificación. El ambiente del sistema

especifica los tipos de procesadores a utilizar. Se dice que los procesadores son homogéneos si la ejecución de una tarea en particular sobre cualquier procesador toma el mismo tiempo de cómputo (capacidad de procesamiento, acceso a memoria y acceso a dispositivos de entrada y salidas), además se considera que los procesadores son heterogéneos. La caracterización de las tareas permite especificar las restricciones de tiempos de las tareas, las relaciones y jerarquías entre ellas. El ambiente de planificación describe las restricciones impuestas sobre el algoritmo de planificación: si las tareas son desalojables o no; si la planificación se lleva a cabo en un esquema en línea o fuera de línea. Finalmente los objetivos nos definen las metas a alcanzar: reducir el número de procesadores, lograr que todas las tareas cumplan sus plazos, minimizar el tiempo de cómputo, etc. Considerando estos puntos, para nuestro estudio asumiremos las siguientes condiciones:

a) Para el ambiente del sistema, se consideran un procesador y el sistema operativo a utilizar (Linux).
b) Para la caracterización de las tareas T_i, se consideran a éstas con restricciones de precedencia y restricciones de potencia. Los parámetros a que definen a una tarea son, el tiempo de ejecución, C_i, el periodo P_i y el plazo de respuesta D_i. Se consideran sólo tareas periódicas, es decir, que su arribo se realiza a intervalos de tiempo de duración fija y asumiremos que el plazo de respuesta de una tarea corresponde a su siguiente tiempo de arribo (o periodo). El periodo y el tiempo de cómputo de la tarea T_i satisface que $T_i > 0$ y $0 \le C_i \le T_i = D_i, (i = 1,...,n)$. Así mismo se define $u_i = C_i / T_i$ como el factor de utilización de la i-esima tarea. El factor de utilización del conjunto de tareas es la suma de los factores de utilización de las tareas en el conjunto,

$$U_T = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{11}$$

c) Para el ambiente de planificación, se asume que las tareas pueden ser desalojadas en cualquier momento, es decir, en cierto instante el despachador determina que tarea va a ejecutar. El costo del desalojo y arribo de una tarea se considera nulo. Las tareas no requieren un acceso exclusivo a otro recurso más que al procesador. Se utiliza un esquema de asignación de prioridad de acuerdo a la función de objetivo y a las restricciones impuestas y se lleva a cabo siguiendo un orden de prioridad decreciente. Esto es, siempre se ejecutará primero la tarea con la más alta prioridad de todas las que han arribado. Otro parámetro en el ambiente de planificación, es aquel relacionado con las características del conjunto completo de tareas. En este caso, consideramos a la planificación fuera de línea (off-line) así como a la planificación en línea (on-line).

d) Los objetivos de planificación son I) Que todos los plazos de respuesta de las tareas se cumplan, II) Minimizar el consumo de energía o maximizar el ahorro de energía de acuerdo a las restricciones de precedencia y restricciones de potencia del conjunto de tareas de tiempo real.

1.5 Motivación.

Con el advenimiento de los sistemas de cómputo portátiles y de tamaño pequeño, incluyendo la nueva tendencia de los sistemas en un solo chip, el consumo de potencia ha emergido recientemente como el punto principal en muchos proyectos de investigación y de los sistemas comerciales. Estos proyectos típicamente direccionan su objetivo en minimizar el consumo de potencia en una plataforma dada, la cual puede ser transportada con la ayuda de una batería de larga duración. Actualmente los sistemas con una potencia eficiente manejan funciones que pueden ser invocadas para apagar algunos componentes del sistema o

pueden escoger entre diferentes estados para cada componente. El Manejo de Potencia Avanzado (Advance Power Management, APM) y la Interfaz de Potencia y la Configuración Avanzada (Advanced Configuration & Power Interface, ACPI) son interfaces estándar que proveen las aplicaciones con alguna funcionalidad para el manejo de potencia. Recientemente la investigación ha demostrado que más allá de la reducción en el consumo de potencia, ésto puede ser acompañado por un escalamiento dinámico del voltaje/frecuencia, éstas pueden ser técnicas de un compilador y poder seleccionar el uso alternativo de algoritmos

1.6 Planteamiento del problema.

Se considera un sistema de tiempo real en el cual todas las tareas son periódicas. Cada instancia se le denominará como sub-tarea y existirán un número finito de instancias, las cuales deberán ser ejecutadas antes del siguiente arribo de la próxima tarea. En su estado inicial todas las tareas estarán a su máximo nivel de velocidad. El conjunto de tareas estará representado por un Grafo acíclico dirigido (DAG en inglés) así que cada vez que arribe o salga un DAG del sistema, el problema es determinar el modo (velocidad del procesador) de ejecución de las tareas (o nodos en el grafo), tal que no se pierdan sus plazos de respuesta y el ahorro de energía del sistema sea maximizado. Todas las tareas tendrán un número finito de modos de ejecución, donde se escogerá un modo para su ejecución. Notar que la solución a este problema deberá ser calculado cada vez que una tarea arribe o salga del sistema, además una solución con una alta complejidad computacional puede ocasionar probablemente que se pierdan los plazos de respuesta. Esto significa que la solución requiere un algoritmo con un resultado eficiente de bajo costo computacional. Las soluciones heurísticas se caracterizan por tener esta propiedad.

1.7 Hipótesis.

En un STR las tareas que se ejecutan en cada hiperperíodo pueden variar en cantidad y por tanto en sus características, lo que obliga a re-planificar el sistema cada vez que hay cambio de tareas. Existe la posibilidad de realizar una planificación en función del hiperperíodo total y adecuar las velocidades de ejecución de las tareas, para lograr un ahorro general de energía en el consumo del sistema.

1.8 Objetivos.

El objetivo de investigación es encontrar una nueva solución al problema de la planificación de tareas de tiempo real con restricciones de precedencia y restricciones de potencia. Con la finalidad de poder minimizar el consumo de potencia conservando las restricciones temporales de las tareas, es decir, que las tareas se ejecuten dentro de sus plazos de respuesta.

1.8.1 Objetivo general.

Planificar un conjunto de tareas de tiempo real concurrentes con restricciones de precedencia y potencia con la finalidad de maximizar el ahorro de energía.

Se utilizará, la metodología de optimización de sistemas dinámicos, es decir, la optimización de sistemas que evolucionan en el tiempo, se proponen algoritmos para su implementación.

1.8.2 Objetivos específicos.

- Realizar la modelación matemática a través de una estructura discreta o combinatoria.

- Calcular la complejidad computacional del problema matemático surgido de la modelación.
- Presentar el algoritmo que se usará para encontrar la solución al problema en cuestión.
- Proponer una solución al presentarlo como un problema de cálculo de variaciones el cual es resuelto como un problema de control optimo en tiempo discreto.
- Resolver el problema planteado como un problema de control óptimo en tiempo discreto, utilizando el método de la programación dinámica
- Implementar un algoritmo específico para un problema específico en donde se maximicé la función objetivo propuesta.

1.9 Justificación.

En la figura 1.2 se aprecia cómo se encuentra el estado del arte para los sistemas de tiempo real utilizando tareas periódicas concurrentes y ejecutándose en un ambiente con un solo procesador. El considerar tareas periódicas de tiempo real con restricciones de precedencia y potencia no ha sido considerado y nadie hasta el momento ha presentado algo en esta área. Por esta razón se está proponiendo precisamente esta situación y es la de presentar una metodología con su algoritmo de optimización para tratar a las tareas periódicas con restricciones de precedencia y restricciones de potencia utilizando un procesador, con el objetivo de maximizar el ahorro de energía de los sistemas de tiempo real.

Figura 1.2. Justificación.

1.10 Limitaciones y alcances.

Las limitaciones y alcances de la presente tesis, son las siguientes:

1. La implementación de los algoritmos propuestos en esta tesis son únicamente probados en un ambiente de simulación. En una Laptop convencional y utilizando alguna versión libre del sistema operativo Linux.
2. La simulación consiste en igualar un ambiente de un sistema operativo de tiempo real, para contar con todas los atributos de las tareas de tiempo real.
3. La simulación también cuenta con todas las características de un procesador de velocidad variable, con cambios discretos de velocidad.

4. Los algoritmos propuestos por otros autores en la literatura, son adaptados al sistema de simulación desarrollado.
5. Los resultados obtenidos en la simulación son graficados utilizando un software comercial (Origin versión 7.5).
6. Las ecuaciones utilizadas para obtener el porcentaje del ahorro de energía, así como para el consumo de potencia son las presentadas por los fabricantes de procesadores de velocidad variable y en ningún momento se modifican o se desarrollan nuevas ecuaciones.
7. Para calcular la utilización de cada tarea a cada velocidad, se usa la que se utiliza en la literatura referida en su momento. Y no se propone una nueva, ni se modifica la existente.
8. La nomenclatura ya definida en la literatura para los sistemas de tiempo real se mantiene a menos que explícitamente se mencione en la tesis.

1.11 Contribuciones de este trabajo.

Se presentan las siguientes contribuciones para este trabajo de tesis:

a) Se diseña un modelo simple dirigido para el manejo de la potencia. Específicamente, son desarrollados dos algoritmos originales eficientes para el manejo de la frecuencia y/o voltaje, considerando que el sistema tiene restricciones de precedencia y potencia, es decir que pueden ser utilizados para cualquier sistema de tiempo real que tienen como limitante el suministro de la energía y sea ejecutado en una Laptop.
b) Los efectos producidos por otros efectos prácticos tales como niveles de frecuencia discretos y contención de acceso a memoria compartida son también tratados.
c) Se presenta el método implementado en un algoritmo, que brinda mejor desempeño que lo referenciado en la actualidad para sistemas con un

procesador, así como también puede ser extendido para múltiples procesadores.

RESUMEN

En este capítulo se describieron los conceptos básicos de los sistemas de tiempo real, describiendo el factor de utilización, así como se da la referencia para consultar las tres restricciones más importantes de un sistema de tiempo real y las definiciones de proceso. En seguida se mencionan los algoritmos de planificación para monoprocesadores y se da la referencia para consultar los algoritmos de planificación para multiprocesadores, luego el modelo del sistema que se utiliza en los próximos capítulos. También se planteó el problema a solucionar así como los objetivos de este trabajo de tesis. Posteriormente se presentó la justificación para realizar este trabajo y por último las limitaciones y alcances y las contribuciones que se obtienen al concluir esta tesis.

CAPITULO 2

ESTADO DEL ARTE

En este capítulo se muestra el estado del arte en el área de planificación de tareas de tiempo real con restricciones de precedencia y/o restricciones de potencia. Se empieza con una breve semblanza referente al ahorro de energía, presentando el consumo de potencia en los dispositivos que integran una PC portátil. Luego se detalla la tecnología de las baterías y su tecnología futura. La arquitectura para el ahorro de energía se presenta a continuación, así como la implementación para el ahorro de energía en un procesador Core 2 Duo. En seguida el consumo de energía en cuatro sistemas operativos comerciales y se continua con las técnicas y el manejo para ahorrar energía. Por último se explica brevemente los parámetros utilizados para la implementación de un procesador de velocidad variable.

Con lo anterior se propone un algoritmo que permita el planificar tareas de tiempo real con restricciones de precedencia y restricciones de potencia, tal que sea maximizado el ahorro del sistema, sin que las tareas pierdan sus plazos de respuesta. Para realizar esto se proponen dos diferentes condiciones de ejecución y una aplicación práctica, mismas que están definidas en el inciso 1.8.2 del capítulo1 y serán desarrolladas en capítulos subsecuentes.

2.1 Motivación.

Existen sistemas computacionales fuertemente relacionados con el entorno que les rodea y con el cual se encuentra en constante interacción. Ejemplos de este tipo de sistemas son los sistemas de adquisición de datos y los controladores industriales, los cuales están compuestos por diversos elementos (actuadores, sensores, unidades de cómputo, redes de interconexión, etc.) que deben trabajar

de forma conjunta y coordinada. Debido a la naturaleza cambiante del entorno con el que interactúan, junto con la necesidad de coordinación entre sus componentes, los resultados obtenidos sólo podrán ser considerados válidos cuando, además de ser correctos desde el punto de vista lógico, hayan sido generados a tiempo. Resultará, por tanto, preciso imponer restricciones temporales cuyo cumplimiento garantice el correcto funcionamiento del sistema. Este tipo de sistemas, capaces de realizar tareas y responder a eventos asíncronos externos dentro de unos plazos temporales determinados son los denominados “Sistemas de Tiempo Real” [61].

Para que sea posible la predecibilidad temporal del sistema completo, todas las partes que le componen deberán de presentar un comportamiento predecible. En consecuencia, en el caso de utilizar un sistema operativo, los servicios que éste proporcione a las aplicaciones deberán presentar tiempos de respuesta acotados, de forma que así sea capaz de garantizar los requerimientos temporales de los procesos bajo su control. Mientras que en un sistema operativo de tiempo compartido, como Unix [7][38], lo importante es proporcionar a los usuarios unos buenos tiempos de respuesta promedios [32], la clave en los sistemas operativos de tiempo real será garantizar los requerimientos temporales; el tiempo de respuesta promedio pasa así a un segundo plano.

Dentro de los sistemas de tiempo real, un tipo particular son los “sistemas empotrados”. En este tipo de sistemas, el computador constituye una parte más de un sistema mayor en el que se encuentra altamente integrado y en el que se dedica a realizar una función (o un pequeño conjunto de ellas). Las aplicaciones tradicionales de este tipo de sistemas incluirían sistemas de control en aviones, trenes, nudos de telecomunicaciones, motores de automóviles, procesos industriales, teléfonos móviles, etc.
Recientemente el consumo de energía en computadoras ha llegado a ser muy popular no sólo para sistemas de computadoras móviles para alargar la vida de las baterías, sino también en grandes sistemas consistentes de múltiples unidades de

procesamiento para reducir el consumo de energía y el costo de enfriamiento asociado. Desde que los procesadores consumen un gran porcentaje de energía en los sistemas de cómputo, especialmente en sistemas empotrados, muchos trabajos han sido realizados para manejar el consumo de energía de los procesadores.

La capacidad y el tiempo de vida de las baterías para las computadoras portátiles han mejorado mucho en los últimos años. Sin embargo los procesadores modernos consumen mucha más energía que los anteriores y cada nueva generación de computadoras portátiles trae consigo más dispositivos que consumen más energía. Por esta razón el manejo de la energía es más importante que nunca. La necesidad de uso más prolongado de la computadora trae como consecuencia el tener uno o más reemplazos de baterías. Pudiendo lograr mucho si se utilizan políticas inteligentes para el manejo de la energía.

2.2 Consumo de potencia en dispositivos.

Los dispositivos que consumen más energía en una computadora portátil son: el procesador, el disco duro y la pantalla. Estos se pueden configurar por separado. El manejo de potencia del procesador permite ajustar la frecuencia del procesador para ahorrar el máximo de energía sin que se pierda el desempeño de la computadora. El manejo de potencia del disco duro permite que se coloque en el umbral de su voltaje de alimentación cuando no se está utilizando y este mismo método es utilizado para el manejo de la potencia de la pantalla. Otros dispositivos como tarjetas gráficas, tarjetas de red inalámbricas y tarjetas de USB permiten que se pasen al estado de dormido.

En la figura 2.1 se puede ver la distribución del consumo de potencia en los diferentes dispositivos que integran una computadora portátil.

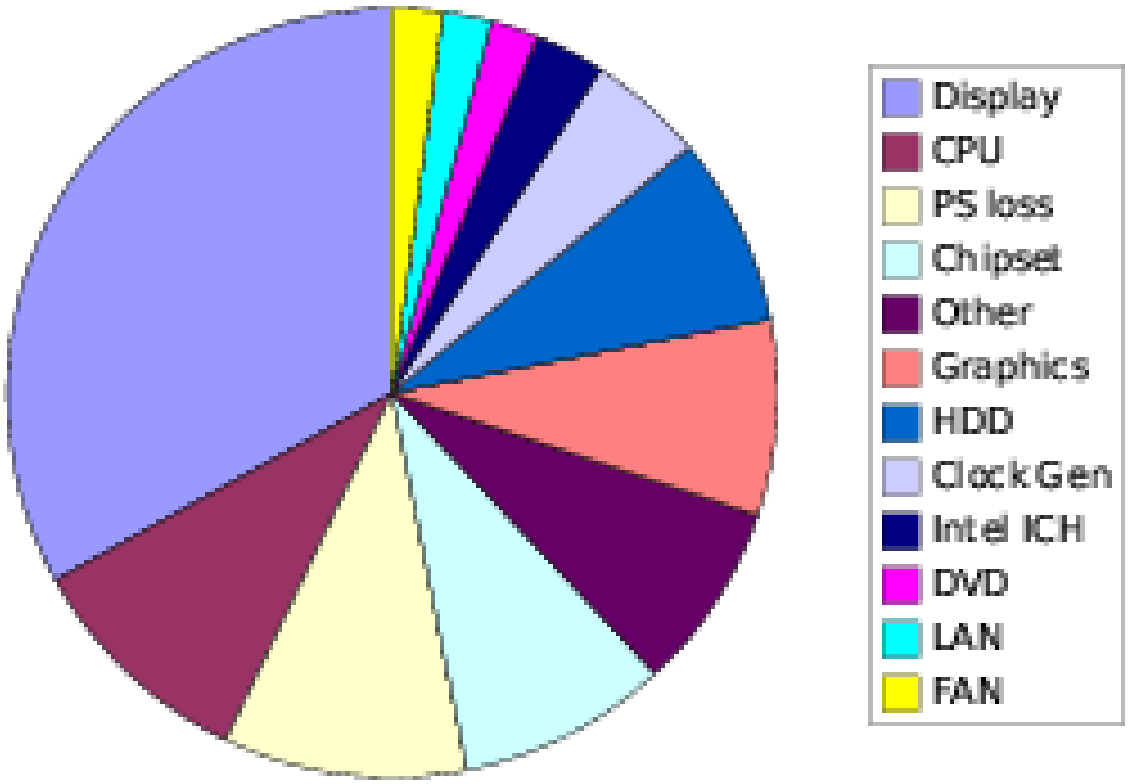


Figura 2.1. Consumo de potencia en los dispositivos.

2.3 Tecnología de baterías.

Las baterías están en todas partes, desde juguetes hasta automóviles. Las más utilizadas se desechan porque no se pueden recargar; las cuales son llamadas baterías "primarias", convencionales o alcalinas y se usan en múltiples aplicaciones, algunas tan vitales como la alimentación de marcapasos. Las baterías “secundarias” o recargables satisfacen necesidades muy distintas.

Actualmente existen dos tipos de baterías recargables que dominan el mercado: las baterías de plomo y las de níquel-cadmio.

Las baterías de plomo se encuentran en los automóviles sólo destinadas para el arranque, iluminación e ignición (no tienen suficiente energía para mover el

coche). Las baterías de níquel-cadmio a falta de mejores baterías, se emplean en artículos de electrónica de consumo como videocámaras y computadoras portátiles o teléfonos móviles.

Cualquiera que sea su uso, se requiere que sea mejorada la técnica de estas baterías, ya que se descargan solas, demasiado rápidamente y presentan un peculiar efecto memoria que reduce su capacidad.

Además de la necesidad de mejorar las técnicas de las baterías actuales, se debe saber que los elementos que las componen son altamente contaminantes, especialmente el plomo y el cadmio, y que en el caso de este último, los procesos de reciclado no están bien establecidos. Sin embargo, la demanda del mercado de las baterías recargables es previsible que siga creciendo tanto a corto como a medio plazo. Así, la búsqueda de baterías más ligeras y de mayor densidad de energía para el mercado de la electrónica de consumo es ya una necesidad urgente.

A este enorme mercado mundial habría que añadir a más largo plazo el no menos importante mercado de baterías recargables para tracción de automóviles eléctricos. En este campo la necesidad de mejora es igualmente patente. De hecho las baterías son el punto débil de los prototipos eléctricos que están empezando a salir ya al mercado del automóvil. Sus prestaciones limitadas y alto precio relativo hacen dura la competencia con vehículos convencionales de combustión. Sin embargo, existe una creciente demanda social de tecnologías limpias, más respetuosas con el medio ambiente que hacen especialmente deseable el desarrollo de vehículos eléctricos al menos para uso en entornos urbanos.

En general, cada tecnología tiene características que se ajustan mejor a ciertas aplicaciones, y existen asimismo numerosos y variados tipos de baterías que se pueden considerar hoy en día en estado de desarrollo.

Una breve lista podría incluir baterías Sodio/azufre, zinc/aire, hidruro metálico/óxido de níquel y baterías de litio. Todas tienen ventajas e inconvenientes que se intentan evitar con diseños adecuados pero las ***baterías de litio***, junto quizá a las de hidruro metálico son las que van encontrando un mayor consenso en cuanto a su potencial y un mayor esfuerzo en su investigación y desarrollo a nivel mundial.

Son muchas las razones que han originado este consenso. En primer lugar ***el litio es el metal más ligero*** y esto da lugar a una alta capacidad específica (figura 2.2), lo que permite obtener la misma energía con un peso muy inferior (figura 2.3).

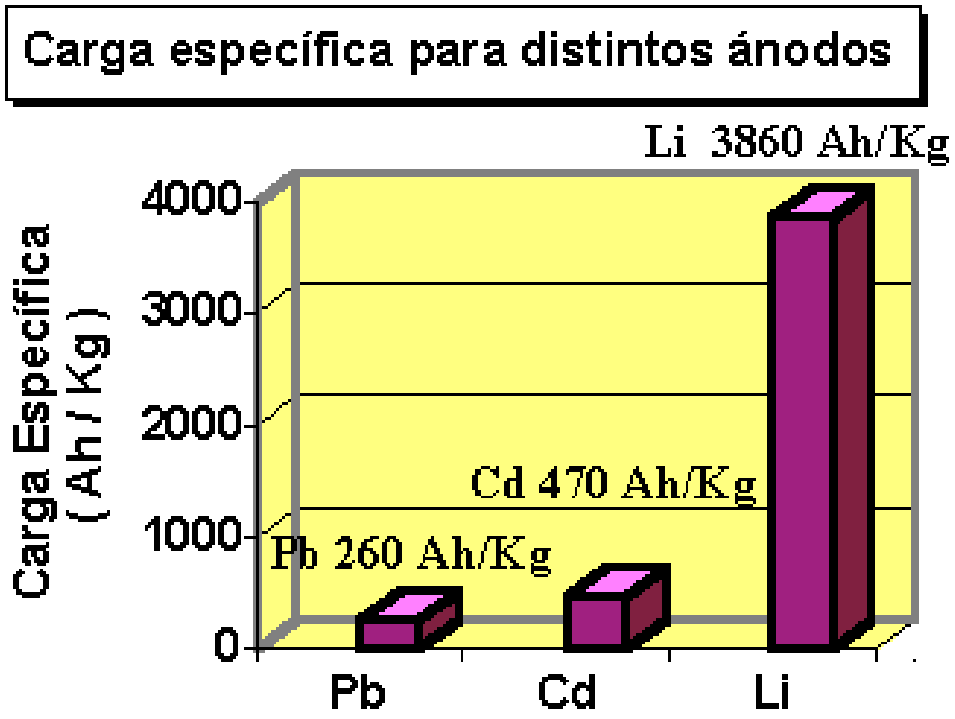


Figura 2.2. Capacidad específica.

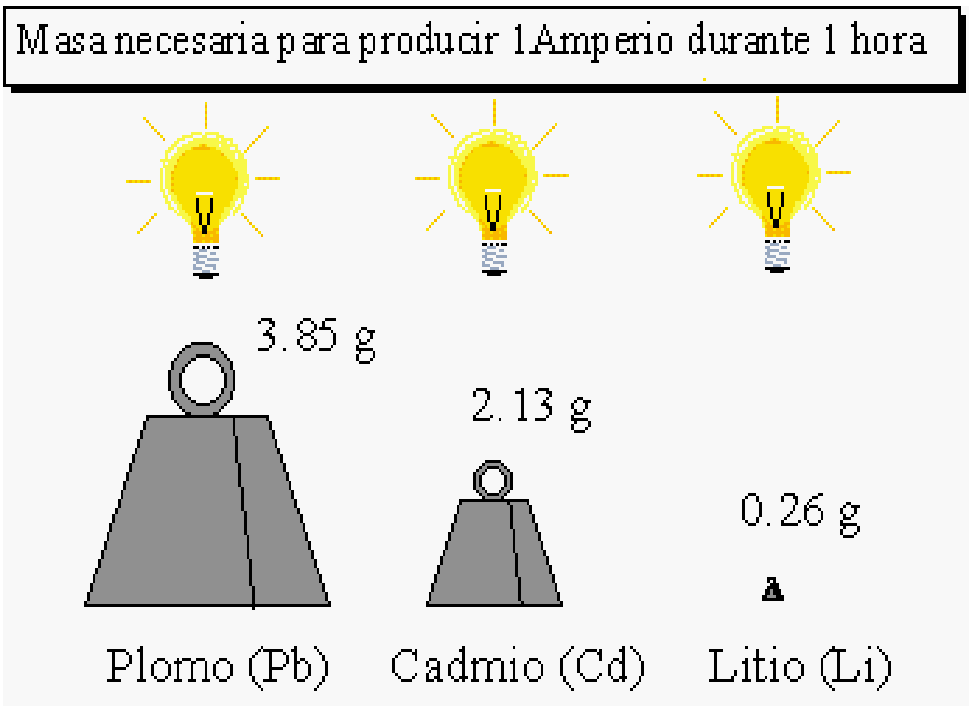


Figura 2.3. Masa necesaria..

Por otro lado, cuando un ánodo de litio metálico se combina con cátodos de ciertos óxidos de metales de transición las celdas electroquímicas reversibles que resultan presentan valores de voltaje superiores al de otros sistemas; ello contribuye a una alta densidad de energía. Además de sus características técnicas, la tecnología de litio es de las más versátiles y puede llegar a encontrar aplicaciones comerciales en muy distintos ámbitos, desde los que requieren pequeñas y delgadas micro baterías hasta baterías de alta capacidad y reducido peso para automóviles. Finalmente, y a diferencia del plomo o cadmio, los materiales que componen las baterías de litio más prometedoras no representan un problema de posible contaminación ambiental.

En los primeros prototipos de baterías de litio, el electrodo positivo (cátodo) era normalmente un óxido o sulfuro metálico con la capacidad de intercalar y des-intercalar iones litio en los procesos de descarga y carga de la batería de un modo reversible; el electrodo negativo (ánodo) en estos primeros sistemas estaba constituido por litio metálico que debía sufrir procesos igualmente reversibles de disolución durante la descarga y deposición durante la recarga. Para llegar a ser

realmente aplicables las baterías de litio han tenido que superar inconvenientes, algunos de ellos graves. El más serio obstáculo para la comercialización de baterías de litio recargables se derivó precisamente de la gran reactividad del litio metálico que podría representar problemas de seguridad; el uso del metal como ánodo se vio asociado a problemas de crecimiento dendrítico del litio durante los procesos de recarga continuados.

Este comportamiento llegó a ser causa de problemas de funcionamiento y seguridad. Afortunadamente estos problemas se resolvieron de forma totalmente satisfactoria con la introducción de dos variantes dentro de esta tecnología: las baterías de "ion-litio", y el desarrollo de electrolitos poliméricos plásticos menos reactivos que sus análogos líquidos. En las baterías de ion-litio el ánodo no está formado por litio metálico sino por otro material mucho más seguro, como por ejemplo el grafito, capaz de intercalar (o almacenar) iones de litio en una forma menos reactiva que la del litio metálico, sin un notable detrimento de su densidad energética.

Durante la descarga: Los iones litio cambian espontáneamente del electrodo negativo al electrolito y de éste al electrodo positivo. El electrolito permite el paso de iones pero no de electrones. Al mismo tiempo, los electrones fluyen espontáneamente del electrodo negativo al positivo a través del único camino que les dejamos libre: a través del circuito eléctrico. A medida que avanza la descarga, el potencial de cada electrodo cambia de forma que su diferencia disminuye y cae por tanto el voltaje de la celda a medida que sacamos carga eléctrica (Q) de la batería.

Durante la carga: Se bombea electrones en el electrodo negativo y son extraídos del positivo. Haciendo por tanto el electrodo negativo más negativo y el positivo más positivo y se aumenta, así la diferencia de potencial entre ellos, o lo que es lo mismo, el voltaje de la celda. Este proceso fuerza también a los iones litio a salir del electrodo positivo y a intercalarse en el negativo.

Este gran avance no sólo representó la introducción de una tecnología mucho más segura, sino que introdujo ventajas adicionales como el excelente comportamiento de reversibilidad durante los procesos de carga y descarga que es característico actualmente de las baterías de ion-litio.

Las baterías recargables de ión-litio que están en el mercado están compuestas de cátodos de $LiCoO_2$, electrolitos poliméricos y ánodos de grafito altamente densificados y con poca superficie para minimizar los fenómenos de pasivación que también les afectan. Se pueden recargar hasta 2500 veces y gracias a su bajo precio constituyen la mejor alternativa en el mercado de la electrónica de consumo.

2.3.1 Tecnología futura.

Puede una computadora portátil funcionar durante 10 horas o más antes de recargarse. Eso es lo que promete una nueva tecnología de baterías llamada células directas de combustible metanol (DMFC, por sus siglas en inglés). La DMFC, una posible sucesora de las baterías recargables y desechables que utilizan muchos de los dispositivos digitales móviles de hoy, genera energía mezclando el metanol con aire y agua.

El problema es que casi todas las compañías todavía están buscando la manera de aumentar la potencia que producen las DMFCs. Dentro de la célula de combustible, el agua y el metanol deben estar separados del catalizador por medio de una membrana. Mientras mayor sea la relación de metanol a agua, más potente será la DMFC (y más pequeña pudiera ser). Tampoco se ha decidido si las nuevas DMFCs serán desechables o recargables. Samsung recientemente mostró un prototipo de DMFC para portátiles. NEC también está trabajando en un modelo para PCs portátiles, e Hitachi tiene planes de vender una para PDAs.

2.4 Arquitectura para el ahorro de energía.

Minimizar el consumo de energía es uno de los principales objetivos de muchos usuarios de computadoras. Para los usuarios de laptop, ésto se refleja en la vida útil de la batería. Para los usuarios de una PC de escritorio, ésto afecta las necesidades de enfriamiento y, por tanto podrá causar problemas con el ruido de los ventiladores dentro de una oficina.

Para los usuarios de servidores, el costo de energía y de enfriamiento es una parte creciente de los gastos operacionales de un centro de datos. Para ayudar a enfrentar estos desafíos, los investigadores de Intel desarrollaron una nueva arquitectura de sistema para consumo y ahorro de energía (EESA) creada para aumentar el desempeño por watt por medio de la gestión del voltaje y de la frecuencia. La EESA utiliza sensores para identificar cuándo una función del sistema está ociosa y después envía este componente a un estado más bajo de consumo de energía. Los desarrolladores de la EESA utilizaron el trabajo conjunto de cinco tecnologías para minimizar el consumo y maximizar el ahorro de energía. Estas tecnologías son:

- Gestión de energía refinada
- Optimización de E/S
- Conversión de energía del sistema
- Arquitectura con sensor para cliente
- Gestión de la política de energía

Gestión de energía refinada (FGPM). La función de gestión de energía refinada (FGPM) es importante para la EESA y ofrece un control más preciso sobre los índices de energía dentro del sistema. Actualmente, la función de espera (sleep) del sistema se inicia sólo cuando todos los componentes están ociosos. La función FGPM, colocará funciones individuales del sistema en el estado de espera cuando

éstas estén ociosas, lo que dará como resultado un menor consumo de energía y un mayor ahorro de energía.

Optimización de E/S. La tecnología de la Optimización de E/S es capaz de reducir el consumo de energía en tres áreas utilizadas por el usuario: displays con pantallas con actualización automática, E/S administradas por la energía y audio con pantallas con actualización automática. La tecnología de la Optimización de E/S observa las interfaces de estos dispositivos para encontrar las maneras de reducir sus dependencias entre los ciclos de procesamiento del sistema. Si las funciones del sistema pudieran colocarse en el estado ocioso más frecuentemente, consecuentemente el consumo de energía será reducido. Los displays con pantallas con actualización automática se benefician por el hecho de que el contenido de la mayoría de los displays de computadoras no cambia muy rápidamente si estuvieran asociados a una PC de escritorio, una laptop, una PDA, o un teléfono celular. Esta es la naturaleza de las interacciones humanas con computadoras que llevan a los usuarios en poco tiempo a asimilar las informaciones que están exhibiéndose; por tanto los displays pueden permanecer inalterados algunos segundos hasta varios minutos por vez. En cuanto a eso, la lógica de gráficos por detrás del display consume energía al restaurar la pantalla varias veces por segundo. Los investigadores de Intel inventaron la función del display con pantallas con actualización automática para permitir que la lógica del display se coloque en espera cuando la pantalla está actualizándose. La idea es instalar una pequeña cantidad de memoria—la cual es cada vez más barata—en el propio panel del display. Esta memoria del caché almacena las informaciones que serán exhibidas al usuario. Durante el tiempo que el display no esté siendo alterado, éste toma la información del caché, como consecuencia la energía se reducirá colocando la lógica restante del display en espera.

En el futuro, la tecnología de display con pantallas con actualización automática será incorporada en los dispositivos móviles. Cuando una red sin cable sondea el dispositivo, un dispositivo de la EESA responderá a partir de un caché con

pantallas con actualización automática en vez de activar todo el procesador y el chipset para responder al sondeo. La tecnología de la E/S administradas por la energía ofrece realimentación a la función FGPM sobre el estado de la gestión de energía de los dispositivos del puerto USB (Universal Serial Bus) integrados en la computadora (dispositivos de E/S externos no son afectados). Si el dispositivo de USB estuviera ocioso, la FGPM lo coloca y el chipset lo direcciona hacia el modo en espera hasta que sus servicios sean necesarios. La función de audio con pantallas con actualización automática mantiene un gran buffer de audio que es usado para llevar la salida de audio al usuario. Después, el controlador de disco del sistema es accesado sólo ocasionalmente cuando el buffer necesite restaurarse. En el transcurso de estos accesos, el disco, el procesador y otros circuitos no utilizados pueden colocarse en estado de baja potencia o en estado de conservación de energía. El usuario continúa escuchando ininterrumpidamente, pero el resto de la máquina permanece desconectado el tiempo suficiente para reducir el consumo de energía.

Conversión de la energía del sistema. La función de conversión de la energía del sistema organiza el modo en como la energía es administrada desde su fuente hasta los circuitos que la utilizan. Las repetidas conversiones de energía, en las actuales computadoras, resultan en economías menores que un 50%— esto significa que la mitad de la energía es consumida en los propios procesos de conversión. Los investigadores de Intel están desarrollando métodos para mejorar estas economías, con el objetivo de alcanzar un 90% de reducción. Además, la función de conversión de la energía del sistema reaccionaría a temperaturas más altas del sistema por medio de la disminución de la velocidad del reloj en algunos componentes con el fin de disminuir el consumo y aumentar el ahorro de energía durante el proceso de conversión de energía. Disminuir el reloj significa reducir el consumo de energía en los circuitos CMOS, bajar la temperatura y permitir una mayor economía en el convertidor de energía. Para servidores con láminas (ver apéndice C), los sensores podrán ofrecer información a la FGPM para ayudar a balancear las cargas de la computadora utilizando una pila de láminas, de este

modo, se reduce la salida térmica de cada pila como un todo. Por ejemplo, si una lámina estuviera trabajando bastante y estuviera generando bastante calor, una segunda lámina subutilizada podría realizar parte del trabajo de la primera lámina, así el calor total generado por ambas láminas es reducido. Además, la función de conversión de la energía del sistema permitirá que componentes ociosos de un servidor de láminas se desconecten para conservar la energía y reducir la generación de calor. La necesidad de reducir la energía mediante el enfriamiento podría reducir aún más el consumo de energía cuando algunos ventiladores este desconectados. Estos mismos principios de la conversión de la energía del sistema también pueden beneficiar los dispositivos portátiles. La tecnología de Conversión de energía del sistema puede convertir la energía de la batería de forma más eficiente—permitiendo más horas de uso a los usuarios de los dispositivos móviles.

Arquitectura con sensor para cliente La arquitectura con sensor para cliente estandariza el modo como los sensores se comunican con la FGPM. Actualmente no existe ningún mecanismo para que varios sensores pasen su información de vuelta para la función de administración central. La arquitectura con sensor para cliente fue creada para organizar los sensores por medio de la identificación de sus capacidades, del monitoreo de sus funciones y de la definición de un proceso metódico para obtener informaciones para la FGPM de manera sistemática.

Gestión de la política de energía La tecnología de la gestión de la política de energía incorpora todas las informaciones anteriores y las controla, para maximizar totalmente la economía de la energía para la plataforma.

2.5 Manejo de potencia en el procesador Core 2 Duo.

Cuando AMD sacó al mercado sus procesadores Dual-Core no quedaron dudas: la arquitectura NetBurst se encontraba en sus límites, así como, basado en un

diseño que hizo popular a Intel en su época, nace el **Conroe**, un procesador que viene a reivindicar a Intel y que trae aparejado todo el beneficio del conocimiento que la empresa vino aprendiendo en todos estos años.

2.5.1 Análisis de la arquitectura:

Enfrentados a todos los problemas y limitaciones que la arquitectura NetBurst les dio en los últimos dos años, los ingenieros de Intel se propusieron dos metas fundamentales en esta nueva arquitectura: una es una mayor efectividad por reloj que la arquitectura anterior y la otra es un menor consumo de energía. Son 2 metas que curiosamente van en contra de lo que Intel nos estuvo mostrando en sus últimos procesadores Presler, por ejemplo. Aquí es donde vemos cómo las falencias o error involuntario en la arquitectura NetBurst no fueron tomadas como una derrota por parte de Intel sino como una forma de poder mejorarse y es justamente por eso que las metas que se impusieron esta vez fueron bien claras y muy exigentes.

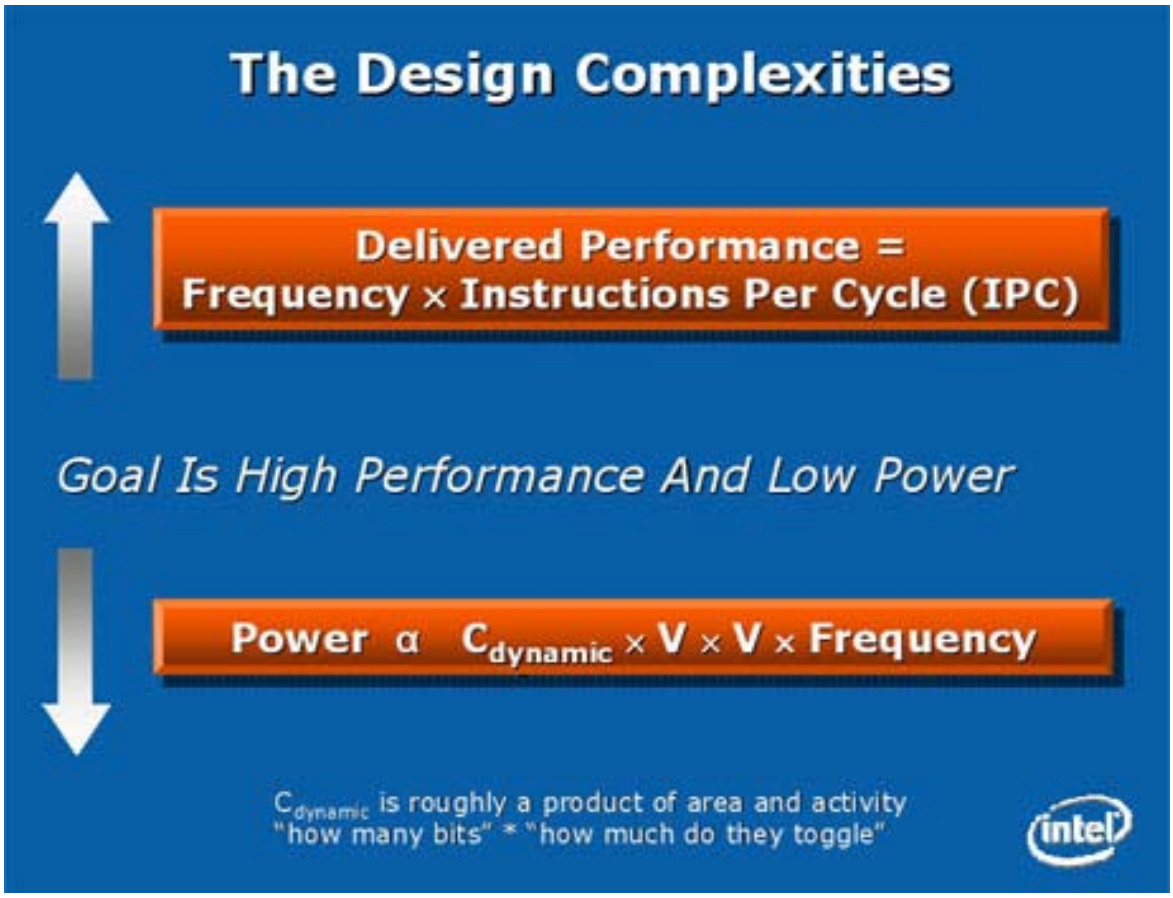


Figura 2.4 Complejidad del diseño. Copyright Intel.

En cualquier diseño de procesadores, hay ciertas variables matemáticas que se deben respetar, ya que son éstas las bases para lograr una arquitectura eficiente. Estas son, "C dynamic", Voltaje y Frecuencia (ver figura 2.4). Estas 3 variables dictan el consumo, frecuencia y movimiento de bits a lo largo de una arquitectura determinada. Debido a que estas variables se encuentran conectadas entre sí, cualquier modificación a una, afecta a la otra.

Lo primero que vemos es que la relación Voltaje x Frecuencia es 2 veces mayor. Esto quiere decir que por cada vez que el procesador sube linealmente su frecuencia de operación, aumenta de forma cuádruple su consumo de energía, esto es perfectamente válido en Overclocking, ya que aplicamos el concepto

opuesto para la misma ecuación, cuando un procesador llega a su límite de clock, es necesario subir el voltaje para permitirle obtener un clock más alto. Pero, como bien vemos en la ecuación, es necesario cuadruplicar el voltaje necesario para apenas subir linealmente su frecuencia. Al ver estos valores, se hizo claro para Intel que elevar su frecuencia de operación no es una opción, como tampoco jugar con el voltaje. La otra forma de elevar el rendimiento de un procesador es claro que está aumentando la cantidad de unidades de ejecución que conlleva en general a aumentar la eficiencia, esto es, la cantidad de instrucciones por ciclo de reloj que un procesador puede realizar, pero es aquí donde aparece la siguiente limitación.

El valor que todavía no mencionamos qué es el "C dynamic", explicar el funcionamiento de esta variable digamos que no es muy fácil, y es por eso que se decide expresarlo en castellano, para que pueda ser entendido. C dynamic es el valor que expresa el producto de un área de actividad de bits, o lo que es lo mismo que decir "cuantos bits" existen en un momento dado en nuestra área determinada, para este caso, el procesador. La cantidad de bits que en un momento determinado se mueven en el procesador determina el área total del "C dynamic". Como pueden ver, es un área no definida por un espacio físico ni por silicio, sinó más bien por "bits activos" en la arquitectura. El problema con esta variable es que a mayor número de bits activos existe, claro está, mayor energía, y por lo tanto, se eleva nuevamente el voltaje del procesador, negando toda posibilidad de lograr una arquitectura con bajo consumo.

Para elevar efectivamente el rendimiento, se deben aumentar la cantidad de unidades de ejecución, esto conlleva a una mayor cantidad de transistores y por consiguiente, potencialmente mayor cantidad de bits activos en un momento dado en el procesador. Esto directamente nos lleva a un aumento en el nivel de energía que el procesador consume y por tanto, aparece nuestro dilema.

El problema se evidencia en procesadores como el Presler, donde una excesiva

cantidad de transistores y su largo Pipeline de Ejecución provocan una gran cantidad de "bits activos" combinado esto con la baja eficiencia de instrucciones por segundo y con un alto clock elevan el consumo de energía a niveles desproporcionados, es por esto que el proceso de manufactura no afecta ni contribuye automáticamente a una mejor eficiencia en el procesador ni en el consumo, ya que el área de "C dynamic" no se encuentra definida por un proceso de miniaturización.

Aquí Intel se enfrenta a su segundo dilema, ya que debe lidiar con el problema de mejorar la arquitectura por medio de una mejor unidad de ejecución sin que esto introduzca una mayor carga de "bits activos", para lo cual recurrió a una serie de "trucos" por así decirlo que aprendió a lo largo del tiempo y por sobre todo con sus últimos procesadores Pentium-M.

En base a esta descripción, se analiza puntualmente por qué esta nueva arquitectura es más eficiente y consume menos energía.
Se añadieron cinco nuevas características para darle un mejor desempeño y un ahorro de energía mayor, estas características son las siguientes:

1) Amplia ejecución dinámica (Wide Dynamic Execution).
2) Impulsar los medios digitales avanzados (Advanced Digital Media Boost).
3) Capacidad de potencia inteligente (Intelligent Power Capability).
4) Acceso a memoria inteligente (Smart Memory Access).

De estas cuatro características sólo se explicará la capacidad de potencia inteligente.

Intel Intelligent power capability:

Debido a que el consumo de energía es el punto más importante para Intel en esta nueva arquitectura (ver figura 2.5), se tomaron pasos intensivos para llegar al nivel

incluso más pequeño en lo que respecta a control de la energía que se administra a cada parte del procesador.

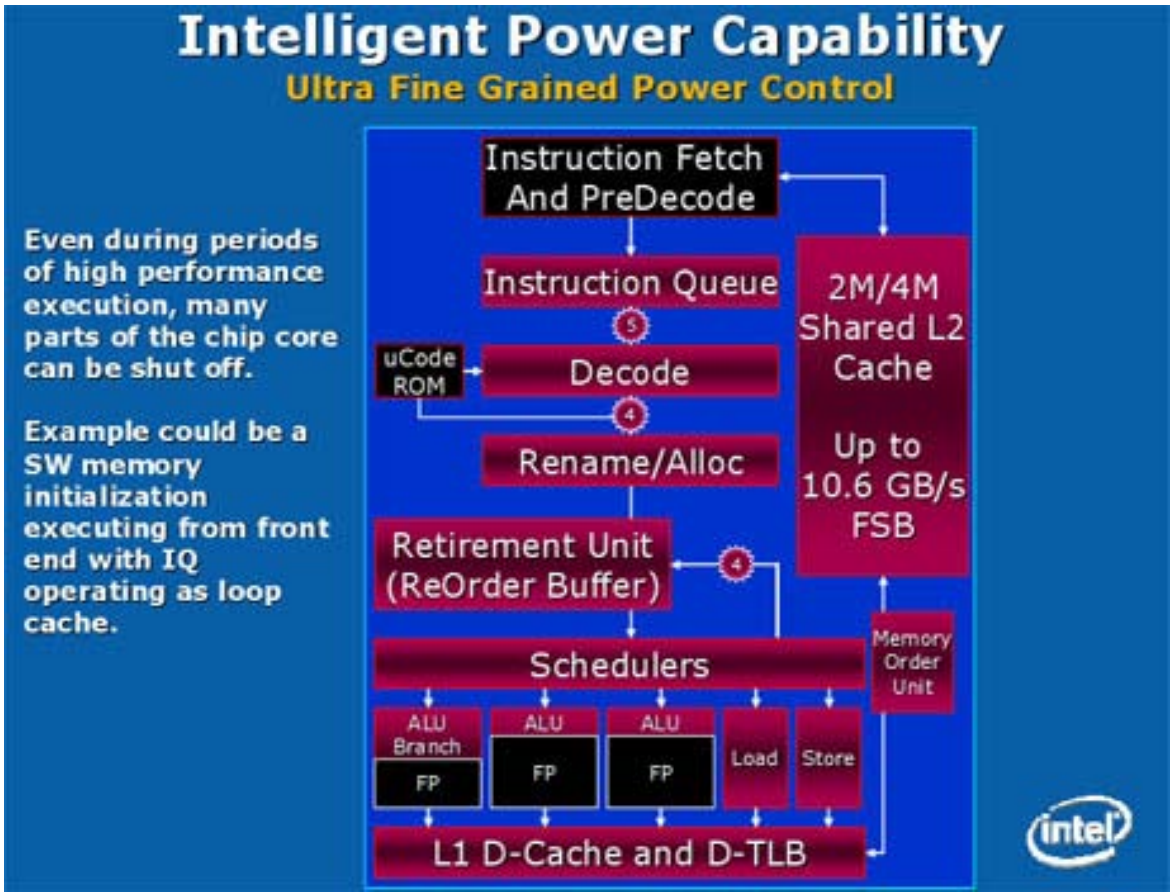


Figura 2.5. Arquitectura de la capacidad de potencia inteligente. Copyright Intel.

Como dicen los Ingenieros de Intel, el acercamiento que emplearon es el mismo que el de la Ley hacia las personas, "Ninguna parte del procesador es digna de ser utilizada hasta que demuestre lo contrario", y así es, la arquitectura Core utiliza un nivel muy refinado de desactivación a nivel "gating" que le permite desactivar prácticamente enormes cantidades del procesador cuando no se usan y deberán realmente demostrar una necesidad de utilización para re-activarse, esto salva una enorme cantidad de energía que no se utilizará, bajando el consumo y la generación de calor.

Figura 2.6 Nivel de abstracción. Copyright Intel.

Existen varios modos de dormir incluso hasta el modo que intel denomina sueño profundo (Deep Sleep), lo interesante de esta tecnología es la parte de "Intelligent", como su nombre lo indica; existen momentos donde en general, el procesador no se encuentra haciendo demasiadas cosas, imaginen un escenario, donde un procesador Dual Core, se encuentran en la pantalla inicial de Windows con tan sólo un programa de Mensajería y un navegador abierto, en este tipo de casos e incluso cuando nos encontramos jugando con algún juego que no utiliza Multithreading, es común que uno de los Cores (e incluso gran parte de tiempo del otro) no se utilicen, en este caso, la tecnología de ahorro de energía es capaz de desactivar enormes porciones de sistemas del procesador en desuso, incluso al nivel de dejarlo casi en stand-by, lo interesante es que no sólo se realiza mediante

Speed-steep (reducción del multiplicador) sino que se realiza a nivel muy interno (ver figura 2.6) a nivel transistores, lo cual le permite tener un control casi absoluto de qué es lo que hace cada parte de la arquitectura y qué partes pueden desactivarse, esta tecnología se encuentra "finamente" perfeccionada desde el Pentium-M, recuerden que su antecesor fue un procesador para computadoras portátiles donde el ahorro de energía es absolutamente importante.

La serie de adiciones que hizo Intel en esta nueva arquitectura realmente merecen mencionarlas como "Revolucionarias" por varios factores: primero, lograron duplicar e incluso triplicar el rendimiento de su antecesor, el Pentium4, claro está con otra arquitectura, pero con igual o mayor mérito, ya que pudieron aprender de sus propios errores y lo más importante es que lo hicieron a expensas de bajar considerablemente el consumo de energía, cosa difícil de hacer, muy difícil, pero que han demostrado ser posible, jugando con variables y aplicando todos los trucos aprendidos en el Pentium-M. Todo esto tuvo sus frutos en un Procesador que es increíblemente eficiente, sumamente fresco en operación y que tiene amplias posibilidades y un gran futuro por delante.

2.5.2 Administración de energía:

APM –ACPI

Todas las técnicas de gestión de energía (power management) requieren un hardware y una rutina de la BIOS apropiados. La mayoría de los ordenadores portátiles y muchos ordenadores de sobremesa y servidores cumplen estos requisitos.

En el hardware más antiguo se utiliza con frecuencia el estándar APM (Advanced Power Management). Debido a que APM consiste básicamente en un conjunto de funciones implementadas en la BIOS, existen diferencias en el soporte de APM en las distintas clases de hardware.

ACPI es todavía más complejo y la calidad de su soporte depende incluso en mayor medida del hardware utilizado.

Funciones para el ahorro de energía:

Stand-by (en reposo)

Solo se desactiva la pantalla y en algunos dispositivos se reduce también el rendimiento del procesador. No todas las implementaciones APM ofrecen esta función. En ACPI este estado se corresponde con S1.

Suspend (to memory)

Para este modo toda la información sobre el estado del sistema se guarda en la memoria y aparte de esta, todo el resto del sistema se detiene. Es un estado en el cual la computadora gasta muy poca energía. El atractivo especial de realizar esto con Linux es el no tener que parar el ordenador nunca; hay otros sistemas operativos que se vuelven inestables después de cierto tiempo. En la mayoría de los portátiles actuales basta con cerrar la tapa para suspender y abrirla después para seguir trabajando. En ACPI este estado se corresponde con S3. El soporte de este estado depende enormemente del hardware utilizado.

Hibernation (suspend to disk)

En este modo, la computadora vuelca todo el contenido de la memoria al disco duro y el sistema se detiene después. El ordenador tarda de 30 a 90 segundos de salir de este período de hibernación. Tras este periodo se restablece por completo el estado anterior al suspend. Algunos fabricantes ofrecen ciertos modos híbridos (por ejemplo RediSafe en IBM Thinkpads). En ACPI el estado de hibernación se corresponde con S4.

Control de batería

Junto a la información del estado de la batería también es importante tener algo previsto en caso de que disminuyan las reservas de energía. ACPI o APM desempeñan aquí esta función de control.

Apagado automático

Después de un shutdown la computadora se para completamente sin necesidad de pulsar el botón de apagar. Esto es importante en caso de que se realice un apagado automático poco antes de que se agote la batería.

Apagado de los componentes del sistema

El componente esencial a la hora de ahorrar energía es el disco duro. Dependiendo de la fiabilidad del sistema, éste se puede poner a dormir durante más o menos tiempo. El riesgo de una perdida de datos se incrementa con la duración del período de reposo de los discos. Se puede desactivar otros componentes vía ACPI (al menos en teoría) o de forma duradera en el setup de la BIOS.

Control del rendimiento del procesador

PowerNow! de AMD y SpeedStep de Intel son dos conceptos diseñados para disminuir el consumo de energía en todo el sistema. Con este fin se reduce la energía utilizada por el componente que normalmente más consume: el procesador. La menor producción de calor constituye un agradable efecto secundario ya que los ventiladores regulables pueden trabajar de forma más silenciosa. Las funciones CPU Frecuency Scaling del Kernel de Linux se encargan de regular estos procesos. En este contexto se distingue entre tres niveles de rendimiento del procesador:

Performance

Máximo nivel de rendimiento del procesador; se recomienda utilizarlo cuando se trabaja con el sistema conectado a la red de suministro eléctrico.

Powersave

Mínimo nivel de rendimiento del procesador para el uso del portátil con baterías.

Dynamic

Ajuste automático del rendimiento del procesador a la carga actual del procesador. Esta es la opción más recomendable en la operación con o sin baterías para ahorrar energía, evitar ruidos y lograr un rendimiento óptimo. El cambio de frecuencia o estado es tan suave que el usuario ni siquiera lo nota en un entorno operativo normal.

2.6 Consumo de energía en sistemas operativos.

Un análisis básico realizado en Phoronix revela los consumos de energía en diferentes sistemas operativos. En las pruebas compararon el consumo de energía del sistema operativo Ubuntu 7.10 RC1, Fedora 8 Test 3, Windows XP SP2 y Windows Vista. Resultando como ganadores en consumo a los sistemas operativos Windows. En estado de inactividad (’idle’) los cuatro sistemas se situaron entre los 38 vatios de Fedora y los 41 de Ubuntu, figura 2.7; pero en el uso normal del ordenador fue donde tanto Fedora como Ubuntu superaron claramente en consumo a Windows XP y Windows Vista, ver figura 2.8.

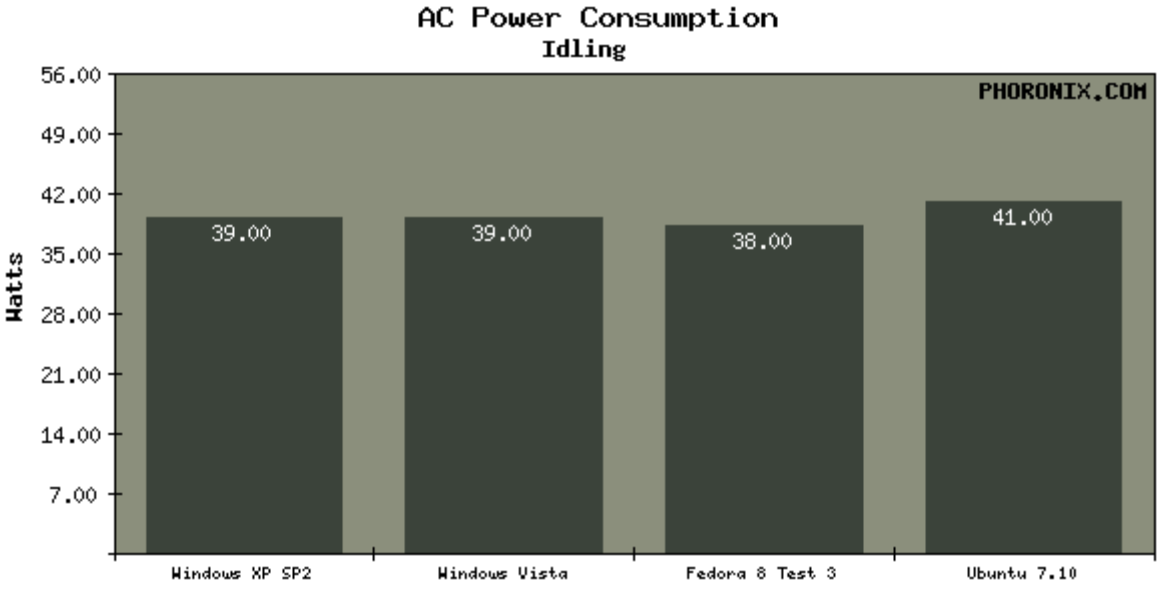


Figura 2.7. Consumo de potencia en estado desocupado. Phoronix corporation.

http://www.theinquirer.es/wp-content/uploads/2007/10/consumo-idle.png. Toda una sorpresa, sobre todo teniendo en cuenta que muchos suponían que Windows Vista era un verdadero tragón de recursos. Aún así y como indican en Phoronix, las pruebas de consumo han sido muy básicas, y se pueden localizar en [35].http://www.theinquirer.es/wp-content/uploads/2007/10/consumo-desktop.jpg

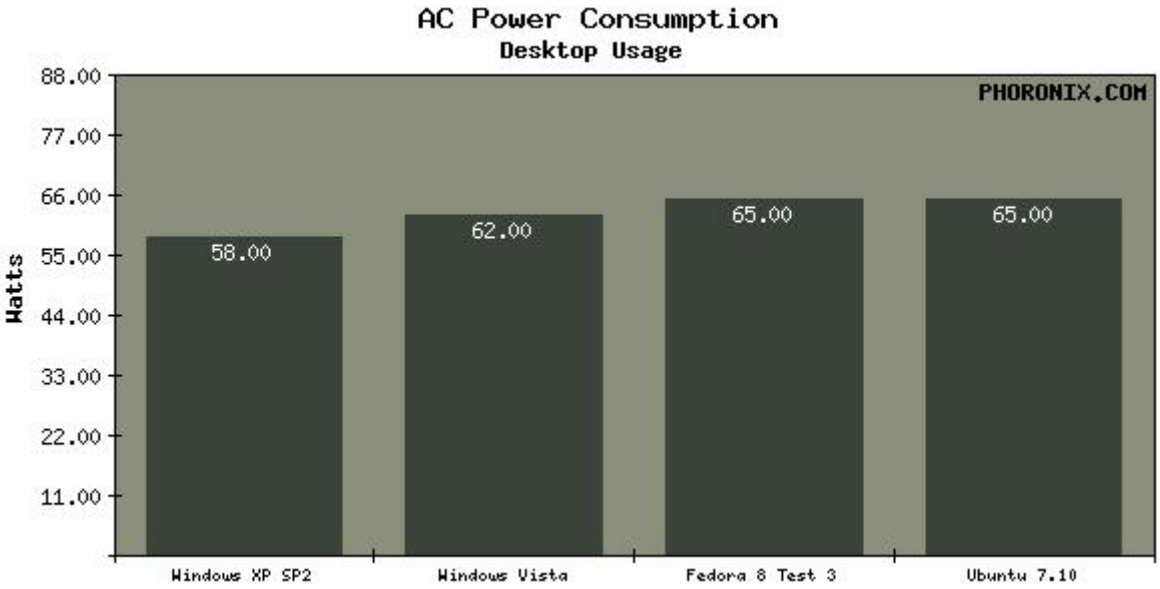


Figura 2.8. Consumo de energía con carga. Phoronix corporation.

2.7 Técnicas para el ahorro de energía.

La principal técnica en computación para bajar la potencia de consumo de los procesadores, es la técnica de escalamiento de voltaje dinámico (DVS) [5, 6, 24, 58, 76], para sistemas de tiempo real de un procesador. La técnica depende de la capacidad para ajustar la velocidad/voltaje de los microprocesadores y de explotar la relación convexa entre la velocidad del CPU y el consumo de la potencia. En principio es posible ahorrar energía, reduciendo la velocidad de CPU. Por otro lado, la factibilidad de la planificación deberá ser conservada a pesar del problema de reducir la velocidad y el minimizar el consumo de energía mientras se conservan los plazos de respuesta.

Otros trabajos de investigación sobre DVS asumen que todas las tareas son independientes y operan bajo procesadores de voltaje continuo. En [16] los autores integran el análisis temporal probabilístico con técnicas para minimizar la energía de una sola tarea, para un procesador con voltaje continuo. En este mismo ambiente, en [53], se demostró y presentó una evaluación extensiva del esquema MEEC (Minimize the Expected Energy Consumption) propuesto en [54], con el objetivo de minimizar el consumo de energía esperado, este esquema fue derivado de la suposición del manejo de la frecuencia continua sin restricciones, eso quiere decir, que la frecuencia del procesador puede ser ajustado continuamente de 0 a infinito. El esquema está dividido en dos fases. a) Fuera de línea: recalcula la velocidad planificable, la cual consiste de un factor de porcentaje de cada tarea. b) En línea: invocada antes de la ejecución de cada tarea, obtiene el tiempo del frame y calcula la velocidad de ejecución de la tarea. La fase fuera de línea corre en tiempo polinomial y la fase en línea corre en un tiempo constante. Propusieron en [43], Labarta y otros una versión del algoritmo PLMDP (Power Low Modified Dual Priority Scheduling) propuesto en [42] llamado EPLDP-m (Enhanced Power Low Dual Priority), que garantiza conocer las restricciones temporales y tener una reducción significativa del consumo de energía. Tarek presentó en [65] un problema de optimización para operar tareas periódicas independientes en un procesador y en un ambiente con una energía limitada y fija. El objetivo es minimizar el número de fallas dinámicas (en términos de restricciones de los plazos de respuesta ((m, k)-suave).

Continuando con los trabajos presentados sin restricciones de precedencia pero en un ambiente de múltiples procesadores y a una velocidad discreta, podemos mencionar el trabajo presentado en [64], en donde proponen tres opciones diferentes. a) Una prueba para el control de admisión RMS. b) Una heurística de particionamiento y c) un algoritmo para asignar la velocidad. En modo fuera de línea las tareas pueden ser ordenadas acorde a sus valores de sus utilizaciones y demostraron que la heurística Worst-Fit es la mejor de entre todas las probadas. En el modo en línea el algoritmo se basa en reservar un subconjunto de

procesadores para tareas ligeras y así garantizar su desempeño consistente. En [33] se presenta un trabajo con el objetivo de minimizar el consumo de energía, proponiendo un algoritmo aproximado a 1.412, derivado de una planificación factible.

Usando una suposición alternativa, pocos trabajos de investigación [41, 72, 81] extienden el trabajo previo de DVS a considerar tareas con restricciones de precedencia (grafo de tareas) para un procesador y múltiples procesadores. El trabajo en [72] introduce la técnica DVS para un grafo acíclico dirigido llamado grafo de tareas condicional (GTC) ejecutándose en múltiples procesadores heterogéneos. La contribución de Wu es la de explotar el tiempo disponible de holgura en el peor caso, tomando en cuenta el comportamiento condicional del GTC. Un algoritmo genético basado en técnicas de mapeo es usado para optimizar la implementación del sistema. Este trabajo está direccionado a reducir el consumo de energía en un sistema empotrado compuesto de un grafo de tareas, el inconveniente es la impredecibilidad en los tiempos de ejecución de las tareas del algoritmo genético. El algoritmo de robo de holgura greedy propuesto en [80] maneja el consumo de energía para aplicaciones de tiempo-real corriendo en un sistema de multiprocesadores. Se utiliza un grafo AND/OR para representar el flujo de control y la dependencia de los datos entre las tareas. En [56] se presenta una planificación de tareas periódicas y aperiódicas en un sistema distribuido empotrado de tiempo-real, las tareas están compuestas de un grafo acíclico dirigido. Los plazos de respuesta de las tareas aperiódicas duras son garantizados reservando ranuras (slots) de ejecución en una planificación estática. Un planificador en línea supera a la planificación estática mejorando los tiempos de respuesta de las tareas aperiódicas. Una técnica de planificación DVS es introducida en un planificador en línea para balancear el ahorro de energía y los tiempos de respuesta de las tareas suaves, mientras se mantiene una planificación factible. Finalmente en [79], Yumin y otros presentan un trabajo dividido en dos fases que integran la asignación de tareas, ordenamiento y selección del voltaje para minimizar el consumo de energía de tiempo-real con

restricciones de precedencia y ejecutándose en uno y en múltiples procesadores de voltaje variable. El problema fue presentado utilizando programación entera, en donde, puede ser resuelto en forma óptima en tiempo polinomial para voltajes continuos y algunos casos de voltajes discretos o puede ser resuelto en forma aproximado para voltajes discretos generales.

2.8 Manejo de potencia.

El ahorro de energía S_i de la tarea T_i es calculado por $S_i = \max_i(E_i) - E_i$, donde $\max_i(E_i)$ es el máximo consumo de energía de todas las tareas en el sistema donde $i = 1,...,n,$ n es el número de tareas y E_i es el consumo de energía de la tarea T_i. Notar que el máximo ahorro de energía para la tarea será igual a 0. Λ_i denota la velocidad de ejecución de una instancia de la tarea T_i. El consumo de la potencia de la tarea T_i es denotado por $g_i(\Lambda)$, asumiendo que será estrictamente una función convexa creciente [12], específicamente un polinomio de al menos de segundo grado. S_i la tarea T_i ocupa el procesador durante el intervalo de tiempo $[t_1, t_2]$, entonces el consumo de energía durante el intervalo es

$$E(t_1, t_2) = \int_{t_1}^{t_2} g_i(\Lambda(t))dt \quad (1)$$

Finalmente, una planificación con una energía óptima es factible, si el consumo de energía total para la ejecución completa del sistema es mínimo. Considerando una velocidad constante f_i, (dado en ciclos por segundo), el tiempo de ejecución de la tarea T_i será $t_i = C_i / f_i$, una planificación de las tareas periódicas es factible, si cada tarea T_i es asignada al menos C_i al CPU antes de su plazo de respuesta en cada instante. La utilización de una tarea será la demanda que tiene el procesador para su ejecución:

$$U_i = t_i / P_i \quad o \quad (C_i / f_i P_i) \quad (2)$$

Acorde al algoritmo de planificación EDF, el conjunto de tareas es factibles (las tareas no pierden sus plazos de respuesta), si la utilización del sistema es menor o igual a la capacidad total del sistema $\sum U_i \leq 100\%$.

El consumo de potencia en los procesadores CMOS es dominada por la disipación de potencia dinámica P_d, la cual es dada por la siguiente expresión $P_d = C_{ef} \circ V_{dd}^2 \circ f$, donde C_{ef} es la capacitancia de intercambio efectiva, V_{dd} es el voltaje de alimentación del procesador y f es la frecuencia de reloj del procesador. La velocidad del procesador f está casi linealmente relacionada al voltaje de alimentación por la siguiente expresión:

$$f = k \bullet \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{3}$$

Donde k es una constante y V_t es el voltaje umbral [13, 14]. Se asume que el desempeño y el consumo de potencia de los sistemas considerados son manejables. Esto quiere decir que un subsistema a través de hardware tiene la capacidad de cambiar sus estados de trabajo. Por ejemplo, RDRAM es una memoria para manejar la energía en forma eficiente, ésta puede ser puesta dentro de diferentes estados para el ahorro de la energía, teniendo diferentes tiempos de respuesta [51].

2.9 Procesadores de velocidad variada.

Por otra parte, los procesadores de voltaje variable tal como el Transmeta [29], Intel Xscale [26], AMD K6+ [49], Strong ARM [31] y el MPC5200 [28], pueden cambiar su consumo de potencia, ajustando la frecuencia del procesador y el voltaje suministrado dentro de ciertos rangos.

2.9.1. Procesador Intel Xscale.

Aunque la potencia dinámica domina a la disipación de potencia del procesador, la potencia de fuga estática crece muy rápida con los avances tecnológicos y

probablemente será comparable a la disipación de potencia dinámica dentro de pocas generaciones de tecnología, específicamente con la tecnología del sub-micrón. Por ejemplo, la potencia de fuga estática del procesador para la tecnología de 1μm fue de 0.01% de la potencia total, pero se está acercándose al 10% para la tecnología de 0.1μm [62].

Además de los procesadores, hay otros componentes que consumen potencia en un sistema. La energía está definida como la integral de potencia en el tiempo, y el consumo de energía E del sistema para ejecutar una aplicación a un nivel de potencia P en el tiempo t está definida como: $E = \int P(t)dt$. Mientras que reduciendo la frecuencia y el voltaje puede ahorrar energía dinámica, no puede ahorrar la energía total del sistema para ejecutar una aplicación específica por que la aplicación tardará más tiempo y consumirá más energía en otros componentes. Así, es necesario incorporar la potencia consumida por otros componentes en el modelo de la potencia cuando sean aplicadas las técnicas de frecuencia y voltaje para el ahorro de energía.

La potencia de fuga estática puede ser removida poniendo a los procesadores en estado de dormir cuando el sistema está desocupado. Por ejemplo, la potencia activa máxima para el procesador Intel Pentium-M está alrededor de 25Watts y la potencia activa mínima está alrededor de 4Watts. Sin embargo, estos sólo consumen alrededor de 0.5Watts en el estado de dormido y necesita pocos ciclos para regresar al estado de funcionamiento activo [30].

Así, cuando no hay ningún requerimiento de cómputo, se puede ponerle sistema dentro de un estado dormido de ahorro de potencia, desde el cual un sistema puede rápidamente (es decir, en pocos ciclos de reloj) responder a los requerimientos de cómputo. Es más, para un máximo ahorro de energía, podemos apagar una unidad de procesamiento cuando esté y probablemente permanezca desocupado por un tiempo y será alimentado nuevamente cuando sea necesitado.

Sin embargo, apagando y encendiendo un sistema lleva tiempo y energía, sobrecargando al sistema. [47].

Para incorporar todos los componentes que consumen potencia en una unidad de procesamiento y conservar el modelo más simple de potencia, se asume que una unidad de procesamiento tiene tres estados: activo, dormido y apagado. El estado activo se define como estando realizando cómputo de una tarea. Dependiendo de las diferentes frecuencias de procesamiento, toda la potencia estática y las cantidades diferentes de potencia dinámica son consumidas en el estado activo. El estado dormido es un estado para ahorrar potencia ya que elimina toda la potencia dinámica y la mayoría de la potencia estática. Las unidades de procesamiento en el estado dormido pueden despertar rápidamente (es decir, en unos cuantos ciclos de reloj) al nuevo requerimiento de cómputo. La sobrecarga de la transición del estado dormido al estado activo se asume que es despreciable. Se asume que la unidad de procesamiento no consume potencia en el estado de apagado. Para unidades de procesamiento que manejan automáticamente la potencia (es decir, despertar una red LAN), el componente correspondiente (es decir, la interfaz de red) necesita estar en un estado activo mientras las unidades de procesamiento están apagadas. Así, el consumo de potencia de una unidad de proceso a la frecuencia f puede ser modelada como:

$$P(f) = P_s + \hbar(P_{ind} P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \quad (4)$$

Donde P_s es la potencia del estado dormido; P_{ind} y P_d es la frecuencia independiente y la frecuencia dependiente de la potencia activa, respectivamente. $\hbar$ es igual a 1 si el sistema está activo y 0 si el sistema está en el estado dormido. C_{ef} y $m(>2)$ son las constantes dependientes del sistema y f es la frecuencia de procesamiento.

El potencia dormida P_s incluye (pero no se limita a) la potencia para conservar los circuitos básicos activos, que el generador de reloj siga corriendo y para mantener el procesador y la memoria en un estado de dormido [30, 2]. La potencia activa es dividida en dos partes: la potencia activa independiente de la frecuencia y la potencia activa dependiente de la frecuencia. La potencia activa independiente de la frecuencia consiste en parte de la memoria y potencia del procesador así como cualquier potencia que pueda ser eliminada poniendo las unidades de procesamiento en dormido y será independiente de la frecuencia y del voltaje de alimentación. La potencia activa dependiente de la frecuencia incluye la potencia dinámica del procesador y la potencia que dependen de la frecuencia de procesamiento y voltaje de alimentación [13, 3].

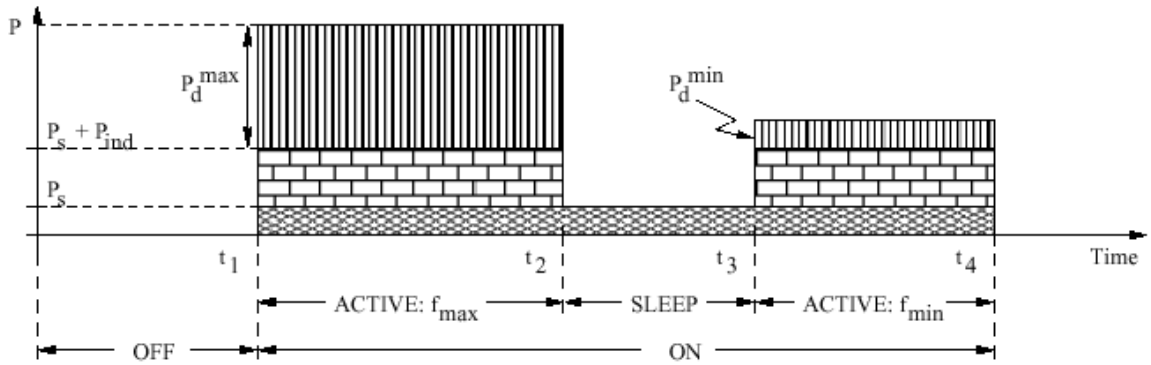


Figura 2.9. Modelo de potencia sencillo: consumo de potencia en diferentes estados.

El modelo de potencia más simple es ilustrado en la figura 2.9, donde el consumo de potencia para diferentes estados es mostrado para una unidad de procesamiento. En la figura, el eje Y es la potencia total en una unidad de procesamiento y el eje X representa el tiempo. Inicialmente, la unidad del procesamiento está apagada y no consume potencia.

De t_1 a t_2, pasa al estado activo y corre a la frecuencia $f_{\max}$. Así, la potencia máxima $(P_s + P_{ind} + P_d^{\max})$ es consumida. Después de t_2, la unidad de procesamiento es puesta en el estado de dormido dado que no hay cómputo que

realizar y sólo la potencia del estado de dormido P_s es consumida. Durante el período de tiempo (t_3 y t_4), es la ganancia activa y corre a $f_{\min}$. Mientras la potencia activa dependiente de la frecuencia se reduce a $P_d^{\min}$, P_s y P_{ind} permanecen los mismos durante el estado activo.

Note que, por simplicidad, la transición de estado apagado al estado activo no es mostrada en la figura. Sin embargo, toma una considerable cantidad de tiempo y consume un cierto nivel de potencia para encender una unidad de procesamiento. Por ejemplo, ha sido reportado arriba de 1 minuto que tarda en encender una PC con Windows instalado [47].

Para validar la efectividad del modelo cuando se modele el consumo de potencia en el sistema, se debe analizar el consumo de algún procesador (por ejemplo, Intel XScale) de velocidad variable y determinar los parámetros en el modelo de la potencia [25]. Las frecuencias y los correspondientes consumos de potencia para el procesador Intel XScale son mostradas en la tabla 2.1. Notar que la parte de la potencia para cada nivel de frecuencia es la potencia en el estado de dormido P_s y la potencia activa independiente de la frecuencia P_{ind}. Considerando las características de potencia de otros componentes en un sistema, sus consumos de potencia contribuyen principalmente a P_s y P_{ind}. Sin embargo, debido a la falta del número de potencia real medido, se valida el modelo de potencia considerando sólo los procesadores.

Tabla 2.1. Velocidad y voltaje del Intel-XScale

$f(MHz)$	1000	800	600	400	150
$V_{dd}(V)$	1.80	1.60	1.30	1.00	0.75

Usando técnicas de regresión no lineal para ajuste de curvas [27], se ajustan las potencias y las frecuencias del procesador Intel XScale con un modelo de potencia

sencillo. Así que los diferentes valores de P_s y P_{ind}, después de ajustar la frecuencia y el número de la potencia con un modelo sencillo, son obtenidos diferentes errores estándares y coeficientes de correlación. El error estándar más pequeño y los coeficientes de correlación más grandes indican mejor ajuste entre el modelo y el par de la potencia y la frecuencia. La figura 2.9 muestra los errores estándares y los coeficientes de correlación para diferentes valores de $P_s + P_{ind}$ cuando se realiza un ajuste de curvas de regresión no lineal a la potencia y la frecuencia del modelo de potencia, ver [82].

De la figura 2.10, se puede ver que cuando $P_s + P_{ind} = 0.028W$, se obtiene el mejor ajuste con el menor error estándar 0.01517 y el máximo coeficiente de correlación 0.99979. Para todos los valores de $P_s + P_{ind}$ de 0.01 a 0.04 cuando se ajuste la potencia y la frecuencia con el modelo de potencia, el coeficiente C_{ef} tiene un rango de 1.592 a 1.566 y el exponente m tiene un rango de 2.644 a 2.767, respectivamente. Para el caso del mejor ajuste (es decir, $P_s + P_{ind} = 0.028W$), se tiene $C_{ef} = 1.577$ y $m = 2.717$. Notar que, si la potencia modelada como un polinomio de 3er grado relacionado con la frecuencia, es obtenido $P = 0.121 - 0.7664 f + 1.8978 f^2 + 0.3475 f^3$ con un error estándar de 0.000778 y un coeficiente de correlación de 0.999999. Considerado una diferencia pequeña (0.02% en el coeficiente de la correlación) entre estos dos modelos, considerando el modelo de potencia más sencillo.

2.10 Porcentaje de ahorro de energía.

En la figura 2.11 se aprecia el porcentaje de ahorro de energía. Observando un máximo ahorro del 82% en [48] y un valor máximo del 84% cuando es utilizado un stream o trama encriptado de audio de alta calidad corriendo a 512 Kb/s. El segundo ahorro importante se presenta en [15] del orden del 78%, en donde utilizaron un simulador para manejar eventos y un 77% en [22] cuando los plazos

de respuestas son relajados al 50%. Los tres primeros artículos de la tabla 2.11, presentan el procesamiento de imágenes como aplicación. En [48] Trevor la descompresión de video MPEG como el corazón de la aplicación, usando la trasformada coseno discreta para descomprimir tramas o frames de video MPEG, con un 24% de ahorro y un 60% de ahorro para el óptimo. Vardhan en [69] utiliza la transformada coseno discreta para decodificar video y un 35% de ahorro. Finalmente en [53] Ruibin obtuvo un 33.45% de ahorro de energía, utilizando un reconocedor automático de objetos.

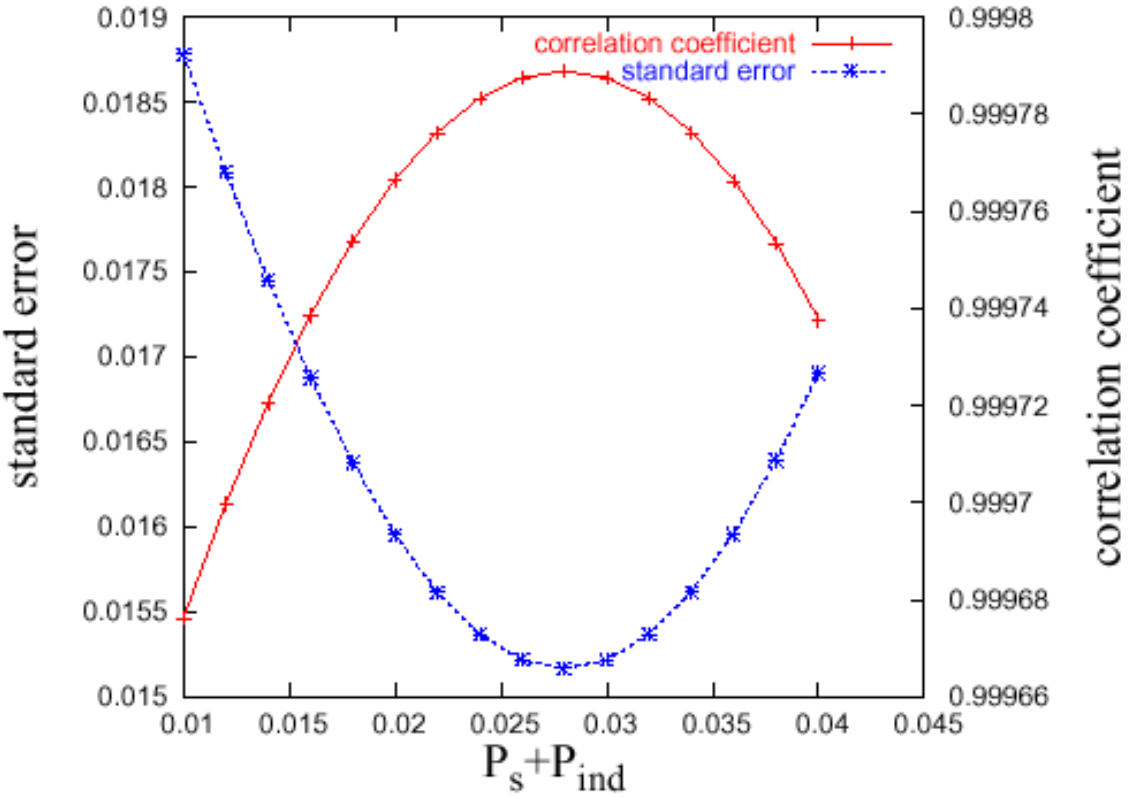


Figura 2.10. Los errores estándares y los coeficientes de correlación para diferentes valores de $P_s + P_{ind}$ cuando se ajustan la potencia y la frecuencia del procesador Intel XScale.

Tabla 2.2. Porcentaje de ahorro de energía.

Autor	Aplicación	Implementación	Año	% de Ahorro máximo
Trevor Pering [48]	Simulación	PDA utilizando 3 benchmarks	1998	UI 50% Audio 82% y 84% óptimo MPEG 24% y 60% óptimo
Vardhan V. [69]	Dispositivos de multimedia móviles	Laptop, Pentium M, Linux 2.6.8-1	2005	35%
Ruibin Xu [53]	Simulación	X-Scale	2005	33.45%
Gruian Flavius [22]	Simulación	Sun Ultra 10 Workstation 440MHz, 10 procesadores.	2001	77%
Hakan Aydin [5]	Simulación	Pc	2001	32%
Roychowdhory [52]	Simulación	12 multiprocesadores	2001	68%
Pillai P. [49]	Simulación	Laptop, HP N3350, AMD K6-2	2001	40%
Gruian Flavius [21]	RTOS	Pc	2001	38%
Yumin Zhang [80]	Strong ARM SA-1100	Pc	2003	25.8%
Wu Dong [72]	Simulación	Pc, Pentium III 866 MHz	2003	44.84%
Ala'Qadi [50]	Robotic Highway Safety Marker real-time application	Rabbit 2000 test board	2003	83%
Mejia Alvarez [45]	Simulación	Pc	2004	28%
Cheng Hui [15]	Simulación	Transmeta Crusoe	2005	78%
Yan Zhang [78]	Simulación	Pc	2005	30% monotarea 74% multitareas
Xiliang Zhong [73]	Simulación	Pc	2006	16% más que el óptimo
Bu Aiguo [11]	Simulación	Pc	2006	68%
Xiliang Zhong [74]	Simulación	Pc simula al Rabbit 2000 processor para el Robotic Highway Safety Marker real-time application	2007	21% comparado con el peor tiempo de ejecución
Xiliang Zhong [75]	Evaluación en un ADS Bitsy Xb platform con Interl Xcale PXA270 microprocesador	Pc, Pentium 4 a 2 GHz	2007	40%

RESUMEN

En este capítulo se presentó el estado del arte en el área de planificación de tareas de tiempo real con restricciones de precedencia y/o restricciones de potencia. Primeramente una breve semblanza referente al ahorro de energía fue presentado y el consumo de potencia en los dispositivos que integran una PC portátil. Luego se detalló la tecnología de las baterías y su tecnología futura. Continuando con la arquitectura para el ahorro de energía, así como la implementación para el ahorro de energía en un procesador Core 2 Duo. En seguida se presentó el consumo en cuatro sistemas operativos comerciales y a continuación con las técnicas y el manejo para ahorrar energía. Por último se explicaron brevemente los parámetros utilizados para la implementación de un procesador de velocidad variable.

CAPITULO 3

FORMULACION DEL PROBLEMA

En este capítulo se presenta como se formularon los objetivos específicos presentados en el capítulo 1. Y cómo se obtuvieron las soluciones a estos problemas planteados. Dando un ejemplo de cómo funciona el algoritmo propuesto.

3.1 Problema de cálculo de variaciones.

El problema puede ser formulado como sigue. Cada vez que arribe o salga un DAG del sistema, el problema es determinar el modo (velocidad) de ejecución de las tareas (o nodos en el grafo), tal que no se pierdan sus plazos de respuesta y el ahorro de energía del sistema sea maximizado. Todas las tareas tendrán varios modos de ejecución, donde se escogerá un modo para su ejecución. Notar que la solución a este problema deberá ser calculado cada vez que un DAG arribe o salga del sistema, además una solución con una alta complejidad computacional puede ocasionar probablemente que se pierdan los plazos de respuesta.

Se considera que la planificación de todas las tareas, puede ser repetida cada hiperperíodo hyp sin que se pierda la factibilidad. Nos enfocaremos en maximizar la suma de las utilizaciones a diferentes velocidades durante $hyp = LCM\{hyp_1,...,hyp_n\}$ (mínimo común múltiplo).

El objetivo del problema es seleccionar a cuál velocidad será ejecutado cada nodo del grafo, para realizar esto, se calcula la utilización de cada tarea a cada velocidad, tal que la suma de las utilizaciones a diferentes velocidades en el sistema sea maximizado, sin que el tiempo de ejecución de cada tarea exceda el mínimo común múltiplo definido, el cuál es necesario (pero no suficiente), para garantizar la condición de planificabilidad

Cuando la utilización del sistema sea igual al 100% del sistema y de todas las posibles combinaciones, la velocidad asignada para cada tarea se obtenga un máximo ahorro de energía, se define en esta tesis que como el valor **óptimo** para el problema planteado.

3.1.1 Solución.

Para solucionar el problema planteado en el inciso anterior, lo resolveremos en dos pasos, los cuales describimos a continuación:

1.- Algoritmo Inicio: Lo primero que se deberá verificar es la planificabilidad del conjunto de tareas que arriban al sistema. La condición será que a la máxima velocidad, la utilización del conjunto de tareas deberá ser menor al 100% de la utilización del procesador. Enseguida se calcula la utilización de cada tarea a diferentes velocidades discretas.
2.- Algoritmo PD-N: El problema de seleccionar a qué velocidad se ejecutará cada tarea, se analizará en su forma inicial como un proceso de decisiones de N etapas.

3.1.2 Algoritmo inicio.

En el algoritmo inicio se describe el algoritmo que se ejecutará cuando arribe un DAG al sistema, este algoritmo puede ser resuelto en O(n). Además de la prueba de planificabilidad se calculan los nuevos valores de la utilización de cada tarea ejecutándose a diferentes velocidades.

1: **Algoritmo inicio**

2: **entrada**: un conjunto de nodos $V_i(i = 1,...,n)$) desde el DAG

3: **salida**: U_{ij}. La utilización de cada tarea a diferentes velocidades discretas.

4: **if** (verifica la planificabilidad) **then**

5: continua;

6: **else**

7: termina programa;

8: **end if**

9: **for** i=1 hasta n

10: **begin**

11: **for** j=1 hasta $N_{i,j}$

12: **begin**

13: $t_i = \frac{C_i}{f_j}$; calcula el tiempo de ejecución de cada tarea.

14: $U_{i,j} = \frac{t_i}{P_i}$; se obtiene la utilización a cada velocidad

15: **end**

16: **end**

3.1.3 Algoritmo PD-N.

En el algoritmo inicio, cada columna está compuesta por las utilizaciones de cada tarea a cada velocidad, es decir, la columna de U_{11} hasta U_{n1} corresponde al conjunto de utilizaciones de la tarea T_1. U_{12} hasta U_{n2} corresponde al conjunto de utilizaciones de la tarea T_2 y así sucesivamente hasta la columna n donde desde U_{1n} hasta U_{nn} corresponde a la n-esima tarea.

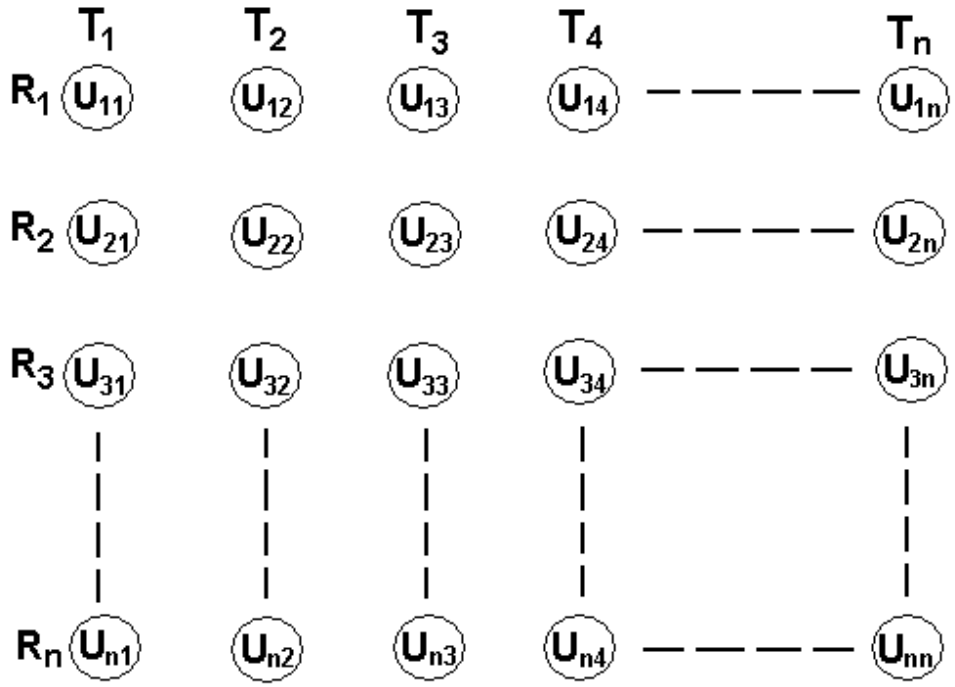


Figura 3.1. Diagrama de estados de N etapas.

Por otro lado las filas representan el conjunto de velocidades del procesador, es decir, U_{11} representa la utilización de la tarea T_1 ejecutándose a la velocidad R_1 del procesador, U_{21} corresponde a la utilización de la tarea T_1 ejecutándose a la velocidad R_2 del procesador y así sucesivamente con las demás utilizaciones. Para resolver este problema de encontrar a qué velocidad deberá ejecutarse cada tarea se puede escoger una ruta cualquiera, por ejemplo, una posible solución seria $U_{11}+U_{32}+U_{43}+...+U_{5n}$, otra posible ruta podría ser $U_{41}+U_{42}+U_{33}+...+E_{4n}$, la cual pudiera ser otra solución. La restricción es que la suma de las utilizaciones sea menor o igual al 100% de la carga del procesador y que se obtenga el mayor ahorro de energía posible.

Una forma de resolver este problema es analizar todas las posibles trayectorias, por ejemplo, si tuviéramos 6 tareas corriendo a 5 velocidades discretas, se necesitarían 5*5*5*5*5*5 = 15625 posibles rutas o soluciones de búsqueda, lo cual

implica un desarrollo computacional muy costoso. Para ejemplificar el desarrollo del algoritmo vamos a considerar que el número de tareas es igual a 6 y el número de velocidades es igual a 5, como se puede apreciar en la figura 3.2.

Se define como variable de decisión X_n donde n es el número de tareas. La variable representa la utilización que es seleccionada o por la que se decide, para cada tarea. Aplicando esta idea, en el caso que se desea escoger las utilizaciones a ejecutar de $U_{11,}U_{22},U_{23},U_{34},U_{45},U_{26}$ la ruta seleccionada seria: $U_{11} \rightarrow X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$; X_1 se escoge entre las utilizaciones $U_{12,}U_{22},U_{32},U_{42},U_{52}$; X_2 se escoge entre las utilizaciones $U_{13,}U_{23},U_{33},U_{43},U_{53}$; X_3 se escoge entre las utilizaciones $U_{14,}U_{24},U_{34},U_{44},U_{54}$; X_4 se escoge entre las utilizaciones $U_{15,}U_{25},U_{35},U_{45},U_{55}$; y X_5 se escoge entre las utilizaciones $U_{16,}U_{26},U_{36},U_{46},U_{56}$; En este ejemplo, los estados son las utilizaciones y las etapas corresponden a las rutas que hay que seleccionar para obtener un ahorro en la energía.

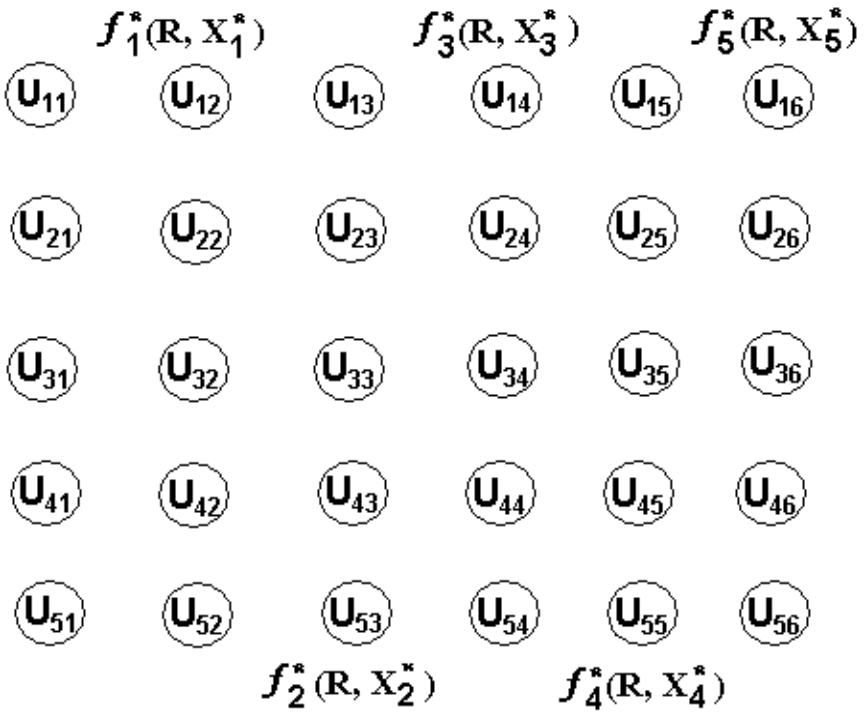


Figura 3.2. Ejemplo del diagrama de estados.

La función $f_n(R, X_n)$ se emplea para representar la utilización total al realizar la mejor selección para la última de n etapas, sabiendo que se está en el estado R y se decide por X_n como la próxima utilización a la que hay que ir. Dado un estado, o sea la utilización R cualquiera y la etapa n del recorrido, se define X_n^* como valor de X_n que minimiza la función $f_n(R, X_n)$. Sea $f_n^*(R)$ el valor mínimo correspondiente de la función; es decir, que $f_n^*(R) = f_n^*(R, X_n^*)$, en donde R va a ser igual a 1.0 (carga total del procesador, 100%), menos U_{11} hasta U_{n1}, este valor representa la holgura del sistema (X_i : $X_i = d_i - a_i - C_i$,) y tiene que ser distribuido entre las demás tareas, por lo que este valor es dividido entre el número de tareas restantes. Una vez encontrado este valor se busca el que más se le acerque de entre las utilizaciones de la tarea T_2. Se suma $U_{11} + f_1^*(R, X_1^*)$ y se vuelve a repetir

este paso hasta que se completa el número de tareas del sistema, en forma general puede ser representado por la siguiente ecuación:

$$R = \frac{1 - \sum_{I=1}^{n} U_i}{\sum_{i=0}^{n-1} \left((n-1) - i \right)} \tag{1}$$

De acuerdo a la nomenclatura anterior y considerando la figura 3.2, se deberá evaluar fila por fila hasta que se encuentre el óptimo o en su defecto el que se aproxime más al óptimo, es decir, se deberá evaluar en primera instancia $f_1^*(R)$, donde R es alguno de los estados $U_{12,}U_{22},U_{32},U_{42},U_{52}$, luego $f_2^*(R)$ donde R es alguno de los estados $U_{13,}U_{23},U_{33},U_{43},U_{53}$, en seguida $f_3^*(R)$ donde R es alguno de los estados $U_{14,}U_{24},U_{34},U_{44},U_{54}$, posteriormente $f_4^*(R)$ es $U_{15,}U_{25},U_{35},U_{45},U_{55}$, y finalmente $f_5^*(R)$ donde R es $U_{16,}U_{26},U_{36},U_{46},U_{56}$, La ejecución de este procedimiento se puede ver en el algoritmo PD-N. En la línea 12 se escoge el máximo dentro del rango de dos utilizaciones, pero se puede buscar el valor mínimo o el valor promedio para seleccionar esa utilización, por lo que se proponen tres diferentes métodos (MAX, MIN y PROM), es decir, si tenemos que el valor de temp=0.45 y se debe de decidir entre el rango de utilizaciones de 0.3 y 0.70, si se desea el máximo se escoge el valor 0.7, si se desea el mínimo se debe escoger 0.3 y si se desea el promedio se selecciona la que esté más cerca, en este caso seria 0.3.

1: **Algoritmo PD-N**

2: **entrada**: Diagrama de estados de N etapas.

3: **salida**: Máximo ahorro de energía.

4: para todo $U[][] = diagrama\ de\ estados; U_{tot}[] = rutas\ optimas;$

5: llama a Inicio ; llama al algoritmo Inicio

6: **for** k=1 hasta v ;primera fila pivote

7: **begin**
8: **for** i=2 hasta n ;va de la segunda columna hasta n
9: **begin**
10: temp = I / ((n-1)-(i-2)); holgura actual del sistema
11: **for** j=1 hasta v ;recorrer toda la columna
12: **begin**
13: **if** (U[j][i] <= temp) **then**
14: $U_{tot}[k] = U_{tot}[k] + U[j][i]$;
15: I = I – U[j][i];
16: **termina el for de la linea 10**
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: **se verifica que se cumpla la siguiente expresión:** $\sum U_i \leq 100\%$
22: **se obtiene el ahorro de energía de cada ruta encontrada**
23: **se encuentra el máximo ahorro de energía**

3.1.4 Ejemplo del algoritmo.

3.1.4.1 Inicio

Para ilustrar la ejecución del algoritmo inicio y el algoritmo PD-N, consideramos el modelo de tareas que se presenta en la figura 3.3. En la tabla 3.1 se describe la carga de trabajo de tiempo-real, la cuál fue calculada asumiendo el máximo nivel de velocidad para cada tarea (condición inicial para todo el DAG), obteniendo una carga total de $\sum_i U_i = 60\%$. La función para el consumo de potencia a utilizar es:

$P_d = k * f^3$, donde k es una constante, para el consumo de energía se calcula con la siguiente ecuación calculado por $E = P_d * I$ donde I es el intervalo de tiempo.

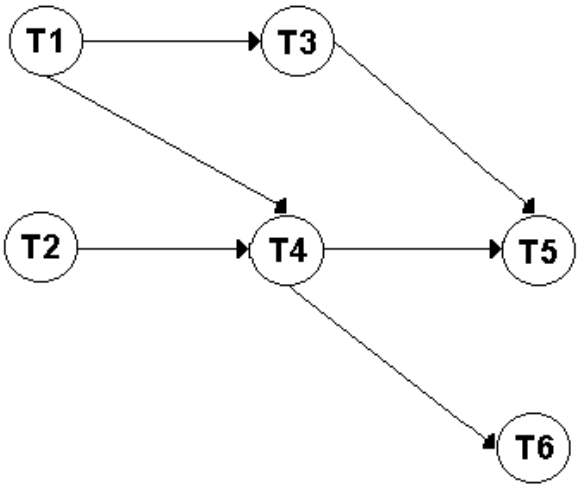


Figura 3.3. Conjunto de tareas (DAG).

Tabla 3.1. Restricciones temporales del DAG.

Tareas	C_i	P_i	U_i	K_i
Tarea 1	1	10	0.10	1
Tarea 2	4	50	0.08	1
Tarea 3	3	50	0.06	2
Tarea 4	3	20	0.15	8
Tarea 5	3	25	0.12	8
Tarea 6	9	100	0.09	4

La tabla 3.2 se muestra la utilización U_{ij} y el consumo de potencia E_{ij} del conjunto de velocidades $N_{ij} = \{1.0, 0.8, 0.6, 0.4, 0.15\}$ para todas las tareas, se calculó la utilización de cada tarea a diferentes velocidades, eso quiere decir, que para calcular la utilización de la primera tarea a la primera velocidad es como sigue, $U_{11} = U_1 / f_1 = 0.10 / 1.0 = 0.10$, la segunda velocidad es, $U_{12} = 0.10 / 0.8 = 0.12$, la tercera velocidad es, $U_{13} = 0.10 / 0.6 = 0.16$ y así sucesivamente para todas las demás tareas.

Tabla 3.2. U_{ij} y E_{ij} a diferentes velocidades.

Tareas	**Niveles de Velocidad**				
	1.0	**0.8**	**0.6**	**0.4**	**0.15**
	U_{ij} | E_{ij}	U_{ij} | E_{ij}	U_{ij} | E_{ij}	U_{ij} | E_{ij}	U_{ij} | E_{ij}
Tarea 1	0.10 | 10	0.12 | 6.4	0.16 | 3.6	0.25 |1.6	0.66 | 0.22
Tarea 2	0.15 | 8	0.18 | 5.12	0.25 | 2.88	0.37 | 1.28	1.00 | 0.18
Tarea 3	0.12 | 12	0.15 | 7.68	0.20 | 4.32	0.30 | 1.92	0.80 | 0.27
Tarea 4	0.08 | 120	0.10 | 76.8	0.13 | 43.2	0.20 | 19.2	0.53 | 2.7
Tarea 5	0.06 | 96	0.07 | 61.44	0.10 | 34.5	0.15 | 15.36	0.40 | 2.16
Tarea 6	0.09 | 36	0.11 | 23.04	0.15 |12.96	0.22 | 5.76	0.60 | 0.81

En la misma tabla 3.2 se muestra el consumo de potencia para cada tarea a diferentes velocidades el cuál es calculado como sigue, se considera el intervalo de tiempo del conjunto de tareas de la tabla 3.1, $I = 100(LCM)$, el tiempo de ejecución a la máxima velocidad, $t_i = C_i / 1$ y el periodo de la tarea 1, $P_1 = 10$, el número de instancias de la tarea T_1 en I es igual a 100/10=10. El consumo de energía de la tarea T_1 es $E_1 = k_1 * f_1^3 * (C_1 / f_1) * I = 1*1^3 *1*10 = 10$, para la tarea 2 el consumo es: $E_2 = 1*1*4*2 = 8$ y así hasta la tarea 6. Para los demás niveles de potencia se realiza como sigue: $E_{12} = E_1 * f^2 = 10*(0.8)^2 = 6.4$, $E_{13} = 10*(0.6)^2 = 3.6$, etc.

3.1.4.2 Desarrollo.

En la figura 3.4 se presentan en un diagrama de estados todas las utilizaciones de todas las tareas, en cada columna tenemos las utilizaciones correspondientes de cada tarea y ordenadas de mayor a menor. La secuencia del orden de las tareas está determinada por las restricciones de precedencia.

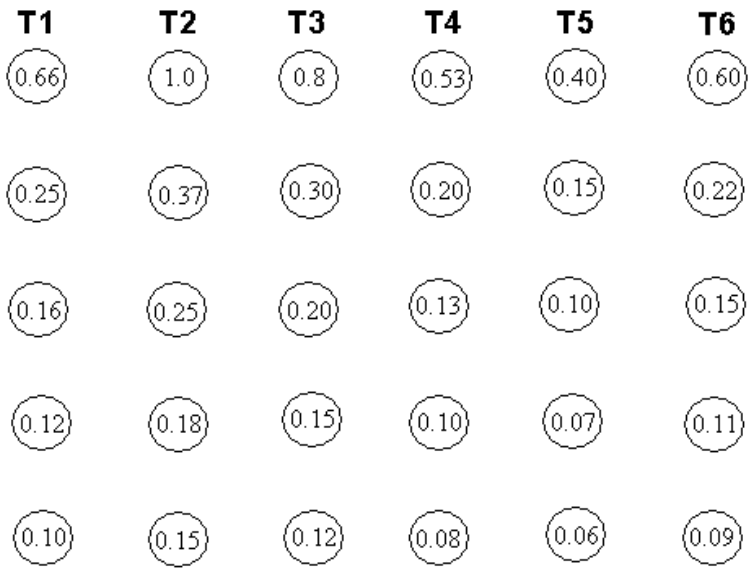


Figura 3.4. Utilizaciones por estados.

Al ejecutar el algoritmo PD-N, obtenemos tres diferentes resultados de los tres métodos propuestos (Max, Prom, Min) los cuales son descritos en la tabla 3.3. Se observa que se cumple la condición de $\sum U_{i,j} \leq 100\%$. También se observa en la última columna el porcentaje de ahorro de energía que se obtiene al seleccionar cada ruta para cada método. La segunda ruta del método Max se obtiene el mayor porcentaje de ahorro de energía que corresponde a 37.58 % y para el segundo método la segunda ruta del método Prom se obtiene 34.04 % de ahorro.

Para obtener el porcentaje de ahorro se tomó como referencia el algoritmo SC (Static Continuous) propuesto en [5], la velocidad del procesador es seleccionada en base a la utilización total del sistema (es decir, $f_i = \sum U_{i1}$). La solución discreta estática usa la solución continua (SC) y aproxima sus resultados. Por ejemplo, si tenemos el siguiente conjunto de velocidades N = {1.0, 0.75, 0.5, 0.35, 0.15} y la utilización total del conjunto de tareas es del 60% (0.6) entonces se escoge la velocidad discreta de 0.75, así para este ejemplo el valor de SD = 3.6 +

2.88 + 4.32 + 43.2 + 34.56 + 12.96 = 101.52, tomando en cuenta el consumo de energía.

Tabla 3.3. Resultados con los métodos.

Método	**RUTA**	**U_{ij}**	**% de S_{ij}**
MAX	a) 0.16+0.18+0.20+0.20+0.15+0.11 = 1.0		30.41 %
	b) 0.12+0.18+0.20+0.20+0.15+0.15 = 1.0		37.58 %
PROM	a) 0.25+0.15+0.12+0.13+0.15+0.15 = 0.98		12.52 %
	b) 0.10+0.18+0.20+0.20+0.15+0.15 = 0.98		34.04 %
MIN	a) 0.25+0.15+0.12+0.13+0.15+0.15 = 0.95		8.27 %
	b) 0.16+0.15+0.15+0.13+0.15+0.22 = 0.96		17.65 %
	c) 0.12+0.15+0.15+0.13+0.15+0.15 = 0.95		7.80 %
	d) 0.10+0.18+0.15+0.13+0.15+0.22 = 0.93		14.18 %

El mayor ahorro de energía obtenido en este ejemplo es de 37.58%.

3.2.- Programación lineal.

El problema puede ser formulado como el problema de la mochila. La generalización de este problema es sabido que tiene una complejidad NP-hard [57], esto quiere decir que es muy difícil encontrar un método exacto en un tiempo polinomial para obtener su solución. Desde un punto de vista práctico esto significa que la solución requiere una solución heurística que sea eficiente y de bajo costo computacional. Esto es, que la solución debería estar dentro del orden de los milisegundos. Dado un conjunto de tareas de tiempo real periódicas éstas serán empacadas dentro de un Knapsack de capacidad c. Cada tarea T_i tiene asociado un tiempo de ejecución a una velocidad constante t_i, una utilización U_i y una acotación m_i. El problema es seleccionar la velocidad de ejecución $x_i (0 \le x_i \le m_i)$ de cada tarea tal que la suma de los tiempos de ejecución de las tareas seleccionadas sea maximizada sin que la suma de las utilizaciones exceda

c. El problema Knapsack acotado (BKP, Bounded Knapsack Problem) es definida como el siguiente problema de optimización:

$$\max \quad z = \sum_{j=1}^{n} t_j x_j$$

Sujeto a : (2)

$$x_{p_j} \geq x_i, \quad j = 1,...,n$$

$$\sum_{j=1}^{n} U_j x_j \leq c$$

$$x_j \in \{0,1,...,m_j\}, \quad j = 1,...,n$$

Donde todos los coeficientes son enteros positivos. Sin pérdida de generalidad se asume que $m_j U_j \leq c$ para $j = 1,...,n$ y así para todas las tareas disponibles dentro del Knapsack y que $\sum_{j=1}^{n} m_j U_j > c$ para asegurar un problema que no es trivial. Para resolver el problema BKP existen dos soluciones: 1) La solución al problema (2) se le llamará algoritmo KP-B (Knapsack Problem Bounded). 2) Si se relaja la restricción de integridad $x_j \in \{0,1,...,m_j\}$ en (2) a una restricción lineal $0 \leq x_j \leq m_j$, se obtiene un problema Knapsack lineal acotado (LBKP, Linear Bounded Knapsack Problem). Si $m_j = 1$ para todas las variables, se obtiene el problema Knapsack 0-1.

3.2.1 Problema LBKP.

El problema Knapsack acotado lineal se resuelve fácilmente utilizando el algoritmo greedy: se ordena las tareas en un orden no creciente tal que

$$e_i \geq e_j \quad si \quad i < j$$

Y se define un elemento de ruptura tipo b como:

$$b = \min\left\{ j : \sum_{i=1}^{j} m_j U_j > c \right\} \tag{3}$$

Entonces una solución óptima al LBKP es definida como $x_j = m_j \quad para \quad j = 1,...,b-1$ y $x_j = 0 \quad para \quad j = b+1,...,n$ mientras es seleccionado:

$$x_b = \frac{\left(c - \sum^{b-1} m_j U_j\right) P_b}{U_b} \tag{4}$$

El elemento de ruptura puede ser encontrado en un tiempo de $O(n)$, adoptando la técnica de [8]. Se trunca la solución óptima LP para $x_b = 0$ obteniendo la solución de ruptura x', el cual tiene la suma del tiempo de ejecución $\bar{t} = \sum_{j=1}^{b=1} m_j t_i$, y la suma de la utilización $\bar{U} = \sum_{j=1}^{b-1} m_j U_j$.

3.2.2 Solución al KP-B.

Se define $J_i(c)$ como la solución óptima al siguiente subproblema del KP-B, definida para la primera variable i del problema (2).

$$J_i(c) = \max\left\{ \sum_{j=1}^{i} t_j x_j : \sum_{j=1}^{i} U_j x_j \le c,\ x_j \in \{0,...,m_j\} para\ j = 1,...,i \right\} \tag{5}$$

Generalizando los resultados por Bellman [10] para el problema Knapsack 0-1, se obtiene la siguiente fórmula para la solución del problema BKP

$$J_i(c) = \max\begin{cases} J_{i-1}(c) \\ J_{i-1}(c - U_i) + p_i \\ . \\ . \\ . \\ J_{i-1}(c - m_i Ui) + m_i p_i \end{cases} \tag{6}$$

Y se selecciona

$$J_0(c) = 0 \ \ para\ todo\ c \geq 0$$

Una solución más eficiente deberá ser tomada en cuenta, ya que solo pocas tareas alrededor de b necesitan ser cambiadas de los valores óptimos de LP a valores óptimos IP. Asumiendo que las tareas son ordenadas acorde a su tiempo de ejecución no creciente y $J_{s,t}(c)(s \leq b \leq t)$ será la solución óptima al problema (2).

$$J_{s,t}(c) = \max\left\{\begin{array}{l} \sum_{j=1}^{s-1} m_j p_j + \sum_{j=s}^{t} p_j x_j; \\ \sum_{j=1}^{s-1} m_j U_j + \sum_{j=s}^{t} U_j x_j \leq c, \\ x_j \in \{0,...,m_j\}\ para\ \ j = s,...t \end{array}\right\} \tag{7}$$

Esto conlleva a la siguiente solución mejorada:

$$J_{s,t}(c) = \max\left\{\begin{array}{ll} J_{s,t-1}(c) & si\ t \geq b \\ J_{s,t-1}(c - U_t) + p_t & si\ t \geq b \\ \vdots & \\ J_{s,t-1}(c - m_t U_t) + m_t p_t & si\ t \geq b \\ J_{s+1,t}(c) & si\ s < b \\ J_{s+1,t}(c + U_s) - p_s & si\ s < b \\ \vdots & \\ J_{s+1,t}(c + m_s U_s) - m_s p_s & si\ s < b \end{array}\right. \tag{8}$$

Mientras es seleccionado:

$$J_{b,b-1}(c) = \sum_{j=1}^{b-1} m_j p_j \ \ para\ todo\ c \geq \sum_{j=1}^{b-1} m_j U_j$$

Así la enumeración comienza en $(s,t) = (b, b-1)$ y continua la tarea s del Knapsack o insertando las tareas en el Knapsack. Una solución óptima al KP-B es cuando se encuentra $J_1, n(c)$.

3.2.3. Ejemplo del algoritmo.

3.2.3.1 Inicio.

El inicio del ejemplo del algoritmo se considera las mismas condiciones al presentado en el inciso 3.1.4.1.

3.2.3.2 Desarrollo.

En la tabla 3.4 se puede apreciar cómo se vería el registro en la primera etapa, la utilización total es 3.99, la cual es más grande que 1.00. En este paso se verifica si la condición de parada es verdadera, para el caso que no se cumpla, se continua con la ejecución del algoritmo, a la variable temp es asignada la utilización de la tarea 1, y será comparada con la utilización de las otras tareas.

Se busca en el registro el valor de la utilización más alto, salvándolo en la variable k el número de la tarea que corresponde. Es asignado el valor de la utilizaron inferior inmediato de la tarea seleccionada, el registro es mostrado en la tabla 3.5, correspondiendo a la segunda etapa. La utilización total es 3.36 que será verificada si la condición de parada es cumplida.

Tabla 3.4. Primer paso.

0.66	0.53	0.40	1.00	0.80	0.60
Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6

Tabla 3.5. Segundo paso.

0.66	0.53	0.40	0.37	0.80	0.60
Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5	Tarea 6

En este mismo camino se sigue la ejecución del algoritmo hasta el **paso trece** (para este ejemplo) donde la utilización total es igual a 0.97 el cual es menor a 1.0, el algoritmo termina y es obtenido para este caso la velocidad óptima, eso quiere decir, la velocidad que las tareas deberán ser ejecutadas, éstas son mostradas en la tabla 3,6. También se obtiene el porcentaje del ahorro de energía el cual es

16.70%. Notar que una vez encontrada la velocidad de ejecución de las tareas, el DAG será ejecutado siguiendo el orden de precedencia de las tareas.

Tabla 3.6. Último paso.

Tareas	1	2	3	4	5	6
Uij	0.16	0.13	0.15	0.18	0.20	0.15
Nij	0.6	0.6	0.4	0.8	0.6	0.6

3.3.- Problema especifico.

Una introducción al procesamiento de imágenes utilizando la transformada wavelet se puede consultar en el apéndice D. El problema puede formularse como sigue. Cada vez que una imagen arribe o salga del sistema, el objetivo es determinar el modo (la velocidad) de ejecución en la transformada wavelet de las diferentes sub-bandas tal que las diferentes sub-bandas no pierdan sus plazos de respuesta y el ahorro de energía en el sistema sea maximizado. Cada imagen en el sistema se ejecuta en un procesador de voltaje variable. Notar que una solución a este problema deberá calcularse cada vez que una nueva imagen arribe o salga del sistema.

El problema es formulado como sigue:

$$\max_{\{u(k)\}_{k=0}^{2}} J = \sum_{k=0}^{2} S_k[u(k)]$$

sujeto a $x(k+1) = x(k) - u(k)$

con:
$$x(0) = 1.0,$$
$$x(3) = 0,$$
$$u(k) \le x(k),$$
$$u(k) \in \{1.0, 0.8, 0.6, 0.4, 0.15\} \ \textit{for}\ k = 0,1,2$$

donde

$k = correponde\ a\ las\ sub-bandas$
$S_k[u(k)] = ahorro\ de\ energía$
$x(k) = \text{var}\, iable\ de\ estado$
$u(k) = \text{var}\, iable\ de\ control, (selecciona\ la\ velocidad)$

Es utilizado el principio de optimización de Bellman para calcular el estado en cada etapa, como sigue:

Paso 1: $x(3)$ es calculada como sigue:

$$J_3^*\{x(3)\} = 0$$

Paso 2: $x(2) \in \{1.0, 0.8, 0.6, 0.4, 0.15\}$ es:

$$J_2^*\{x(2)\} = \max_{u(2)}\{S_2[u(2) + J_3^*\{x(2) - u(2)\}\},$$

donde:

$$x(3) = 0 = x(2) - u(2), u(2) \le x(2), u(2) \in \{0.15,0.4,0.6,0.8,1.0\}$$

Paso 3: $x(1) \in \{1.0, 0.8, 0.6, 0.4, 0.15\}$, la ecuación correspondiente es:

$$J_1^*\{x(1)\} = \max_{u(1)}\{S_1[u(1) + J_2^*\{x(1) - u(1)\}\},$$

donde:

$$u(1) \le x(1), u(1) \in \{0.15,0.4,0.6,0.8,1.0\}$$

Paso 4: $x(0) = 1.0$ es:

$$J_0^*\{1.0\} = \max_{u(0)}\{S_0[u(0) + J_1^*\{1.0 - u(0)\}\},$$

donde:

$$u(0) \le 5, u(0) \in \{0.15,0.4,0.6,0.8,1.0\}$$

3.3.1. Ejemplo del algoritmo

Para ilustrar la ejecución del algoritmo, se considera la imagen Lena y utilizamos cinco niveles de velocidades discretas $\{1.0, 0.8, 0.6, 0.4, 0.15\}$, en un procesador de voltaje variable. La utilización del CPU puede ser medida mientras el sistema esté bajo varios estados de carga. Obviamente no hay un camino directo para medir directamente la utilización del CPU. La utilización puede ser derivada por los cambios en el período de un lazo ejecutándose en segundo plano. El período del lazo de segundo plano promedio deberá ser medido bajo varias cargas del sistema y posteriormente puede ser obtenida la utilización del CPU. La utilización del CPU se define como el tiempo 'no' gastado en ejecutar una tarea desocupada u ociosa (es una tarea con una prioridad muy baja en un sistema de multiprocesamiento). La cantidad de tiempo gastado en ejecutar la tarea ociosa puede ser representada como la razón entre el período de la tarea ociosa en CPU sin carga y el periodo de la tarea ociosa bajo algún sistema con carga conocido.

% tiempo de la tarea ociosa = (Período promedio de la tarea ociosa sin carga) * 100% /
(Período promedio de la tarea ociosa con carga) (9)

Basado en la técnica de DVS, se ajusta la velocidad del procesador para cada sub-banda con el reclamo de holgura.

En la tabla 3.7 se presenta el tiempo en microsegundos para cada sub-banda de la imagen. Son considerados 5 niveles. El tiempo de ejecución promedio de la tarea ociosa es 195 microsegundos, este tiempo fue calculado igual al presentado en [68] donde se utilizó un Analizador de Estado Lógico para medir los datos que fluyen por el bus de datos y de direcciones, al estar ejecutando una tarea en segundo plano. Este tiempo obtenido es el periodo promedio de la tarea en segundo plano sin carga. En seguida se obtiene el tiempo de la tarea ociosa con carga, o sea, al ejecutar cada sub-banda que se observa en la segunda columna de la tabla 3.7 y finalmente de la ecuación 1 se obtiene la tercera columna. La

cuarta columna es obtenida de restar 100 menos la tercera columna para obtener la utilización del CPU.

Tabla 3.7. Utilización por sub-bandas.

Sub-bandas	T (microsegundos)	% Idle	% CPU
1	974	20.02	79.97
2	535	36.49	63.55
3	388	50.26	49.74
4	859	22.70	77.30
5	417	46.76	53.24
6	353	55.24	44.76
7	595	32.77	67.23
8	368	52.99	47.01
9	287	67.94	32.06
10	445	43.82	56.18
11	313	62.30	37.70
12	267	73.03	26.96
13	342	57.02	42.98
14	281	69.39	30.61
15	247	78.95	21.05

Después de ejecutar la primera sub-banda del nivel 1, a la velocidad máxima, podemos obtener el porcentaje de utilización de esta sub-banda, el tiempo de holgura entre la sub-banda 1 y la sub-banda 2 es el porcentaje de ahorro de energía, este porcentaje de holgura también se obtiene entre la sub-banda 2 y la sub-banda 3, la velocidad de ejecución de estas sub-bandas depende del resultado de la primera sub-banda. Este mismo proceso se aplica a todos los niveles de la transformada, calculando el porcentaje de utilización para la primera sub-banda de cada nivel y asignando el porcentaje de holgura a la siguiente sub-banda. En la tabla 3.8 es presentado el porcentaje del ahorro de energía por nivel. El ahorro de energía promedio total de toda la imagen es 44.19%.

Tabla 3.8. Ahorro de energía por nivel.

Level	Sub-band	% Energy Saving	Total
1	1	0	
	2	20.02	
	3	20.02	= 37.50 %
2	4	0	
	5	22.70	
	6	22.70	= 36.40 %
3	7	0	
	8	32.77	
	9	32.77	= 42.64 %
4	10	0	
	11	43.82	
	12	43.82	= 48.91%
5	13	0	
	14	57.02	
	15	57.02	= 55.53%

El ahorro de energía por nivel de la tercera columna de la tabla 3.8 fue el que se obtuvo por sub-banda. En la cuarta columna se calculó el porcentaje del ahorro de energía total por nivel teniendo en cuenta el máximo ahorro que se podía obtener por nivel con respecto al obtenido de la tercera columna. Por ejemplo, para el primer nivel, el máximo ahorro que se pudiera obtener seria de 20.02 para la primera sub-banda, 36.49 para la segunda sub-banda y 50.26 para la tercera sub-banda, sumando un máximo total de 106.77. El ahorro total obtenido por sub-banda fue de 40.04, el porcentaje de ahorro energía se obtiene de una regla de tres y es para el primer nivel de 37.50%. Y así para los siguientes niveles.

RESUMEN

En este capítulo se formuló el problema para cada uno de los objetivos específicos definidos en el capítulo 1. El primer planteamiento se maximizó la suma de las utilizaciones durante un hiperperíodo, este problema se resolvió mediante un proceso de decisiones de N etapas. En el segundo planteamiento se maximizó el tiempo de ejecución a una velocidad constante y fue presentado como un problema de programación entera. Y finalmente el tercer planteamiento fue realizado proponiendo un nuevo método para ahorrar energía al realizar la transformada wavelet.

CAPITULO 4

RESULTADOS

En este capítulo son presentados los resultados que se obtuvieron al implementar los algoritmos presentados en el capítulo 3. Explicando cómo se realizó la simulación para cada caso y los resultados obtenidos para cada objetivo específico.

Se tomó como referencia el algoritmo óptimo continuo propuesto en [5] como algoritmo de comparación, en donde la velocidad del procesador es seleccionada en base a la utilización total del sistema (es decir $f_i = \sum U_{i1}$). El algoritmo óptimo discreto usa el algoritmo óptimo continuo y aproxima sus resultados. Por ejemplo, si tenemos el siguiente conjunto de velocidades N={1.0, 0.75, 0.5, 0.35, 0.15} y la utilización total del conjunto de tareas es del 60% (0.6) entonces se escoge la velocidad discreta de 0.75.

4.1 Resultado del apartado 3.1.

Para los experimentos de simulación se utilizó el modelo de velocidad/voltaje discreto de [76], para verificar el algoritmo PD-N y constatar que los resultados estén cercanos al esperado. Se utilizó un conjunto de tareas constituido por un DAG. Los objetivos en esta simulación son los siguientes: a) comparar el porcentaje del ahorro de energía al variar tanto la carga del procesador, así como el número de nodos que forma el DAG. b) medir los tiempos de ejecución para comprobar que los tres métodos estén dentro del rango aceptable (milisegundos) y c) variar el número de velocidades discretas para observar un cambio significativo en el desempeño del algoritmo PD-N.

En la figura 4.1 y la figura 4.2 se representan los resultados obtenidos al generar 3000 conjuntos de tareas, cada ciclo de 3000 conjuntos de tareas se promedian y se obtiene un solo resultado, en el conjunto de ciclos varía la carga del procesador que va de 0.2% a 0.9%. En la figura 4.1 se utilizaron 5 niveles de velocidad discretas y un número de nodos que van de 6 a 16. En la figura 4.2 se dejó fijo el número de nodos a 6 y se varió el número de velocidades discretas de 4 a 16. Al término del conjunto de ciclos se obtuvieron los tiempos de ejecución. En la figura 4.3 se presenta el tiempo que tardó el algoritmo PD-N, el cual permite medir el tiempo físico en microsegundos, usando una PC Intel PENTIUM IV a 3.2 GHz con 512MB de RAM y corriendo en un Sistema Operativo Linux. La función usada para la medición es psched_get_time(). El hiperperíodo varía dependiendo del conjunto de tareas generado. Para el primer nivel del DAG se sigue una distribución uniforme entre 2 y 5 nodos. Los nodos de los siguientes niveles también siguen una distribución uniforme entre 0 y 3 nodos hasta que se completa el número de nodos que va de 6 hasta 16. Los arcos entre los nodos son generados aleatoriamente.

El periodo P_1 de cada nodo en el grafo es generado por una función de distribución uniforme entre 10 y 100. El tiempo de ejecución de cada tarea T_1 se obtiene de una función de distribución uniforme entre 1 y 10, la utilización de cada tarea se obtiene de la siguiente expresión: $U_i = C_i / P_i$, la utilización total es la suma de todas las utilizaciones de los nodos que forman el DAG, con la restricción de que sea el 60% de carga del procesador.

En la figura 4.1 se observa que al variar la carga del procesador, el sistema tendrá menor holgura para ahorrar energía, por consiguiente; para una carga del procesador baja se tiene un alto ahorro de energía cercano del 75% y para una alta carga del procesador se tiene un bajo ahorro, para un 90% de carga se tiene un ahorro de 6%. El algoritmo se acerca al óptimo por lo que es una buena

aproximación. En esta gráfica se varió el número de nodos que conforman el DAG observándose que el comportamiento es muy parecido variando muy poco entre 0.3% y 0.6% de carga.

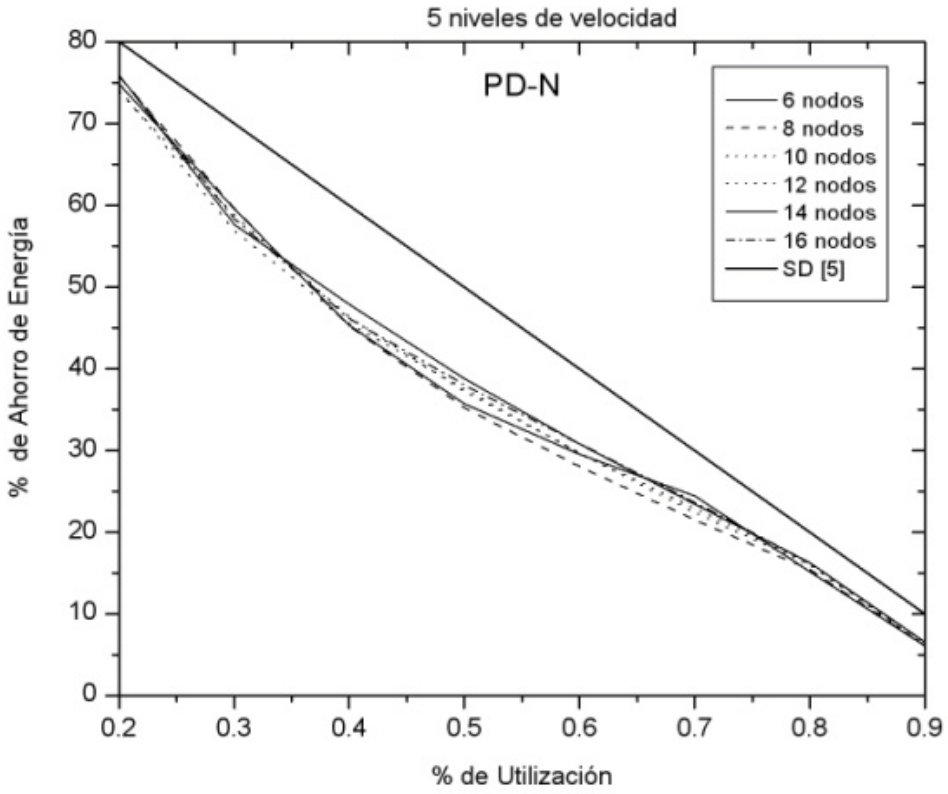


Figura 4.1. % de S_{ij} ahorro al variar el # de nodos.

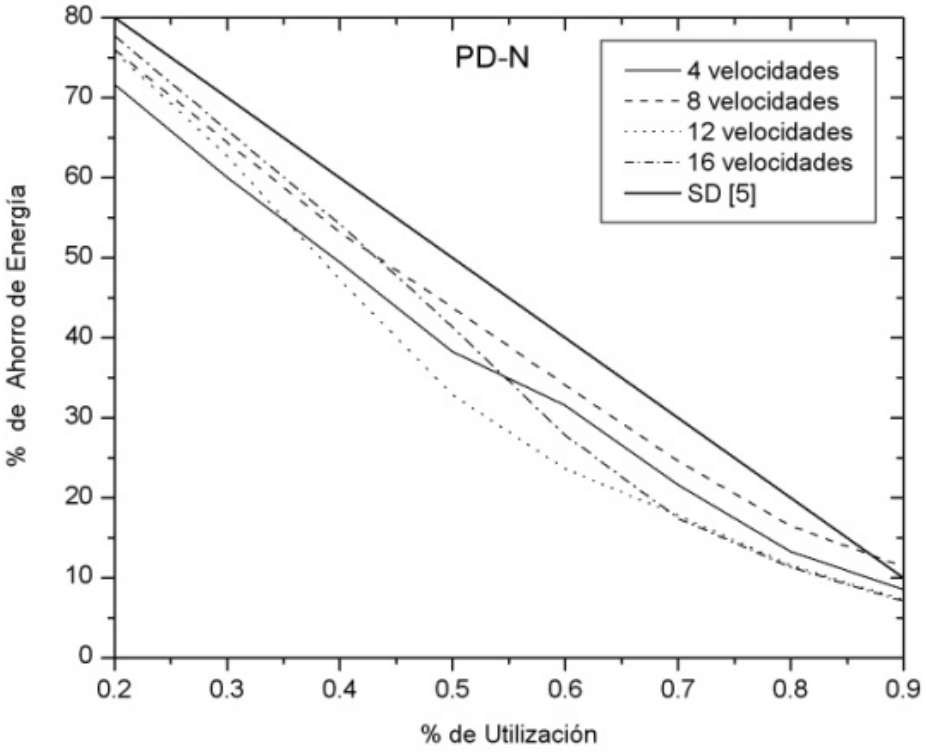


Figura 4.2. % de S_{ij} al variar las velocidades.

En la figura 4.2 el porcentaje de ahorro de energía sigue el mismo comportamiento al presentado en la figura 4.1, con la diferencia de que se abre un poco más la diferencia entre las diferentes velocidades, obteniendo un comportamiento mejor para 8 velocidades y un peor comportamiento para 12 velocidades.

El la figura 4.3 se utilizaron los tres métodos mencionados en el desarrollo del algoritmo PD-N, en donde si hay mucha diferencia para una carga de procesador menor al 50%, en donde el método Max ocupó en tiempo de ejecución mayor, mientras para el Min obtuvo un tiempo de ejecución menor. Después del 50% como se reduce la holgura los tres métodos tienden a igualar su tiempo de ejecución.

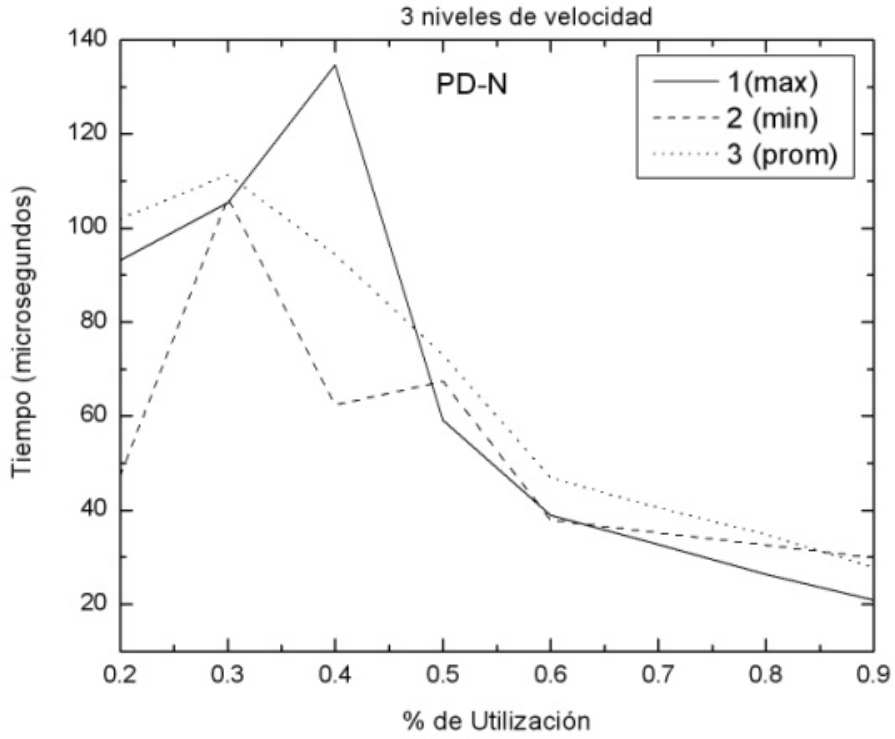


Figura 4.3. Tiempo de ejecución para los tres métodos.

4.2 Resultado del apartado 3.2.

Para los experimentos de simulación se utilizó el modelo de velocidad/voltaje discreto para verificar los algoritmos que se están proponiendo y constatar que los resultados estén cercanos a uno de la literatura y que se considera como óptimo. Los objetivos en esta simulación son los siguientes: a) comparar el porcentaje del ahorro de energía al variar la carga del procesador. b) comparar el porcentaje de ahorro de energía al variar el número de tareas. c) medir los tiempos de ejecución para comprobar que los dos algoritmos estén dentro del rango aceptable (milisegundos).

En las figura 4.4 y la figura 4.5 se representan los resultados obtenidos al generar 3000 conjuntos de tareas, cada ciclo de 3000 conjuntos de tareas se promedian y se obtiene un solo resultado, en el conjunto de ciclos varía la carga del procesador que va de 0.2% a 0.9%. En la figura 4.4 se utilizaron 5 niveles de velocidad discretos y un número de nodos que van de 10 a 100. En la figura 4.5 se deja fijo el número de tareas a 10 y manejaron 5 niveles de velocidad discretos, variando la carga del procesador del 20 al 100%. Al término del conjunto de ciclos se obtuvieron los tiempos de ejecución. En la figura 4.6 se presenta el tiempo que tardaron los algoritmos, el cual permite medir el tiempo físico en microsegundos, usando una PC Intel PENTIUM IV a 3.2 GHz con 512MB de RAM y corriendo en un Sistema Operativo Linux. La función usada para la medición es psched_get_time(). El hiperperíodo varía dependiendo del conjunto de tareas generado. Para el primer nivel del DAG se sigue una distribución uniforme entre 2 y 5 nodos. Los nodos de los siguientes niveles también siguen una distribución uniforme entre 0 y 3 nodos hasta que se completa el número de nodos que va de 10 hasta 100. Los arcos entre los nodos son generados aleatoriamente.

El periodo P_1 de cada nodo en el grafo es generado por una función de distribución uniforme entre 10 y 100. El tiempo de ejecución de cada tarea T_1 se obtiene de una función de distribución uniforme entre 1 y 10, la utilización de cada tarea se obtiene de la siguiente expresión: $U_i = C_i / P_i$, la utilización total es la suma de todas las utilizaciones de los nodos que forman el DAG, con la restricción de que sea el 60% de carga del procesador.

Se evalúan los dos algoritmos propuestos en este artículo, el PD-N (capítulo 3, inciso 3.1.3) y el KP-B (capítulo 3, inciso 3.2.2) comparándolos con otros tres algoritmos. El primero a comparar es el resultado de resolver el problema (2) del capítulo 3, utilizando Programación Dinámica (PD). El segundo a comparar es el LBKP que es otra forma de resolver el problema (2) del capítulo 3. Y finalmente se

compara con el algoritmo OPT-LU presentado en [5], en donde los autores consideran las tareas con diferentes características de potencia y entrega una solución continua. (Para la solución discreta aproximan la solución continua usando niveles discretos de velocidad).

Para la primera evaluación se consideraron cinco niveles discretos de velocidad y una carga de procesador del 60%, se varió el número de tareas de 10 a 100, observando el porcentaje del ahorro de energía (ver figura 4.4). El algoritmo que obtiene más ahorro de energía (en promedio 32% de ahorro) es el PD, ya que su búsqueda es más exhaustiva, el inconveniente es que ocupa muchos ciclos de reloj al realizar esa búsqueda. En segundo lugar se tiene el algoritmo LBKP el cual ocupa cierto tiempo para llegar al elemento de ruptura y tiene del orden del 28% de ahorro de energía. El algoritmo que le sigue con alrededor del 23% de ahorro es el PD-N. Los que obtienen menor ahorro de energía son el KP-B y el Opt-Lu. El Opt-Lu tiene un ahorro del 15% y el KP-B del 14% de ahorro.

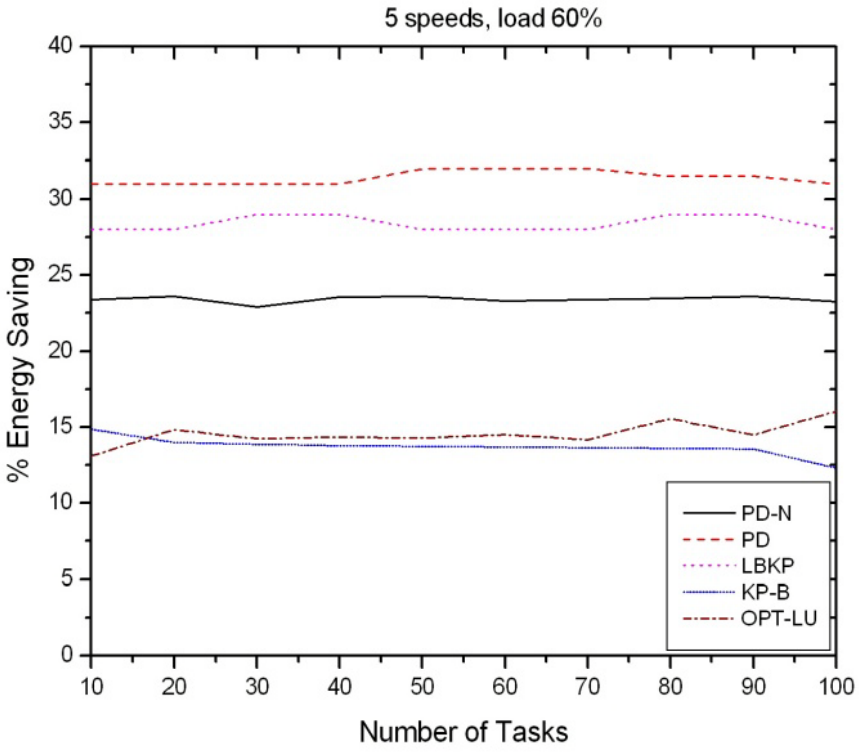


Figura 4.4. Ahorro de energía.

El siguiente experimento consistió en establecer 5 niveles de velocidad discretos para 10 tareas, variando el porcentaje de utilización del procesador del 20% al 100% (figura 4.5). Todos siguen el mismo comportamiento excepto el Opt-Lu el cual tiene un comportamiento más lineal, partiendo del 30% de ahorro de energía, pero llegando al 90%, comienza la caída lineal hacia 0. El algoritmo que tiene más ahorro de energía es el LBKP seguido por el KP-B. El peor es el PD ya que al 80% de carga, el ahorro de energía es cero.

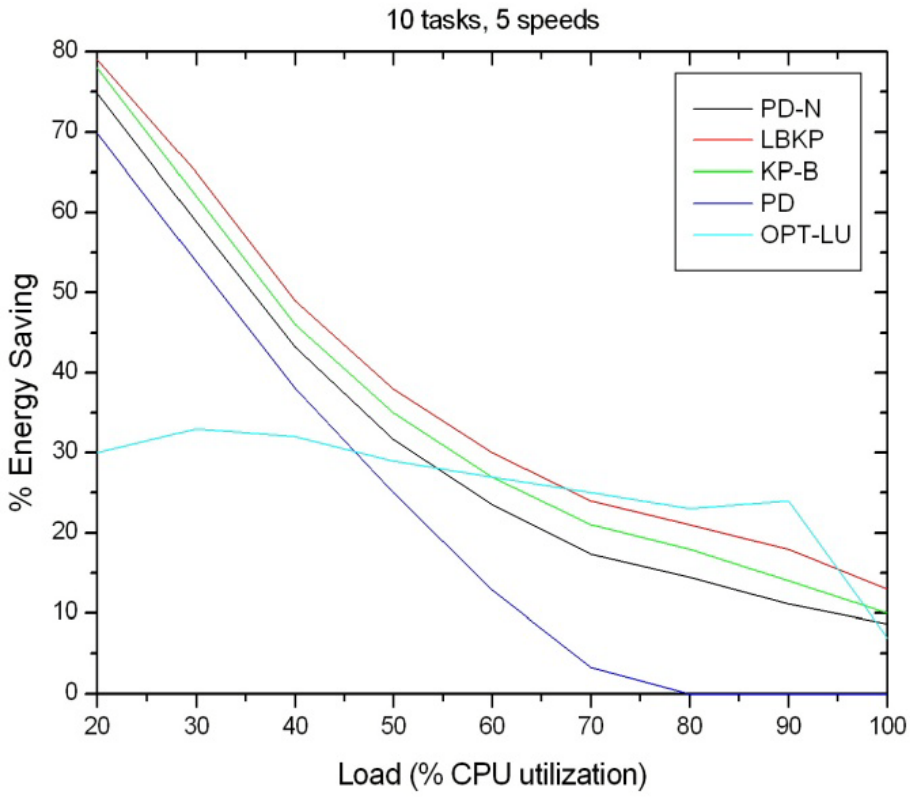


Figura 4.5. Diferentes cargas de trabajo.

El último experimento que se realizó consistió en tener como condición inicial el 60% de la carga del procesador y 5 niveles de velocidad discreta variando el número de tareas de 10 a 100 para obtener el tiempo de ejecución de cada uno de los algoritmos analizados (los resultados se despliegan en la figura 4.6). El algoritmo que consume más tiempo de ejecución es el PD, ya que como se sabe es un algoritmo recursivo, el que le sigue el OPT-LU que conforme aumenta el número de tareas en el sistema, aumenta también el tiempo de ejecución. Los restantes algoritmos tienen un comportamiento parecido. El algoritmo que tiene un menor tiempo de ejecución es el KP-B estando dentro de 13 y 175 milisegundos, seguido muy cerca del LBKP.

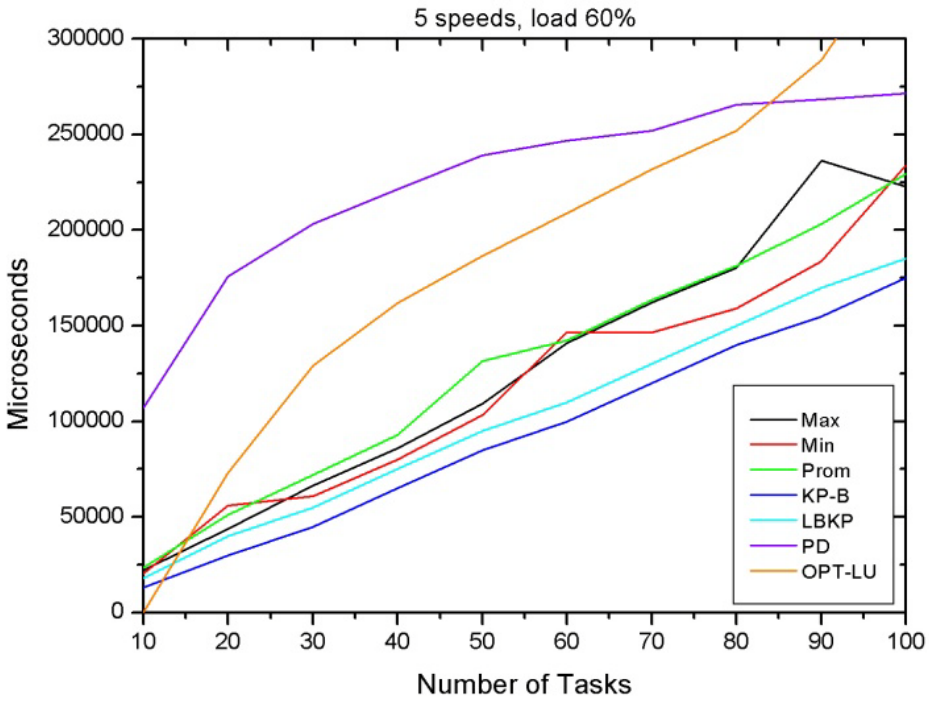


Figura 4.6. Tiempo de ejecución.

4.2.1 Discusión de resultados.

El algoritmo Opt-Lu se prefirió para esta simulación en lugar del algoritmo presentado en [46] ya que al implementar los dos, el primero tuvo un mejor desempeño que el segundo, y en la literatura no se encontró hasta la escritura de este trabajo, ningún otro que resuelva este tipo de problema con las condiciones establecidas y éstas son: la planificación de tareas de tiempo real ejecutando en un procesador con restricciones de precedencia y potencia con un algoritmo estático. En seguida se mencionan los resultados.

a) El algoritmo KP-B tiene el ahorro de energía muy parecido al Opt-Lu cuando se varía el número de tareas de 10 a 100, y el segundo mejor cuando se varía la carga del procesador del 20% al 100%. Y por último, es

el algoritmo con el menor tiempo de ejecución cuando se varía el número de tareas. Por lo que este algoritmo es el mejor de todos los probados en esta simulación.

b) El algoritmo PD-N tiene un comportamiento promedio cuando varía el número de tareas y el mismo comportamiento cuando se varía la utilización del procesador. En cuanto al tiempo de ejecución tiene un tiempo menor cuando se obtiene la opción min ya que ésta permite seleccionar la velocidad más baja. En contra parte cuando es seleccionada la mayor velocidad.

4.3. Resultado del apartado 3.3.

En esta simulación de experimentos, se verifica el algoritmo propuesto con base a nuestro criterio de optimización usando diferentes imágenes y diferentes filtros wavelet. El objetivo en esta simulación de experimentos es medir el ahorro de energía para diferentes tamaños de la imagen. Cada tamaño de cada imagen será observado para diferentes filtros wavelets. Podemos observar que si utilizamos diferentes filtros wavelets podemos obtener diferentes tiempos de ejecución. El tiempo de ejecución medido para cada nivel es medido físicamente en microsegundos, usando una Laptop marca Sony, modelo VGN-T350P, con un procesador Intel Centrino a 3.2 GHZ con 512MB de RAM y corriendo en el Sistema operativo Fedora Linux versión 5.0. La función usada para medir el tiempo es la psched_get_time ().

En la figura 4.7 se observan los tiempos de ejecución obtenidos al realizar la transformada wavelet para cada nivel, usando diferentes filtros. En esta figura se observa que el filtro Villa2 obtiene el peor tiempo y el filtro Haar el mejor, ésto dentro de los primeros 3 niveles, para los niveles posteriores todos los filtros tienden a tener un tiempo igual y tendiendo a cero, debido principalmente a que la transformada en el nivel 5 se realiza sólo para un bloque de 32x32 píxeles. De la

figura 4.7 observamos que los tres mejores filtros son el Haar, Daub4 y Daub6, de estos filtros se obtendrá el tiempo de ejecución para cada nivel utilizando diferentes tamaños de la imagen como se muestra en la figura 4.8.

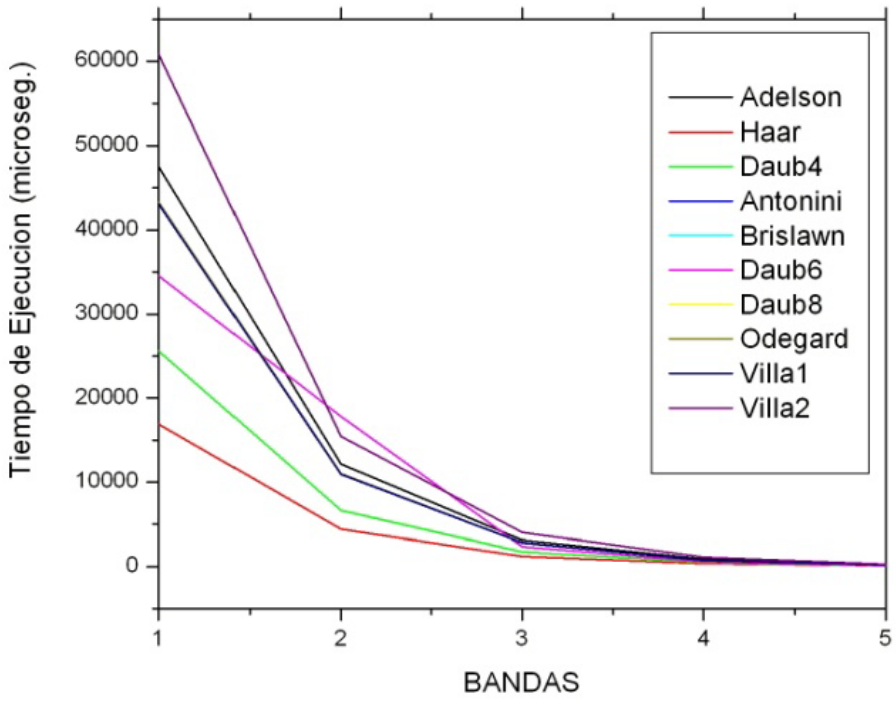


Figura 4.7. Tiempo de ejecución para diferentes filtros

En esta figura 4.8 se aprecia que entre más pequeña es la imagen, el tiempo de ejecución por cada nivel es menor. Observando este mismo comportamiento para los filtros Daub4 (figura 4.9) y Daub6 (figura 4.10). En estas tres figuras se conserva la diferencia entre los tiempos de ejecución para los tres tipos de filtros. Cabe mencionar que estos tiempos (para cada filtro) se obtuvieron casi iguales cuando se probó con otras imágenes, concluyendo que estos tiempos son constantes para cualquier tipo de imagen del mismo tamaño.

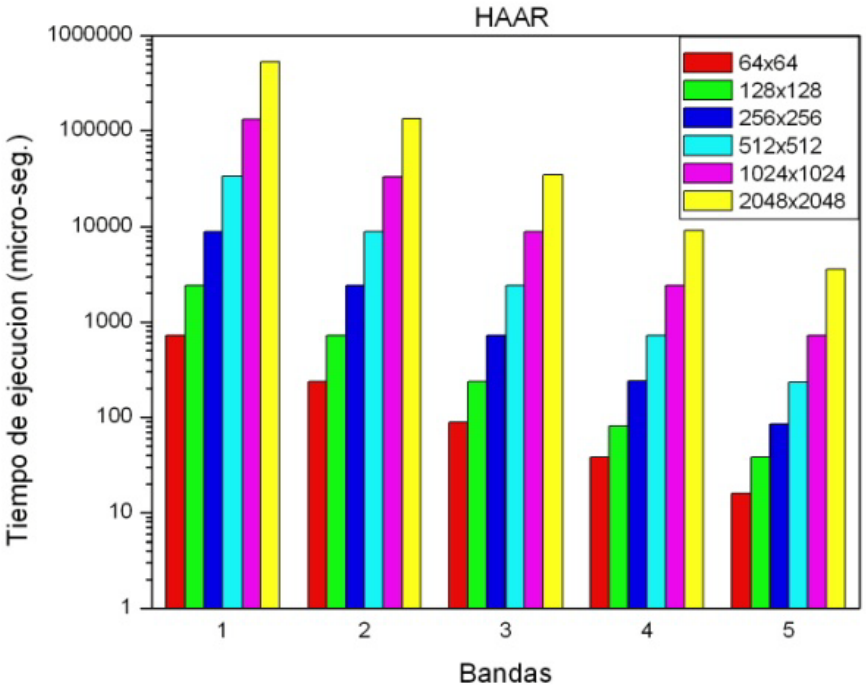


Figura 4.8. Tiempo de ejecución para el filtro Haar.

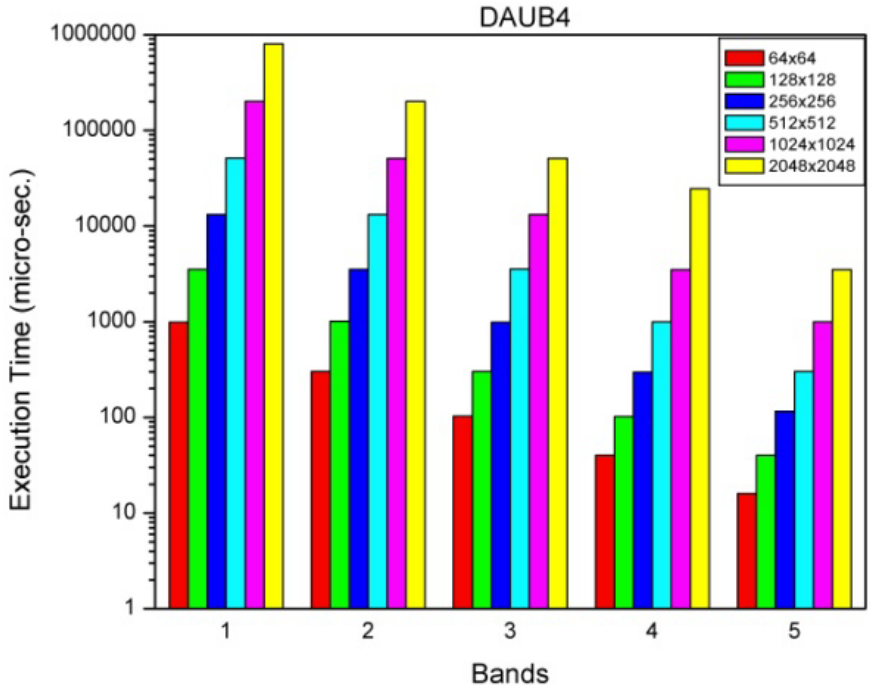


Figura 4.9. Tiempo de ejecución para el filtro Daub4.

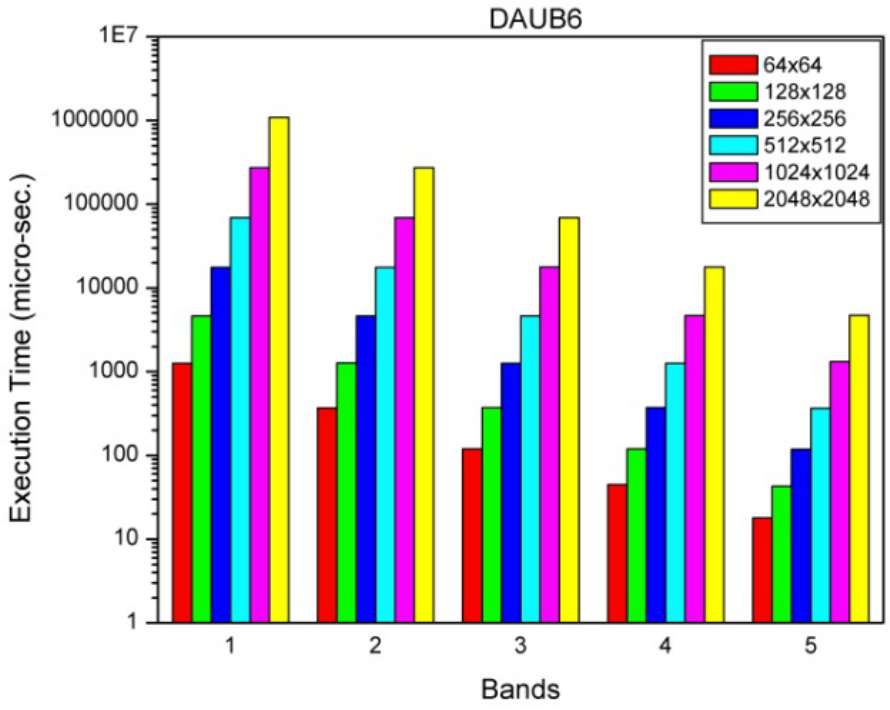


Figura 4.10. Tiempo de ejecución para el filtro Daub6.

El ahorro de energía para estos tres filtros utilizando diferentes tamaños de imagen, se puede observar en la figura 4.11. Para tamaños de imagen pequeños se observa un mayor ahorro de energía y para tamaños de imágenes grandes se obtiene un ahorro de energía menor. El ahorro de energía promedio obtenido fue del orden del 44.19%. En [53] se obtuvo un ahorro del 33.45%, implementado en un sistema embebido y simulando los modelos de potencia. Lo que se planifica son de 2 hasta 6 imágenes que están listas para su procesamiento. En [69] se implemento en una laptop aplicada a dispositivos de multimedia móviles en donde el video es codificado para ser enviado de un dispositivo móvil a otro. Obteniendo un ahorro del 35%. Y finalmente en [48] se simulo en un PDA la descompresión de un trama de video de imágenes en MPEG utilizando la transformada coseno discreta, obteniendo un ahorro de energía del 24%.

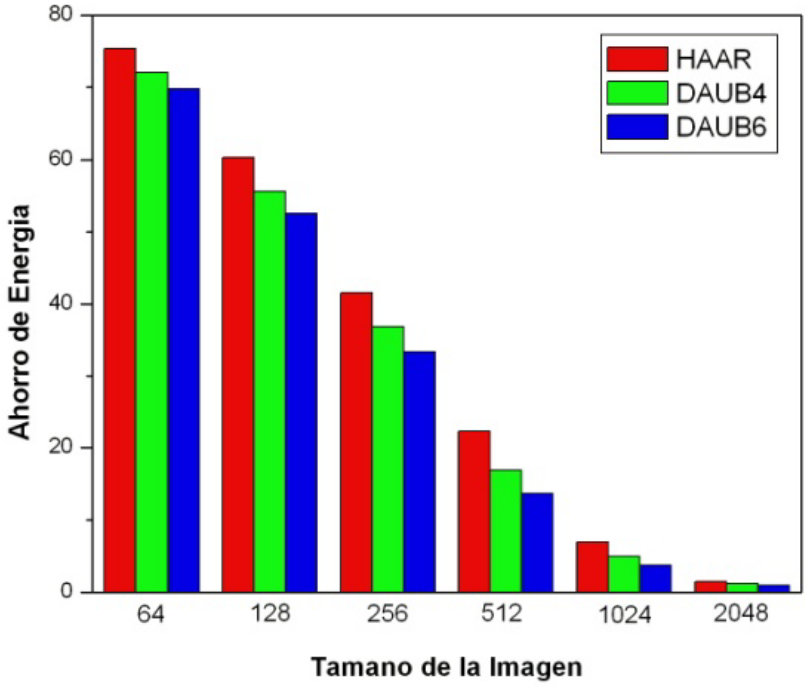


Figura 4.11. Ahorro de energía a diferentes tamaños de imagen

El último paso consistió en correr nuestro algoritmo en forma paralela para observar cuanto se reduciría el ahorro de energía en este ambiente. El algoritmo se programó para ser ejecutado para memoria compartida. Se utilizó la librería de Pthread la cual cumple con los estándares POSIX y nos permitió trabajar con dos hilos de ejecución (threads) al mismo tiempo. En la figura 4.12 se observa que tenemos un ahorro de energía promedio extra del orden de 28.77%.

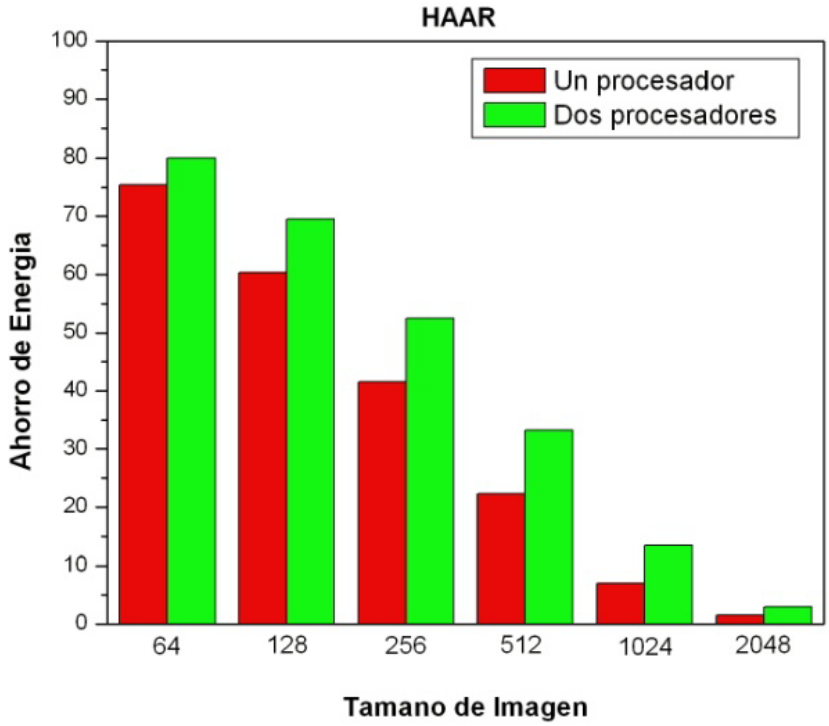


Figura 4.12. Ahorro de energía para dos threads.

RESUMEN

En este capítulo se mostraron los resultados obtenidos al implementar los algoritmos del capítulo 3 y los propuestos en el capítulo 1.

CAPITULO 5

CONCLUSIONES

En este capítulo se presentan las conclusiones finales de cada uno de los planteamientos hechos en los objetivos específicos plasmados en el capítulo 1 y su planteamiento en el capítulo 3 y sus respectivas soluciones en el capítulo 4.

De forma general, se cumple con la hipótesis planteada en el inciso 1.7, ya que el ahorro de energía total del conjunto de tareas de tiempo real, son ejecutadas todas las tareas cada hiperperíodo del sistema y se calcula la velocidad de ejecución de cada tarea por el algoritmo de planificación.

5.1 Conclusión del cálculo de variaciones.

Se propuso un método de optimización para el manejo de la potencia para un DAG corriendo en un procesador de velocidad variable con hasta 16 niveles discretos de velocidad. El problema es presentado como un problema de decisiones de N etapas. Los resultados presentados muestran que el algoritmo PD-N tiene un desempeño muy cercano al de la literatura pero con una baja complejidad computacional. Las utilizaciones correspondientes de cada tarea se ordenan en un diagrama de estado de mayor a menor por columnas y conservando las restricciones de precedencia entre las tareas. El algoritmo PD-N presenta tres diferentes opciones para encontrar la velocidad deseada. Los resultados muestran que el algoritmo tiene un buen desempeño al alcanzar un ahorro de energía muy cercana a la holgura en el sistema (óptimo), para un número de nodos que van de 6 a 16 su comportamiento es muy parecido y muy cercano al óptimo, pero cuando varía la velocidad del procesador de 4 a 16

niveles, el comportamiento en base a la holgura es muy similar, lo que varía es que se abre un poco más la diferencia entre los diversos niveles pero muy cercanos al óptimo, por último el tiempo de ejecución tiende a decrecer en cuanto la carga del procesador aumenta, pero una carga entre 0.2% y 0.5% vemos que el método Max emplea más tiempo de ejecución y el método Min el menor tiempo, este comportamiento se explica debido a que el algoritmo empieza a buscar el valor mínimo a partir de la menor utilización, por lo que encuentra primero este valor.

5.2 Conclusión de la programación lineal.

En esta conclusión se desarrolló un método para el manejo de la potencia para un conjunto de tareas de tiempo real corriendo en un procesador de velocidad variable con hasta 16 niveles discretos de velocidad. Se presentó como un problema de programación lineal. Los resultados presentados muestran que el algoritmo PD-N tiene un desempeño promedio del 23% de ahorro de energía cuando el número de tareas van de 10 a 100 y el número de velocidades discretas es igual 5. Igualmente se tiene un comportamiento promedio (del 75% al 8 %, ver figura 4.5) cuando se varía la carga del procesador del 20 al 100%, para 10 tareas y 5 niveles de velocidad discretas. Este mismo algoritmo se dividió en tres formas posibles de selección para la velocidad más idónea. Estos son: el valor máximo (MAX), mínimo (MIN) y el promedio (PROM) de la siguiente velocidad a escoger. De estas tres opciones se obtiene un tiempo de ejecución muy parecido para el KP-B y está entre el PD y el LBKP. Para el algoritmo KP-B el ahorro de energía, cuando se varía el número de tareas, es muy cercano al Opt-Lu. Cuando se varía la carga del procesador tiene un mejor ahorro de energía que el Opt-Lu a partir del 60% de carga del procesador. Con respecto al tiempo de ejecución el KP-B es el que tiene menor tiempo, estando dentro de los 13 y 175 milisegundos.

5.3 Conclusión del problema especifico.

El tercer objetivo específico se propuso un método de optimización para la transformada wavelet discreta aplicado a imágenes corriendo en un procesador de velocidad variable. El problema se presenta como un problema lineal con restricciones discretas. La solución aproximada propuesta está basada en la ecuación Bellman. Para una imagen es obtenido un ahorro de energía del 44.19%. Comparando los tiempos de ejecución entre los diferentes filtros se observa que el filtro Villa1 es el peor y el filtro Haar es el mejor, pero acercándose al quinto nivel, para todos los filtros el tiempo de ejecución tiende a cero. Tomando en cuenta sólo los tres mejores filtros, obtenemos el tiempo de ejecución para diferentes tamaños de imagen, observando que para tamaños de imagen pequeños se obtiene el menor tiempo de ejecución. También se obtuvieron los tiempos de ejecución de otras imágenes y se concluyó que estos tiempos son iguales para cualquier tipo de imagen del mismo tamaño. En seguida se obtuvo el ahorro de energía para diferentes tamaños de imágenes, en donde los tamaños pequeños son los que presentan un ahorro de energía mayor y para un tamaño mayor se aprecia un ahorro de energía menor. Por último al ejecutar el algoritmo en una ambiente de memoria compartida, se obtiene un ahorro promedio extra del orden de 28.77%.

5.4 Aportaciones.

El objetivo general plantado en el desarrollo de esta presente tesis fue la de proponer un método de implementación algorítmica para resolver el problema de planificar un conjunto de tareas de tiempo real con restricciones de precedencia y potencia. Simulando el comportamiento de los procesadores de velocidad variable y demostrando la efectividad de los algoritmos propuestos. Además se consideró simultáneamente el manejo de energía y la confiabilidad, es decir, se garantizó que la ejecución de las tareas de tiempo real, no pierdan sus plazos de respuesta. En cuanto a las aportaciones de este desarrollo se concluyen como sigue:

1) Se propuso un método y se desarrollaron dos algoritmos: uno se implementó para una aplicación representativa. El primer algoritmo se presentó como un problema de decisiones de N etapas. Los resultados muestran que el algoritmo PD-N tiene un desempeño muy cercano al de la literatura con una baja complejidad computacional. El segundo algoritmo (KP-B) se presentó como un problema de programación línea con restricciones discretas, en donde la solución está basada en la ecuación de Bellman. Estos dos algoritmos se implementaron y se compararon con el más cercano de la literatura, siendo el KP-B (mejor, peor, etc.). Para el algoritmo KP-B el ahorro de energía, cuando se varía el número de tareas, es muy cercano al Opt-Lu. Cuando se varía la carga del procesador tiene un mejor ahorro de energía que el Opt-Lu a partir del 60% de carga del procesador. Con respecto al tiempo de ejecución el KP-B es el que tiene menor tiempo, estando dentro de los 13 y 175 milisegundos.
2) Se implementó una aplicación para el ahorro de energía enfocado a la transmisión y recepción de imágenes, enfocándose en realizarlo en la transformada wavelet, tal que las sub-bandas no pierdan sus plazos de respuesta. Este proceso se realiza cada vez que arribe o salga una imagen del sistema. Se implementó el algoritmo propuesto y se realizaron los experimentos pertinentes al problema en cuestión.

5.5 Trabajo a futuro.

El trabajo a futuro, desde el punto de vista práctico, necesariamente tiene que considerar la implementación de los algoritmos desarrollados en esta tesis dentro de una plataforma de tiempo real. Esta plataforma deberá contener un procesador de velocidad variable, tal como el Xscale de Intel y la ejecución de los algoritmos de planificación sobre un sistema operativo de tiempo real.

5.6 Publicaciones.

Reportes técnicos:

1) “Análisis del Tiempo de Ejecución de Algoritmos para la asignación de Tareas de Tiempo Real en Multiprocesadores”, Héctor Eduardo Silva López y Sergio Suárez Guerra, No. 235, Serie: Azul, Fecha: Octubre 2005. ISBN: 970-36-0309-3.

Articulos:

1) “Planificación de un grafo de tareas de Tiempo-Real para el ahorro de energía”, Héctor Silva-López, Sergio Suárez-Guerra. Revista Científica, Vol. 12 Núm. 4, Octubre-Diciembre de 2008.

Proceedings:

1) “Execution Time Comparison of Algorithms for the Assignment of Real Time Tasks for Multiprocessors”, Hector Silva-Lopez and Sergio Suarez Guerra, Proceedings of the Research on Computing Science, Advances on Digital System Design, Vol. 15, pag. 13-23, September, 2005. CICINDI-2005. ISSN: 1665-9899.

2) “Haar Wavelet Transform for Low Power Video Frames”, Héctor Silva-López, Sergio Suárez-Guerra. Proceedings of the Research on Computing Science, Control, Virtual Instrumentation and Digital Systems, Vol. 24, pag. 21-30, November, C ICINDI-2006. ISSN: 1870-4069.

3) “Algoritmo para el ahorro de Energía en la Compresión de Imágenes”, Héctor Silva-López, Marcelino Becerril Silva, Sergio Suárez-Guerra. Primer Encuentro de Estudiantes en Ciencias de la Computación E2C2. 14 al 18 de Mayo de 2007. ISBN-10 970-36-0404-8 y ISBN-13 978-970-36-0404-3

4) “Energy Saving Analysis for the Wavelet Transform Processing”, Hector Silva-Lopez and Sergio Suarez-Guerra, Research in Computing Science, Special issue: Advances in Computer Science and Engineering, Vol 29, pag. 285-295, 16th International Conference on Computing, November 4-9 2007. ISSN: 1870-4069.

REFERENCIAS BIBLIOGRAFICAS

[1] A. Burchard, J. Liebeherr, Y. Oh, and S,H, Son. “New strategies for assigning real time tasks to multiprocessor systems”. IEEE Transactions on Computers, 44(12):1429-1442, December 1995.

[2] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. “Power aware page allocation”. In Proc. Of the 9 th International Conference on Architectural Support for Programming Languages and Operating Systems, Nov. 2000.

[3] A. Sinha and A. P. Chandrakasan. Jouletrack, “A web based tool for software energy profiling”. In Proc. of Design Automation Conference, Jun 2001.

[4] Audsley N. C., “Deadline Monotonic Scheduling”, Department of Computer Science, University of York, Technical Report YCS_146, September 1991.

[5] Aydin H., Melhem R., Mossé D. and Mejia-Alvarez P., “Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics”. Proceedings of the 13th EuroMicro Conference on Real-Time Systems (ECRTS’01), June 2001.

[6] Aydin H., Melhem R., Mossé D. and Mejia-Alvarez P., “Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems”. Proceedings of the Real-Time Systems Symposium, December 2001. London U.K.

[7] Bach, Maurice J., “The Design of the UNIX Operating System”. Prentice/Hall International, Inc., 1986.

[8] Balas E. and E. Zemel,”An Algorithm for Large Zero One Knapsack Problems”, Operations Research, 28 (1980), 1130-1154.

[9] B. Andersson. “Adaption of time-sensitive tasks on shared memory multiprocessor: A framework suggestion. Master´s thesis, Department of Computer Engineering, Chalmers University of Technology, January 1999.

[10] Bellman R.E., “Dynamic Programming”, Princeton University Press, N.J. (1957).

[11] Bu Aiguo Shi Longxing Hu Chen Li Jie Wang Chao, **"**Energy-optimal dynamic voltage scaling for sporadic tasks", Proceedings IEEE International Symposium Circuits and Systems, ISCAS 2006. 2006.

[12] Burd T. D., Pering T. A., Stratakos A. J. and Brodersen R. W., " A Dynamic Voltage Scaled Microprocessor System", IEEE J. of Solid-State Circuits, Vol. 35, No. 11, 2000.

[13] Burd T. D. and Brodersen R. W., "Energy efficient CMOS microprocessor design", In Proc. of The HICSS Conference, Jan. 1995, pp. 288-297.

[14] Chandrakasan A., Sheng S. and Brodersen R., "Low-power CMOS digital design", IEEE Journal of Solid-State Circuit, 27(4), 1992, pp. 473-484.

[15] Cheng, H. and Goddard, S. "Integrated device scheduling and processor voltage scaling for system-wide energy conservation". In Proceedings of the International Workshop on Power-Aware Real-time Computing. 24--29. 2005.

[16] Claudio Scordino and Enrico Bini, "Optimal Speed Assignment for Probabilistic Execution Times", The 2nd Int"l Workshop on Power-Aware Real-Time Computing, Jersey City, New Jersey, September, (PARC), 2005.

[17] C. L. Liu. "Scheduling algorithms for multiprocessor in a hard real-time environment". In JPL Space Programs Sumary 37-60, volume II, pages 28-31, 1969.

[18] C. L. Liu and W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment". Journal of the Association for Computing Machinery, 20(1):46-61, January 1973.

[19] D. Cruz Pérez, P. Guevara López, J.J. Medel Juárez, "Modelado de los Tiempos de Ejecución para una Tarea con n Instancias", XVIII Congreso Nacional y IV Congreso Internacional de Informática y Computación, Torreón Coahuila, México, Octubre-2005.

[20] Giorgio C. Buttazo, "Hard Real-Time Computing Systems Predictable Scheduling and Applications", Kluwer Academic Publisher, 1977.

[21] Gruian, F**., "**Hard real-time scheduling for low-energy using stochastic data and DVS processors", In Low Power Electronics and Design, International Symposium on, 2001.

[22] Gruian, F. Kuchcinski K., **"**LEneS: task scheduling for low-energy systems using variable supply voltage processors", In Design Automation Conference, Proceedings of the ASP-DAC, Asia and South Pacific, 2001.

[23] Harter P. K., "Response times in level-structured systems", ACM Transactions on Computer Systems, vol. 5, no. 3, pp. 232-248, Aug. 1984.

[24] Hong I., Qu G., Potkonjak M. and Srivastava M., "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors", In Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98), Madrid, December 1998.

[25] http://developer.intel.com/design/intelxscale/benchmarks.htm,2002.

[26] http://developer.intel.com/design/xscale, Intel XScale.

[27] http://www.curvefit.com/2004.

[28] http://www.motorola.com/webapp/sps/library/prod_lib.jsp,Motorola MPC5200 32 bits.

[29] http://www.transmeta.com/crusoe, Transmeta Crusoe.

[30] Intel Corp. "Mobile Pentium iii processor-m datasheet". Order Number: 298340-002, Oct 2001.

[31] Intel Strong ARM SA1100, Intel.strong ARM SA-1100 Microprocessor Developer's Manual 2003.

[32] James, L.P. y Silberschatz, A., "Sistemas Operativos. Conceptos fundamentales". Editorial Reberté, S.A., 1991.

[33] Jian-Jia Chen and Tei-Wei Kuo, "Energy-Efficient Scheduling of Periodic Real-Time Tasks over Homogeneous Multiprocessors", The 2nd Int"l Workshop on Power-Aware Real-Time Computing, Jersey City, New Jersey, September, (PARC), 2005.

[34] Joseph M. y Pandya P., “Finding Response Times in a Real-Time System”, The Computer Journal (British Computing Society) 29, 5, pp. 390-395, October 1986.

[35] jpastor@netmediaeurope.com

[36] J.Y. T. Leung. A new algorithm for scheduling periodic, real time tasks. Algorithmica, 4(2):209-219, 1989.

[37] J. Y. T. Leung and J. Whitehead. “On the complexity of fixed-priority scheduling of periodic, real time tasks”. Performance Evaluation, 2(4):237-250, December 1982.

[38] Kernighan, B. W. y Pike, R. , “The Unix Programming Environment”. Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[39] Lehoczky J., Sha L. y Ding Y., “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior”, IEEE Real-Time Systems Symp., CSPress, Los Alamitos, CA, pp. 166-171, 1989.

[40] L. Lundberg. “Multiprocessor scheduling of age constraint processes”. In 5th International Conference on Real Time Computing Systems and Applications, Hiroshima, Japan, October 27-29, 1998.

[41] Luo J. and Jha N. K., “Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems”, Proc. Intl. Conf. Computer-Aided Design, 2000, pp. 357-364.

[42] M. A. Moncusi, A. Arenas and J. Labarta, “Improving energy saving in hard real time systems via a modified Dual Priority scheduling”, ACM SigArch Computer Architecture Newsletter, Vol. 29, 19-24, 2004.

[43] M. A. Moncusi, A. Arenas and J. Labarta, “Moving Average Frequency Reduction for Low Power in Hard Real Time Systems”, Power-Aware Real-Time Computing Workshop (PARC2005), N. J. U.S.A., 2005.

[44] Medel J. J. J., Guevara L. P., Cruz P. D., Sistemas en tiempo real. Conceptos básicos, IPN- SEP, 2008.

[45] Mejia-Alvarez P., Levner E., Mossé D., “Adaptive scheduling server for power-aware real-time tasks”, ACM Transactions on Embedded Computing Systems (TECS), v.3 n.2, p.284-306, May 2004

[46] Nevine A. , D. Mosse , B. Childers and R. Melhem, “Toward the Placement of Power Management Points in Real Time Applications”, In COLP’01:Workshop on Compilers and Operating Systems for Low Power, IEEE Press, Piscataway, N:J.

[47] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Ra-jamony. “The case for power management in web servers”, chapter 1. Power Aware Computing. Plenum/Kluwer Publishers, 2002.

[48] Pering T., Burd T., Brodersen R., “The simulation and evaluation of dynamic voltage scaling algorithms”, Proceedings of the 1998 international symposium on Low power electronics and design, p.76-81, August 10-12, Monterey, California, United States, 1998.

[49] P. Pillai and K. G., “Shin. Real-time dynamic voltage scaling for low-power embedded operating systems”. In Proc. of 18th ACM Symposium on Operating Systems Principles (SOSP’01), Oct. 2001.

[50] Quadi A., Goddard S., Farritor S., “A Dynamic Voltage Scaling Algorithm for Sporadic Tasks”, Proceedings of the 24th IEEE International Real-Time Systems Symposium”, p.52, December 03-05, 2003

[51] Rambus. Rdram. http://www.rambus.com/, 1999.

[52] Roychowdhury D., Koren I., Krishna C. M., “A Voltage Scheduling Heuristic for Real-Time Task Graphs”, In Proceedings of International Conference on Dependable Systems and Networks, PP. 741-750, June 2003.

[53] Ruibin Xu, Daniel Mosse and Rami Melhem, “Evaluating a DVS Scheme for Real-Time Embedded Systems”, The 2nd Int”l Workshop on Power-Aware Real-Time Computing, Jersey City, New Jersey, September, (PARC), 2005.

[54] R. Xu, C. Xi, R. Melhem and D. Mosee, “Minimizing Expected Energy in Real-Time Embedded Systems”, In EMSOFT, Jersey City, New Jersey, September 2005.

[55] Santa-Cruz, D., T. Ebrahimi, J. Askelof, M. Larsson and C.A. Christopoulos, “JPEG 2000 Still Image Coding Versus Other Standards”, Proceeding of SPIE 89 45th Annual Meeting Applications of Dogital Image Processing XXIII, Vol 4115, San Diego, CA, July 30 August 4, 2000, pp. 446-454.

[56] S. Davari and S. K. Dhall, “An on line algorithm for real time allocation”, 19th Ann. Hawaii Int´l Conf. System Sciences, pp 133-141, 1986.

[57] Shaw D. X. and Cho G., “The Critical Item, Upper Bounds and a Branch and Bound Algorithm for the Tree Knapsack Problem”, Networks, 31(4), 1998.

[58] Shin Y. and Choi K., “Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems, In Proceedings of the 36th Design Automation Conference, DAC’99, pp. 134-139.

[59] S.K. Dhall and C. L. Liu. “On a real time scheduling problem”. Operations Research, 26(1):127-140, January/February 1978.

[60] S. Lauzac, R. Melhem, and D. Mossé. “Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor”. In 10th Euromicro Workshop on Real Time Systems, pages 188-195, Berlin, Germany, June 17-19, 1998.

[61] Stankovic J. A. y Ramamritham K., “Hard Real-Time Systems”, IEEE Computer Society, catalog no. EH0276-6, 1988.

[62] S. Thompson, P. Packan, and M. Bohr. “Mos scaling: Transistor chanllenges for the 21st century”. Intel Technology Journal, Q3, 1998.

[63] Subramanya, S.R., “Image Compression Techniques”, IEEE Protentials Vol 20, No. 1, February/March 2001.

[64] Tarek A. AlEnawy and Hakan Aydin, “Energy-Aware Task Allocation for Rate Monotonic Scheduling”, IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), San Francisco, California, March 7-10, 2005.

[65] Tarek A. AlEnawy and Hakan Aydin, “Energy-Constrained Scheduling for Weakly-Hard Real-Time Systems”, Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS’05), Miami, FL, December, 2005.

[66] Toledo M. and Medel J., “Maximum Entropy Stochastic Tasks Classification”, Journal of Advances in Computer Science and Engineering, vol. 33, pp. 53-62, 2006.

[67] Topiwala, P.N.,”Wavelet Image and Video Compression”, Kluwer Academic Publishers, 1998.

[68] Trader Michael, “Embedded Real Time Techniques for Calculating CPU Utilization”, Spring Embedded Systems Conference, Course ESC-449, San Francisco, USA, March 2004.

[69] Vardhan V., Grobe Sachs D., Yuan W., Harris A., V. Adve S., L. Jones D., H. Kravets R., Nahrstedt K., ”Integrating fine-grain application adaptation with global adaption for saving energy”, In Proceedings of the 2nd International Workshop on Power-Aware Real-Time Computing, 2005.

[70] Villaseñor, J. Belzer B. and J. Liao, “Wavelet Filter Evaluation for Image Compression”, IEEE Transactions on Image Processing, Vol. 4, No. 8, pp. 1053-1060, August 1995.

[71] Welstead, S.,”Fractal and Wavelet Image Compression Techniques”, SPIE Optical Engineering Press, 1999.

[72] Wu D., Al-Hashimi B. M. and Eles P., “Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems”, Design Automation and Test in Europe Conference (DATE’03), March, 2003, Munich, Germany.

[73] Xiliang Zhong, Xu Cheng-Zhong, “System-wide energy minimization for real-time tasks: lower bound and approximation”, Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design, November 05-09, San Jose, California, 2006.

[74] Xiliang Zhong, Xu Cheng-Zhong, “Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Real-Time Guarantee”, IEEE Transactions on Computers, v.56 n.3, p.358-372, March 2007.

[75] Xiliang Zhong , Cheng-Zhong Xu, “Frequency-aware energy optimization for real-time periodic and aperiodic tasks”, Proceedings of the ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, June 13-15, San Diego, California, USA, 2007.

[76] Yao F., Demers A. and Shanker S., “A Scheduling Model for Reduced CPU Energy”. IEEE Annual Foundations of Computer Science, 1995. pp. 374-382.

[77] Y. Oh and S. H. Son. “Tight performance bounds of heuristics for a real time scheduling problem”, Technical Report CS-93-24, Univ. of Virginia, Dept. Computer Science, May 1993.

[78] Yan Zhang, Z. Lu, J. Lach, M. Stan, K. Skadron, “Optimal Procrastinating Voltage Scheduling for Hard Real-Time Systems", Design Automation Conference, pp. 905-8, 2005.

[79] Yumin Zhang, Xiaobo Hu and Danny Z. Chen, “Task Scheduling and Voltage Selection for Energy Minimization”, Proceedings of the 39th Conference on Desing automation, New Orleans, USA, 2002.

[80] Yumin Zhang, Hu, X.S., and Chen, D. Z., **“**Energy minimization of real-time tasks on variable voltage processors with transition energy overhead”, In Design Automation Conference, Proceedings of the ASP-DAC 2003. Asia and South Pacific, 2003.

[81] Zhu D., Melhem R. and Mossé D. “Power Aware Scheduling for AND/OR Graphs in Real-Time Systems”, IEEE Trans. on Parallel and Distributed Systems, Nov. 2003.

[82] Zhu Dakai, “Energy and Reliability management in parallel real-time systems”, dissertation thesis, November 12, 2004.

APENDICE A

Definición de sistema

Un *sistema* es una caja negra que tiene un conjunto de una o más entradas y un conjunto de una o más salidas. En el caso de sistemas de software, las entradas consisten en datos digitales provenientes de dispositivos de entrada u otros sistemas de software, y las salidas son datos digitales.

Estado de un proceso

Un proceso es un programa en ejecución. La ejecución de un proceso debe proceder en forma secuencial, es decir, en cualquier momento se ejecuta como máximo una instrucción en nombre del proceso. Un proceso es más que el código del programa aunado a la actividad que se desarrolle; por lo general, incluye también la pila del proceso que contiene datos temporales (como parámetros de subrutinas, direcciones de retorno y variables temporales) y una sección de datos con variables globales.

Al ejecutarse un proceso, este cambia de estado. El estado de un proceso se define por la actividad de este proceso, y cada proceso secuencial puede encontrarse en uno de los tres estados siguientes:

- **En ejecución.** Las instrucciones se están ejecutando.
- **En espera o bloqueado.** El proceso está esperando a que ocurra algún suceso (como la conclusión de una e/s).
- **Listo.** El proceso está esperando que se le asigne a un procesador.

Es importante observar que en un momento determinado sólo un proceso puede encontrarse en ejecución, aunque varios procesos pueden estar listos o en espera. En la figura A.1 se representa el diagrama correspondiente a estos tres estados

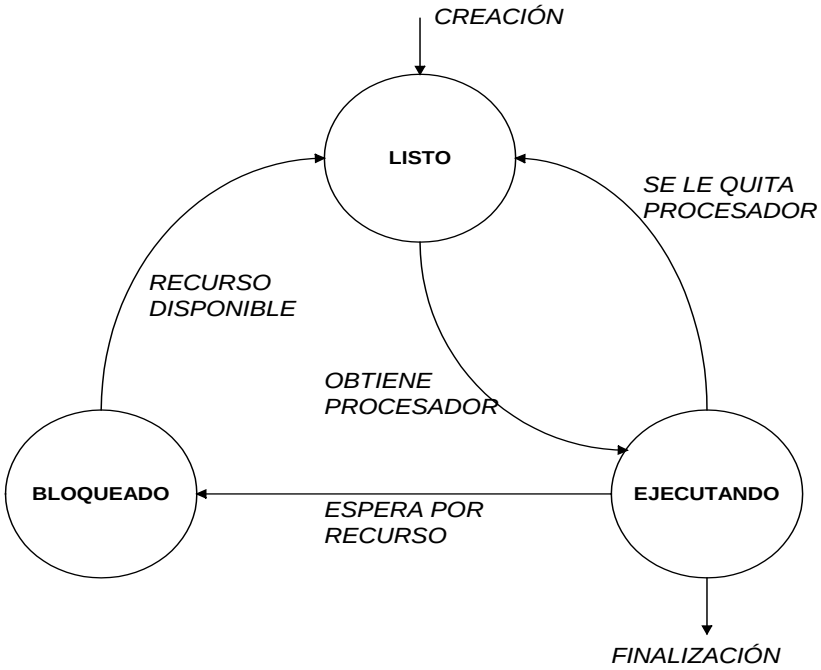


Figura A.1. Diagrama de estado de un proceso.

Tipos de restricciones de tareas

Desde el punto de vista de las restricciones de los sistemas de tiempo real, se pueden clasificar en tres diferentes categorías: restricciones temporales, restricciones de precedencia y restricciones de recursos [20, 44].

Restricciones temporales.

Normalmente los requerimientos temporales que se imponen a un sistema se refieren al tiempo de respuesta de cada una de las tareas que lo componen. El más usual y sobre el que se centra gran parte del análisis de tiempo real, es el

concepto de plazo de una tarea, considerando como el máximo tiempo necesario para obtener una respuesta. Otros tipos de requerimientos se refieren a la separación entre dos respuestas consecutivas, a la capacidad mínima de procesamiento, etc. Los parámetros asociados a una tarea de tiempo real son:

- **Tiempo de activación** a_i – Es el tiempo en que una tarea está lista para su ejecución.
- **Tiempo de ejecución** C_i - Es el tiempo necesario para ejecutar una tarea sin interrupción
- **Tiempo de inicio** si – El tiempo en que una tarea comienza su ejecución.
- **Tiempo de finalización** f_i – Es el tiempo en la cual una tarea finaliza su ejecución.
- **Tiempo de respuesta** R_i – Es el tiempo (medido desde su activación) hasta su terminación.
- **Plazo de respuesta absoluto** d_i - Es el tiempo en que debe estar completada la tarea. $d_i = R_i + D_i$.
- **Plazo de respuesta relativo** D_i - Se mide con respecto al tiempo de activación. $D_i = d_i + a_i$.
- **Latencia** L_i : $L_i = f_i - d_i$, representa el retardo en la terminación de una tarea con respecto a su plazo de respuesta. Si una tarea termina su ejecución antes de su plazo de respuesta, su latencia es negativa.
- **Tiempo de holgura** X_i : $X_i = d_i - a_i - C_i$, Es el tiempo máximo que una tarea puede ser retardada para completar su plazo de respuesta.

Algunos parámetros temporales se muestran en la figura A.2.

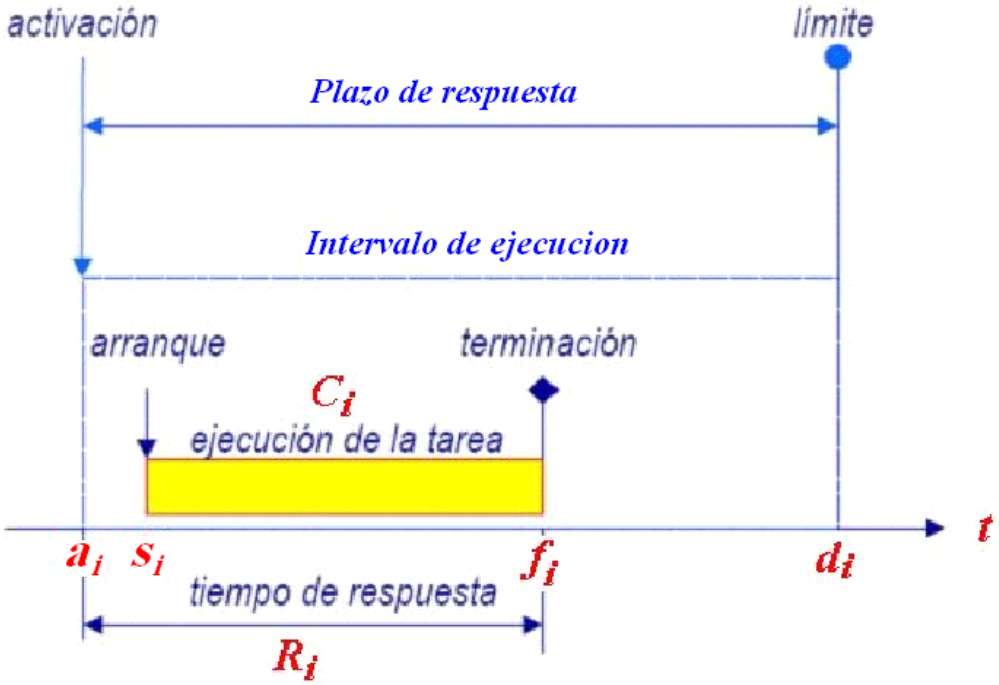


Figura A.2. Requerimientos temporales.

Restricciones de precedencia

Las restricciones de precedencia se pueden representar a través de un grafo y un árbol, Un grafo se representa como **G = (V, A)**, en donde **V** y **A** son conjuntos finitos. Los elementos de **V** y de **A** se llaman, respectivamente, "vértices" y "aristas" de **G**. Las Aristas son las líneas con las que se unen los vértices de un grafo y con la que se construyen también caminos. Si la arista carece de dirección se denota indistintamente *{a, b}* o *{b, a},* siendo a y b los vértices que une. Si *{a ,b}* es una arista, a los vértices a y b se les llama sus extremos.

Los grafos se pueden clasificar en dos grupos: dirigidos y no dirigidos. En un grafo no dirigido el par de vértices que representa un arco no está ordenado. Por lo tanto, los pares (v1, v2) y (v2, v1) representan el mismo arco. En un grafo dirigido cada arco está representado por un par ordenado de vértices, de forma que representan dos arcos diferentes.

Algunos de los principales tipos de grafos son los que se muestran a continuación:

- *Grafo regular:* Aquel con el mismo grado en todos los vértices. Si ese grado es k lo llamaremos k-regular
- *Grafo bipartito:* Es aquel con cuyos vértices pueden formarse dos conjuntos disjuntos de modo que no haya adyacencias entre vértices pertenecientes al mismo conjunto.
- *Grafo completo:* Aquel con una arista entre cada par de vértices. Un grafo completo con n vértices se denota K_n.
- *Un grafo bipartito regular:* se denota $K_{m,n}$ donde m, n es el grado de cada conjunto disjunto de vértices.
- *Grafo nulo:* Se dice que un grafo es nulo cuando los vértices que lo componen no están conectados, esto es, que son vértices aislados.
- *Grafos isomorfos:* Dos grafos son isomorfos cuando existe una correspondencia biunívoca (uno a uno), entre sus vértices de tal forma que dos de estos quedan unidos por una arista en común.
- *Grafos platónicos:* Son los Grafos formados por los vértices y aristas de los cinco sólidos regulares (Sólidos Platónicos), a saber, el tetraedro, el cubo, el octaedro, el dodecaedro y el icosaedro.
- *Grafos eulerianos.* Para definir un grafo euleriano es importante definir un camino euleriano primero. Un camino euleriano se define de la manera más sencilla como un camino que contiene todos los arcos del grafo.
- *Grafos conexos.* Un grafo se puede definir como conexo si cualquier vértice V pertenece al conjunto de vértices y es alcanzable por algún otro. Otra definición que dejaría esto más claro sería: "un grafo conexo es un grafo no dirigido de modo que para cualquier par de nodos existe al menos un camino que los une". Ver figura A.3.

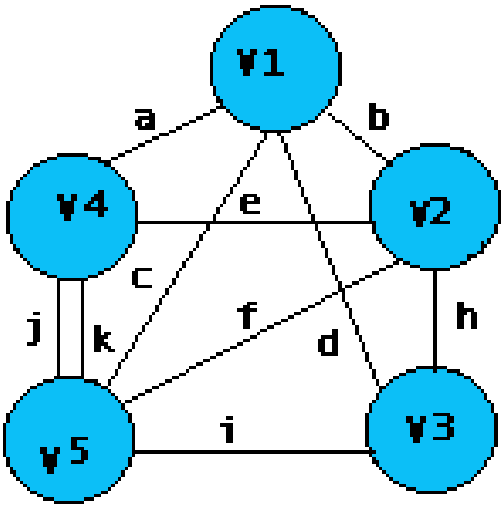


Figura A.3. Grafo conexo

- *Árboles.* Un árbol se define como un tipo de grafo que no contiene ciclos, es decir es un grafo también acíclico, pero a su vez es conexo.
- *Bosques de árboles.* Los bosques de árboles son un caso similar a los árboles, son acíclicos, pero no son conexos.

Recorrer un grafo significa tratar de alcanzar todos los nodos que estén relacionados con uno que llamaremos nodo de salida. Existen básicamente dos técnicas para recorrer un grafo: el recorrido en anchura y el recorrido en profundidad.

- *Recorrido en anchura:* El recorrido en anchura supone recorrer el grafo, a partir de un vértice dado, en niveles, es decir, primero los que están a una distancia de un arco del vértice de salida, después los que están a dos arcos de distancia, y así sucesivamente hasta alcanzar todos los vértices a los que se pudiese llegar desde el nodo de salida.
- *Recorrido en profundidad:* el recorrido en profundidad trata de buscar los caminos que parten desde los vértices de salida hasta que ya no es posible

avanzar más. Cuando ya no puede avanzarse más sobre el camino elegido, se vuelve atrás en busca de caminos alternativos.

Dígrafo (grafo dirigido). A un grafo dirigido se le puede definir como un grafo que contiene aristas dirigidas, como en la figura A.4.

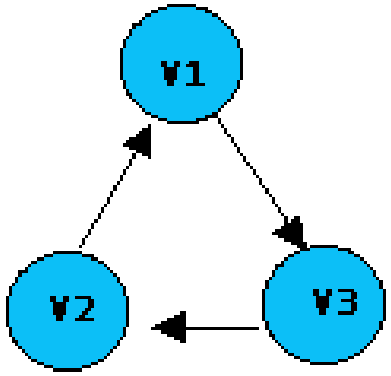


Figura A.4. Grafo dirigido.

Restricciones de recursos.

Un recurso es cualquier dispositivo de entrada/salida o estructura de software que puede ser utilizada por el proceso. Típicamente, un proceso puede ser una estructura de datos, un conjunto de variables, un área de memoria, un archivo, una porción de un programa o un dispositivo periférico. Un recurso que es dedicado exclusivamente a un proceso en particular se dice que es un recurso privado, mientras que un recurso que puede ser utilizado por dos o más tareas es llamado recurso compartido.

Para mantener la consistencia de datos en la mayoría de los recursos compartidos, no se permite el acceso simultáneo por dos o más tareas a estos datos, sino que se requiere del cumplimiento de la condición de exclusión mutua

entre las tareas que compiten por este recurso. Supongamos que R es un recurso exclusivo compartido por las tareas T_a y T_b. Si α es la operación realizada por T_a sobre R, y β es la operación realizada sobre R por T_b, entonces α y β nunca deben ejecutarse al mismo tiempo. El conjunto de líneas de código que se ejecutan bajo restricciones de exclusión mutua se le conoce como sección o región critica. En este caso deben de implementarse mecanismos de exclusión mutua,
entre las tareas que compiten por este recurso, tales como la entropía u otras técnicas de discriminación [66].

Para asegurar el acceso secuencial sobre los recursos compartidos, los sistemas operativos normalmente proveen mecanismos de sincronización (tales como monitores, semáforos, etc.) que pueden ser usados por las tareas para crear regiones críticas.

APENDICE B

Multiprocesadores

En la última década las aplicaciones que se ejecutan sobre ambientes de procesamiento paralelo han ido en aumento, originado principalmente por que ha demostrado ser una herramienta de ingeniería útil para los problemas de automatización industrial, simulación de sistemas, problemas complejos de tiempo real, etc. Esto es debido principalmente al enorme progreso en la tecnología de computadoras paralelas, herramientas de programación paralela y de los algoritmos de procesamiento paralelo.

Existen dos estrategias para la planificación de tareas de tiempo real en un sistema con multiprocesadores. Un esquema global (llamado también no-particionado) cada tarea de tiempo real puede ser ejecutada en un procesador diferente. En contraste, en un esquema particionado todas las instancias de una tarea son ejecutadas en un sólo procesador. El esquema particionado tiene algunas ventajas sobre el esquema global. En primer lugar, el esquema particionado es menos complejo por que no introduce sobrecarga al planificador de multiprocesamiento, ya que solamente son asignadas las tareas al procesador una sola vez al principio de la ejecución. En segundo lugar, un algoritmo ya conocido de planificación puede ser usado para cada procesador.

El desempeño del esquema particionado está determinado por dos factores; el algoritmo de asignación de las tareas, el cual se encarga de distribuir las tareas a los procesadores y un algoritmo de planificación, el cual determina el orden de ejecución de las tareas en cada procesador. El objetivo de un algoritmo de asignación de tareas será el tener una planificación factible (es decir, que todas las tareas conozcan sus plazos de respuesta en tiempo de ejecución) para cada

procesador con el menor número de procesadores. Sin embargo, el problema de encontrar una asignación óptima para los algoritmos de prioridad fija, como el RM (Rate Monotonic), así como para la planificación con prioridades dinámicas, como el EDD (Earliest Due Date), es NP-duro. Esto lo demostraron Leung y Whitehead en [37].

Entre los dos métodos, el particionado es en el que se ha concentrado la investigación, principalmente porque es fácil de usar para garantizar la planificabilidad en tiempo de ejecución. Muchos algoritmos heurísticos para este método se han propuesto, por ejemplo, en [59], fueron presentados dos esquemas de asignación, el RMNF (Rate Monotonic Next-Fit) y el RMFF (Rate Monotonic First-Fit) basándose en el algoritmo heurístico bin-packing. En ambos esquemas, las tareas son ordenadas en orden decreciente a sus periodos, antes de ser asignadas. Las tareas son asignadas al procesador hasta que es violada la condición de planificabilidad, en tal caso, el procesador es marcado como lleno y es seleccionado un nuevo procesador. El RMFF primero trata de acomodar una tarea en un procesador ya marcado como lleno antes de ser asignada la tarea al procesador actual. El método FFDUF (First-Fit Decreasing-Utilization Factor) es una variación del esquema heuristico first-fit. Aquí las tareas son ordenadas en base a su factor de carga [56]. En [77] es usado el algoritmo RMBF (Rate Monotonic Best-Fit), el cual es parecido al RMFF ya que asigna tareas a los procesadores que han sido marcados como llenos. Sin embargo, los procesadores llenos son inspeccionados en un orden específico. Como en [59], las tareas son ordenadas por sus periodos. Todos ellos basados en el algoritmo bin-packing, exhibiendo en promedio un buen desempeño.

Burchard y otros, proponen otros dos algoritmos [1], el RMST (Rate Monotonic Small Tasks) particiona cada tarea en un número pequeño de sub-tareas, las cuales contienen tareas periódicas simples. Una buena aproximación para asignar la tarea, es particionar primero el conjunto de tareas periódicas en sub-conjuntos.

Cada sub-conjunto de tareas periódicas es planificable todo el tiempo que la utilización total del sub-conjunto no sea mayor a uno. Si más de un sub-conjunto es asignado a un procesador, su utilización de planificación es una función del número de sub-conjuntos y no del número de tareas. El otro algoritmo es el RMGT (Rate Monitonic General Task) el cual primero particiona toda las tareas periódicas dentro de dos sub-conjuntos acorde a sus utilizaciones. Las tareas cuya utilización es igual o menor a 1/3 están en un sub-conjunto. Esas tareas son primero asignadas a los procesadores acorde al algoritmo RMST. Después las tareas más largas cuya utilización es más grande de 1/3 son asignadas por first-fit a los procesadores los cuales tiene al menos una tarea asignada por el algoritmo RMST. El método de análisis consume tiempo ya que se utiliza para checar si una tarea es grande y puede ser asignada a un procesador que ya tenga una tarea.

El método no-particionado no ha recibido tanta atención, debido a las siguientes limitaciones: Primero, actualmente no existe una prueba eficiente de planificabilidad. Se conoce una prueba necesaria y suficiente de planificabilidad con una complejidad de tiempo exponencial [36]. La complejidad fue reducida a polinomial con una prueba de planificabilidad suficiente [9, 17, 40, 60] o con una complejidad de tiempo seudo-polinomial [40]. Pero en [17], la prueba llega a ser pesimista cuando el número de las tareas crece y en [9, 40, 60] llega a ser pesimista cuando el número de procesadores se incrementa. Segundo, no se ha encontrado un esquema de asignación de prioridades óptimo. La asignación de prioridades con Rate Monotonic (RM) [18], es óptimo en un sistema con un procesador, pero no es óptimo para multiprocesadores usando el método no-particionado [37, 59]. Un peor caso se presenta cuando un conjunto de tareas con una muy baja utilización no puede ser planificable con RM [59], conocido como el efecto Dhall´s.

APENDICE C

Servidor de Láminas

Los servidores de láminas o servidores blade están diseñados para su montaje en bastidores al igual que otros servidores. La novedad estriba en que los primeros pueden compactarse en un espacio más pequeño gracias a sus principios de diseño.

Cada servidor blade es una delgada "tarjeta" que contiene únicamente microprocesador, memoria y buses. Es decir, no son directamente utilizables ya que no disponen de fuente de alimentación ni tarjetas de comunicaciones.

Estos elementos más voluminosos se desplazan a un chasis que se monta en el bastidor ocupando únicamente de cuatro (4U) a seis alturas (6U). Cada chasis puede albergar del orden de dieciséis "tarjetas" o servidores blade (según fabricante). El chasis lleva integrados los siguientes elementos, que son compartidos por todos los servidores:

- Fuente de alimentación: redundante y hot-plug.
- Ventiladores o elementos de refrigeración.
- Conmutador de red redundante con el cableado ya hecho, lo que simplifica su instalación.
- Interfaces de almacenamiento. En particular, es habitual el uso de redes SAN (Storage Area Network) de almacenamiento.

Además, estos servidores suelen incluir utilidades software para su despliegue automático. Por ejemplo, son capaces de arrancar desde una imagen del sistema

operativo almacenada en disco. Es posible arrancar una u otra imagen según la hora del día o la carga de trabajo, etc.

Ventajas

- Son más baratos, ya que requieren menos electrónica y fuentes de alimentación para el mismo número de servidores. También consumen menos energía.
- Ocupan menos espacio, debido a que es posible ubicar dieciséis servidores donde habitualmente solo caben cuatro.
- Son más simples de operar, ya que eliminan la complejidad del cableado y se pueden gestionar remotamente.
- Son menos propensos a fallos ya que cada servidor blade no contiene elementos mecánicos.
- Son más versátiles, debido a que es posible añadir y quitar servidores sin detener el servicio.

Usos

Los servidores blade son aptos para los mismos usos que cualquier otro servidor. No obstante, son especialmente ventajosos para instalaciones de entornos de virtualización, en cluster y para web hosting.

APENDICE D

Procesamiento de imágenes

De los muchos métodos disponibles para el tratamiento de imágenes, los dos más populares son la Transformada Coseno Discreta (DCT) utilizado en el formato JPEG, y la Transformada Wavelet Discreta (DWT) utilizado en el nuevo formato JPEG 2000. El DWT difiere entre el tradicional DCT de varias maneras fundamentales. El DCT opera dividiendo la imagen en bloques de 8x8 que son transformados independientemente [55]. A través de este proceso de transformación, la energía compactada del DCT asegura qué los datos originales están concentrados en sólo unos pocos coeficientes, los cuales son usados más adelante para la cuantificación y codificación [63]. Desafortunadamente, la naturaleza rígida de los bloques de 8x8 del DCT lo hace particularmente susceptible a introducir ruido extraño alrededor de los bordes en una imagen. Este es el "efecto halo" visto en imágenes comprimidas transmitidas en la red.

En contraste con el DCT, el DWT opera sobre toda la imagen entera, eliminando el ruido extraño, igual a los causados por bloques 8x8 del DCT. Como el DCT, la transformada wavelet fundamental es completamente reversible, significando que si la transformada hacia delante e inversa son aplicados en secuencia, los datos resultantes serán idénticos a los originales. Además, el DWT está basado en la codificación por sub-bandas donde la imagen es analizada y filtrada para producir componentes de la imagen a diferentes frecuencias cada sub-banda [71]. Esto produce una compactación significativa de la energía que posteriormente es explotada en el proceso de comprensión. La naturaleza bidimensional de los wavelet resulta en la visualización de la imagen siendo dividida en cuatro cada vez que pasa la transformada wavelet. Un efecto importante en esta transformación es

que los cuadrantes de alta frecuencia en la imagen contienen datos equivalentes [67].

En el campo de wavelets, la wavelet Haar es tradicionalmente usada para una compresión de imágenes rudimentaria a causa de la simplicidad algorítmica y la baja complejidad computacional, pues su diseño está basado en enteros [70]. Esta transformación está representada como:

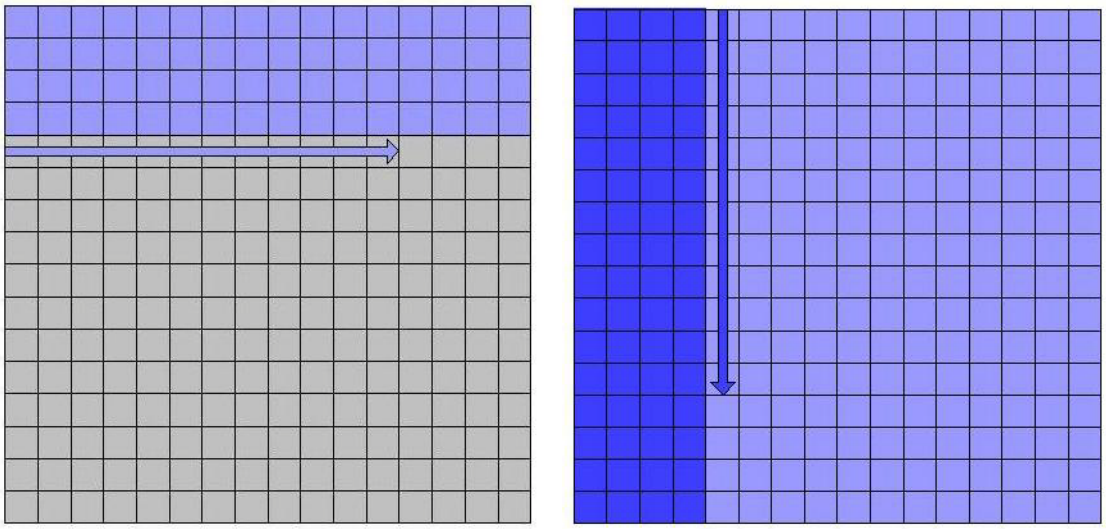
$$h_n = \begin{cases} \frac{1}{2} & n = 0,1 \\ 0 & otro\ caso \end{cases}$$

$$g_n = \begin{cases} \frac{1}{2} & n = 0 \\ -\frac{1}{2} & n = 1 \\ 0 & otro\ caso \end{cases} \qquad (1)$$

Donde h_n es la función de escalamiento y g_n es la transformada wavelet ver ecuación (1). La aplicación de la transformada wavelet a una sola imagen se puede apreciar en la figura D.2 en donde se simplifica el algoritmo.

En la figura D.1 se aprecia cómo se realizaría la transformada wavelet, esta podría ser por filas o columnas. Cuando se realiza por filas, figura D.1.a se aplica la transformada a toda la imagen (píxel x píxel) o sea fila por fila. Y cuando se realiza la transformada por columna figura D.1.b seria realizar (píxel por píxel) columna por columna.

En la figura D.2 la wavelet fue aplicada sólo en forma horizontal, así que cuando el wavelet es aplicado a través de su proceso de filtración producen dos tipos de coeficientes: el coeficiente de la función de escalamiento (baja frecuencia) y los coeficientes wavelet (alta frecuencia). De la aplicación de la transformada Haar, es evidente que los coeficientes de la función de escalamiento es simplemente el promedio de dos valores de píxeles consecutivos, mientras los coeficientes wavelet correspondientes, es la diferencia entre los mismos dos valores de píxeles. Los coeficientes de la función de escalamiento parecen contener todos los datos de la imagen, mientras que los coeficientes wavelet aparecen en negro.



a) Transformada por fila. b) Transformada por columna.

Figura D.1. Transformada por fila y por columna.

En la figura D.3 se observa el proceso que sigue la transformada wavelet, se empieza por una imagen original, cuando se realiza la transformada por fila, la imagen se divide en dos mitades, la primera mitad corresponde a la función de escalamiento y la segunda mitad corresponde a la transformada wavelet, cabe mencionar que a cada mitad se le llama sub-banda.

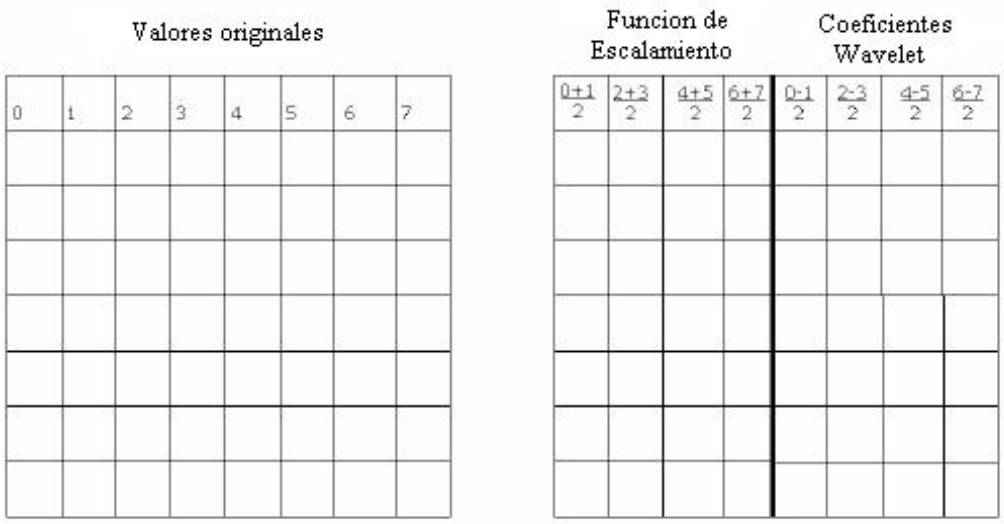


Figura D.2. Aplicación de la transformada wavelet.

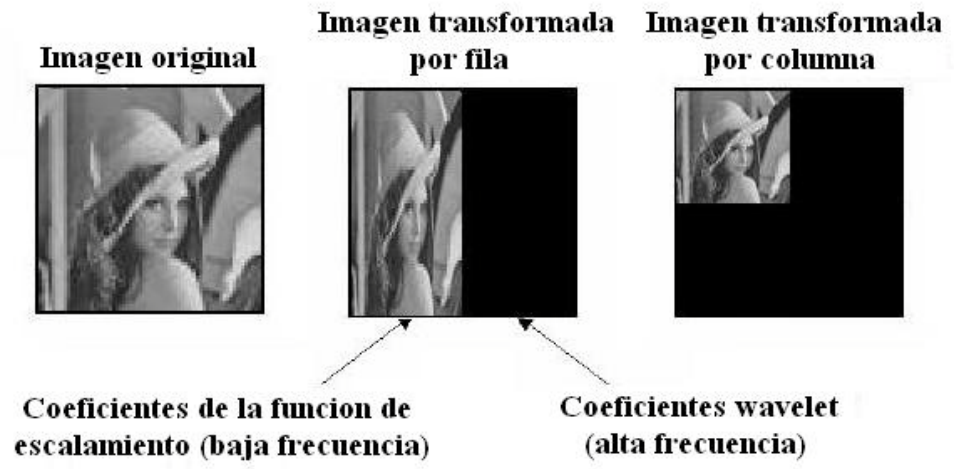


Figura D.3. Transformada en el primer nivel.

Cuando se realiza la siguiente transformada por columna esta imagen se divide en cuatro partes iguales o sub-bandas, el primer cuadrante o sub-banda corresponde a la función de escalamiento y las otras tres sub-bandas a la transformada wavelet, en este primer paso se tienen cuatro sub-bandas. Esta primera etapa corresponde al aplicar la transformada en el primer nivel de la transformada. Este

proceso se repite sobre la función escalamiento para el segundo nivel y así sucesivamente. Al final la imagen se ve como se muestra en la figura D.4.

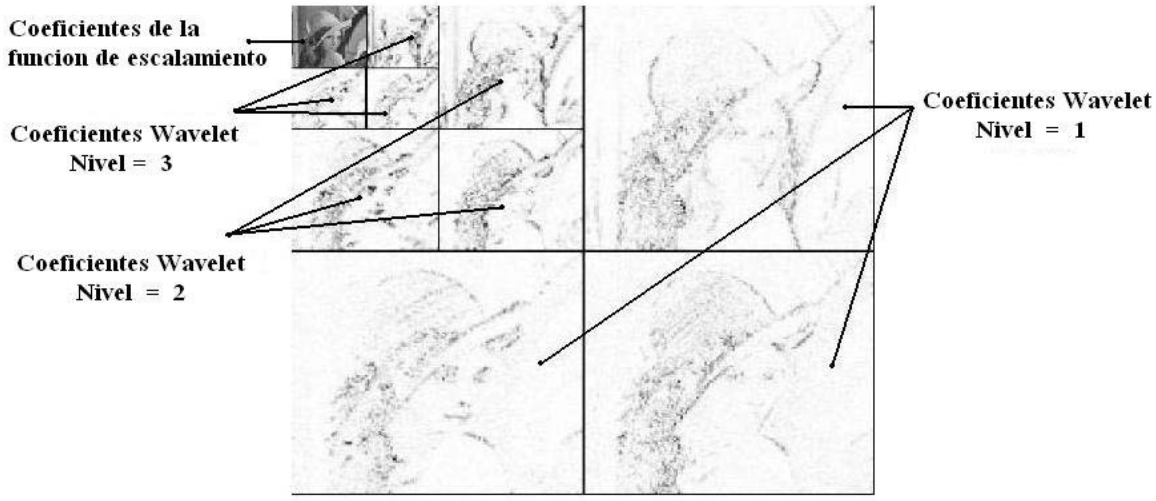


Figura D.4. Transformada wavelet para tres niveles.

La transformada wavelet se repite progresivamente la información de baja frecuencia, algunos datos de la imagen original se repiten más de una vez. Así que los coeficientes wavelet de baja frecuencia son transformados por un wavelet de diferente amplitud y duración que los coeficientes wavelet de más alta frecuencia. Así cada sub-banda fue generada por una función wavelet diferente como se muestra en la figura D.4.

GLOSARIO DE TERMINOS

Algoritmo heurístico: Es un procedimiento de búsqueda de soluciones casi óptimas a un costo computacional razonable, sin ser capaz de garantizar la factibilidad de las soluciones empleadas ni determinar a qué distancia de la solución óptima nos encontramos.

Algoritmo óptimo: Es un algoritmo de planificación que minimiza o maximiza alguna función de costo definida en base al conjunto de tareas.

Carga: El tiempo de cómputo demandado por un conjunto de tareas en un intervalo, dividido por la longitud del intervalo.

Complejidad computacional de un problema: Es una medida de los recursos computacionales (generalmente el tiempo) requeridos para resolver el problema.

Conjunto de tareas planificable: Un conjunto de tareas para los cuales existe una planificación factible.

Evento: Una ocurrencia que requiere una reacción del sistema.

Factor de utilización: La fracción de tiempo del procesador utilizado por un conjunto de tareas periódicas.

Hiperperíodo: El intervalo de tiempo mínimo después del cual la planificación se repite. Para un conjunto de tareas periódicas, representa el menor común múltiplo de todos los períodos.

Máquina determinista: Estriba en que para una misma entrada, el resultado de aplicar repetidas veces un algoritmo, siempre es el mismo.

Máquina no determinista: Es equivalente a decir que pueden obtenerse resultados diversos para una misma entrada.

Período: El intervalo de tiempo entre la activación de dos intervalos consecutivos de una tarea periódica.

Planificación: Una actividad del kernel que determina el orden en el cual los trabajos concurrentes son ejecutados en un procesador.

Planificación estática: Un método en el cual todas las decisiones de planificación son precalculadas fuera de línea y los trabajos son ejecutados en una forma predeterminada.

Planificación dinámica: Un método de planificación en el cual todos los trabajos activos son reordenados cada vez que una nueva tarea entra al sistema o un nuevo evento ocurre.

Planificación factible: Una planificación en la cual todas las tareas de tiempo real son ejecutadas dentro de sus plazos de respuesta y todas sus restricciones (si las hay), son conocidas.

Plazo de respuesta: El tiempo dentro del cual una tarea de tiempo real deberá completar su ejecución.

Problema P: Están formados por todos aquellos problemas de decisión para los cuales se tiene un algoritmo de solución que se ejecuta en tiempo polinomial dentro de una máquina determinista.

Problemas de decisión: Son aquellos donde se busca una respuesta de “si” o “no”.

Problemas de optimización: Son aquellos en los cuales se busca minimizar o maximizar (es decir, optimizar) el valor de una solución en un grupo de soluciones generadas para una entrada especifica (instancia).

Problemas NP: Están formados por todos aquellos problemas de decisión para los cuales existe un algoritmo de solución que se ejecuta en tiempo polinomial dentro de una máquina no determinista. Dicho de otro modo, no se ha encontrado un algoritmo determinista que lo resuelva en tiempo polinomial.

Predecible: Una propiedad importante de un sistema de tiempo real que permite anticipar la consecuencia de alguna decisión de planificación.

Restricciones de precedencia: La relación de dependencia entre dos o más tareas que especifican que una tarea puede comenzar su ejecución después de que se complete su ejecución de una o más tareas (llamados predecesores).

Sobrecarga: Es la condición de carga del procesador, tal que el tiempo de cómputo demandado por las tareas en un cierto intervalo excede el tiempo del procesador disponible en el mismo intervalo.

Tarea periódica: Un tipo de tarea que consiste de una secuencia de trabajos idénticos (instancias), activados en intervalos regulares.

Tiempo de procesamiento: La cantidad de tiempo requerido por el procesador para ejecutar un trabajo sin interrupción. También es llamado “tiempo de cómputo” o “tiempo de servicio”.

Tiempo polinomial: Se puede decir que un algoritmo tiene complejidad polinomial, o se ejecuta en tiempo polinomial, si tiene un orden $O(n^k)$. Estos algoritmos se dice que son algoritmos eficientes y los problemas que se resuelven con estos algoritmos se dice que son problemas tratables.

Trabajo: Un cálculo en el cual las operaciones, en la ausencia de otras activaciones, son ejecutadas secuencialmente por el procesador hasta que son completadas.