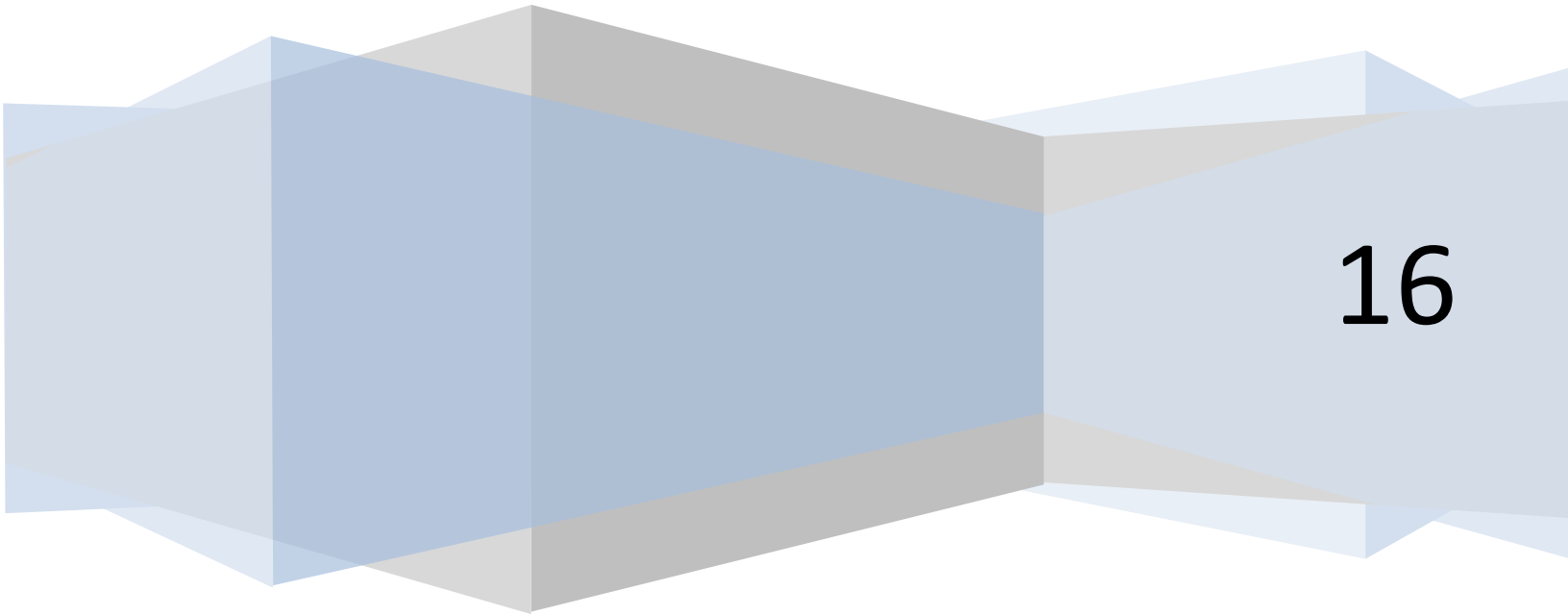
Universidad Valladolid
Facultad de Ciencias

Ejemplo de programación en paralelo

De diferentes algoritmos estadísticos

Autor: Daniel Ballesteros Álvarez

Tutor : Eusebio Arenal Gutiérrez



16

Contenido

1. Introdución

Este trabajo pretende mostrar cómo ha cambiado recientemente el mundo de la computación ,esencial para el cálculo estadístico, y como poder abordar dichos cambios pudiendo aprovechar la enorme potencia extra que disponemos actualmente de forma, que el usuario del programa estadístico GNU R, no tenga la necesidad de aprender nuevos paradigmas de programación, lenguajes.. simplemente aplicando unos pequeños “trucos”, bien pudiera ser llamados “recetas” con las que una vez aplicadas al instalar dicho programa (o en cualquier momento posterior) pueda hacer uso de toda esa potencia extra, teniendo que enfocarse simplemente en su tareas .

También hago un recorrido a como la estadística y la informática están totalmente relacionas ayudando a ampliarse mutuamente, como ha ido desarrollándose dicha evolución y una breve pero interesante explicación de por qué de estos cambios.

2. Relación estadística e informática:

2.1.La informática como herramienta para la estadística

La forma en que se entiende en la actualidad la estadística es relativamente nueva. Hasta hace no mucho tiempo casi todo el desarrollo se ha producido de manera teórica debido a la falta de recursos de cálculo para poder aplicar dichas técnicas de forma realmente práctica.

Al aparecer los primeros computadores (años 60-70) de uso solo en las grandes compañías, universidades, hicieron que la estadística dejara de ser algo eminentemente teórico para empezar a utilizarse de forma práctica.

Al poco tiempo (principio de los 80) aparecieron los primeros computadores de propósito general destinados a un público más amplio, pero debido a su potencia su uso estaba restringido principalmente a usarlos como una gran calculadora y juegos sencillos.

Inicialmente la potencia de estos computadores estaba muy restringida, pero en no mucho tiempo se ha producido un avance progresivo de los recursos que proporcionan, llegando al límite físico del diseño de los microchips de propósito general.

Gracias estos avances se han podido utilizar en estos ordenadores de propósito general técnicas estadísticas que en años anteriores ni si quiera podían ser utilizados en grandes ordenadores corporativos, a la vez que se han desarrollado otras nuevas.

Esto hace que desde finales de los 80 este íntimamente ligado el uso de los computadores con el tratamiento y análisis de datos.

A partir de ahora se denominará computador a las maquinas que predominan en nuestras casas basadas en la arquitectura IBM PC compatible x86, cuya expansión comenzó principalmente con las gamas 286 y 386.

Hasta ahora los procesadores que montan dichas maquinas han estado diseñados en base a la arquitectura que desarrollo Von Newman , pero debido al incremento tan enorme que se ha producido en su potencia , actualmente se está llegando al límite físico en el diseño de los procesadores, lo que hará cambiar el tipo de arquitectura hacia otras mas complejas.
Ante los problemas físicos, entre los que cabe destacar el calor que se genera al alcanzar frecuencias tan altas, los fabricantes han optado por empezar a utilizar nuevas soluciones para poder seguir creciendo en potencia.

La primera solución fue añadir un coprocesador matemático junto a la CPU principal. De echo los computadores 486 eran meros 386 en los que se incorporaba de serie dentro del mismo chip el coprocesador matemático. Años después se añadieron instrucciones específicas para acelerarlos cálculos: MMX, SSE,... como ocurrió en los últimos Pentium en las siguientes generaciones Pentium 2, Pentium 3 y Pentium 4 .

A partir del año 2000, cuando se llegó a la frecuencia de un Gigahercio (tras varios años anunciándolo bajo el nombre clave “Merced”) y los primeros chip de propósito general de 64 bits (Athlon 64), empezaron a verse los problemas físicos comentados anteriormente. En este momento comenzó la tendencia de insertar varios procesadores de propósito general en el mismo encapsulado en vez de incrementar la frecuencia del procesador. Este hecho es lo que nos lleva a tener que programar en paralelo en vez de la clásica programación secuencial poder aprovechar los nuevos recursos.

Antes de continuar con la exposición de los recursos actualmente disponibles, es importante dejar claro que son y en que se diferencian la multitarea y el paralelismo, ya que tiende a confundirse.

Pese a ser dos conceptos muy diferentes es habitual utilizar el término multitarea de una manera muy general y poco precisa, lo que nos llevaba a creer que es un sinónimo de paralelismo: lo cual es falso. La manera correcta de denominar la multitarea es *concurrencia*;

La multitarea (nacida junto a los grandes ordenadores corporativos y en la era 486-Pentium en los ordenadores de propósito general) consiste en dividir el tiempo de cpu en fracciones (llamados quantum) donde se ejecutan fragmentos de los programas, pasando de un programa a otro, produciendo la ilusión de que se están ejecutando a la vez. Y aquí es donde reside el problema conceptual: en un solo procesador solo se puede ejecutar una tarea a la vez .

Si que existen un tipo de procesador, denominados procesadores vectoriales que suelen ser incluidos en maquinas muy especificas Un ejemplo es el procesador CELL construido por varios fabricantes, entre los que se encuentra IBM, siendo conocido por su inclusión en la videoconsola PlayStation 3. Cabe destacar la complejidad de programar para este tipo de procesadores, siendo este el mayor distintivo de esta máquina con respecto al resto por parte de los programadores especializados

El paralelismo, que también es un concepto antiguo, consiste en ejecutar dos o mas aplicaciones simultáneamente en diferentes procesadores. Para ello debemos disponer de varios procesadores, ya sean en la misma máquina (paralelismo multiprocesador) o en maquinas diferentes unidas mediante una red (paralelismo en redes).

- Paralelismo multiprocesador

Empezó utilizándose con placas base que permitían tener varios procesadores a la vez compartiendo el resto de recursos de la maquina. Sobre esto se tratará más adelante, ya que para su programación se usa el paradigma denominado “memoria compartida”. Este concepto es importante ya que antaño solo se usaba en maquinas caras .

Basándose en esta idea nacieron los denominados procesadores multinúcleo, siendo cada núcleo un procesador de propósito general. La principal diferencia respecto a las placas base con varios procesadores es que aquí están contenidos en el mismo encapsulado, como si fueran un único chip, haciendo que los datos se compartan más rápidamente.

Comentar que en algunos campos, entré los que se incluye supercomputación, Mainframe.., siguen usando placas base que soportan varios procesadores, y a su vez estos llevan varios núcleos.

- Paralelismo basado en redes

La otra alternativa clásica para el paralelismo, la más antigua, es usar el poder computacional de varias maquinas conectadas en red para hacer los cálculos. Puede hacerse en redes comunes con ordenadores cada uno con su sistema operativo o bien los denominados clusters (una red de ordenadores un solo sistema operativo gobernándolos a todos. De nuevo pueden usarse todas las combinaciones de lo anterior.

Como última opción está el aprovechar los nucleos denominados kernels incluidos en las GPU o tarjetas aceleradoras de video.

Estas tarjetas se diseñaron para liberar al procesador central de los complejos cálculos necesarios para la creación y renderizado de entornos 3D.

Han ido evolucionando hasta poder ser programables, permitiendo usar sus procesadores, que realizan pocas tareas pero de manera muy rápida, debido a su naturaleza vectorial.

Para concluir este apartado, considero interesante realizar una análisis de un [i]artículo[1] publicado en 1996,.en él se explicaba como la aparición de nuevos contenidos, que denominaron "dinámicos", desembocía en cambios fundamentales en el diseño de los procesadores, y detallaban como serian algunos de esos cambios.

Comienzan distinguiendo las aplicaciones que necesitan grandes cantidades de datos de las aplicaciones más tradicionales, categorizando a las primeras como ejemplos de lo que llamaron "procesamiento dinámico". Esto significa básicamente que el flujo de operaciones a penas cambia y si lo hace es muy lentamente, pero el flujo de datos cambia constantemente. Para concretar esto más: los programas multimedia y programas científicos procesan enormes cantidades de datos, pero las operaciones que hacen sobre esos bloques enormes de datos son muy pocas.

El caso más común es cuando hay un pequeño bucle que recorre una o incluso una series de matrices enormes muchas veces.

En contraste a esto están los programas más "tradicionales", o aplicaciones de "procesado estático" como un procesador de textos, el cual usa muchos trozos diferentes de código (la parte

[1]"How Multimedia Workloads Will Change Processor Design."(Como las cargas de multimedia cambiaran los procesadores) y sus autores: Diefendorff y Pradeep K. Dubey.

de los menus, ayudas, correctores, etc.) para operar en un simple archivo de datos (el documento).
En este tipo de aplicaciones, el flujo de datos suele ser lento y a penas cambia. En cambio el flujo de instrucciones/operaciones lo esta haciendo constantemente.

Tan pronto estas usando el auto corrector, para pasar inmediatamente a cambiar las Fuentes y cuando has terminado exportarlo a un formato diferente como pdf.

Los PC se diseñaros en base a este tipo de aplicaciones de "procesamiento estático" (procesadores de textos, hojas de cálculo...) en mente.

Con el paso de los años ha aparecido cambios significantes, particularmente con la aparición de hardware de propósito especial como las tarjetas graficas 3D y procesadores de sonido.

El añadir dicho hardware extra, sin embargo, representa un intento de "rellenar con vino nuevo una botella vieja".

También predicen que todas esas capacidades de procesamiento extra llegaran a ser integradas dentro del propio procesador y ese hardware llegara a estar obsoleto. Además dichos diseños se caracterizaran por unas rutas de datos extremadamente amplias entre los componentes empotrados en el mismo chip.

Actualmente estamos en un punto medio ya que los procesadores dedicados de las GPU ya están empezando a ser incluidos dentro del encapsulado de ciertos procesadores de propósito general.

Lo que ellos dicen, en resumen, es que las aplicaciones estáticas, tienen cierto grado de paralelismo a nivel de instrucción/operaciones que realizan, pero muy poco respecto a los datos (que apenas cambian)

En contraste, el procesado de medios dinámicos exhibe muy pocos cambios en las operaciones que realizan pero cantidades de paralelismo respecto a los datos.

De nuevo, el problema es como hacer uso de los recursos que se han añadido: los programadores de PC no han tenido que lidiar con dicha cantidad de datos y procesamiento hasta ahora

- Las tarjetas graficas que actualmente están en el mercado, con cientos de procesadores dedicados (mucho más simples que los procesadores de ámbito general) pueden realizar los cálculos de una forma muchísimo más rápido que los procesadores más potentes de uso general.

- Es en estas donde más dinero están invirtiendo las compañías, haciendo que estas estén experimentando un incremento enorme en ciclos muy pequeños de tiempo.

Desgraciadamente la posible paralelización de los cálculos ha implicado que hay que cambiar algunas cosas a la hora de programar para poder sacar el máximo rendimiento a estas maquinas.

Esto suele hacerse usando librerías especificas o por parte del programador.

Muchos programas ya se han adaptado pero hay otros como GNU R que todavía sigue anclado a la programación secuencial "clásica".

Afortunadamente hay varias maneras para ayudar a utilizar esto y es en lo que en el presente trabajo nos vamos a enfocar, pudiendo servir de “guía” para que cualquier persona que vaya a usar GNU R (u otro programa) pueda beneficiarse de dichos avances sin necesidad de grandes cambios.
Por supuesto se profundiza en los temas tratados para quien tenga interés en aprender más o simplemente entender cómo funcionan.

2.2. La estadística como herramienta para la informática

A medida que ha aumentado la potencia, también ha ido aumentando la complejidad de los sistemas operativos y programas programados para ellas.

Al diseñar los sistemas operativos para estas maquinas más complejas (El SO es el encargado de organizar y distribuir los recursos y tareas), aparecieron problemas sobre el reparto y planificación a los cuales se han buscado ciertas soluciones, muchas de ellas son modelos matemáticos sacados del ámbito de la estadística e investigación operativa.

Un ejemplo es la planificador de tareas y recursos que hace un sistema operativo.

-Otro ejemplo es utilizar la estadística para comprobar si estos resultados son satisfactorios en el aprovechamiento de los recursos. Para esto se usan las técnicas habituales (regresiones, análisis de datos, diseño de experimentos..) por lo que no entrare en ello

Algunas técnicas importantes aplicadas a la resolución de conflictos de acceso a memoria y al análisis de rendimiento son

2.2.1. Redes de Petri

Es una de las primeras técnicas y la más sencilla de las tres propuestas. Son modelos gráficos con los que se representa el comportamiento de los computadores entre otras cosas. Hare mención de ellas y luego explicare el modelo más completo, ya que el resto son simplificaciones de este.

Esta técnica ha ido evolucionando

- Redes Petri Temporizadas: primero añadiendo la representación del tiempo como variable. De forma simplificada es el uso de grafos

- Modelos deterministas: si las variables, incluida el tiempo, tomaban valores fijos. Es una evolución de lo anterior donde se incluyen pesos en los grafos.

- Redes estocásticas de Petri o SPN: En caso que el numero de variables sea alto o que algún parámetro aparezca sin obedecer a ley alguna (por ejemplo, una petición de memoria ocurre de manera aleatoria).

Una SPN se compone de los parámetros P, T, A, M y G, representando cada uno de ellos

P: Representa los estados del sistema mediante círculos (nodos)

T: Representa mediante barras, las transiciones que ocurren

A: Representa mediante flechas, los arcos que unen P con

M: Son los indicadores del estado que existe en un momento determinado. Se representan por puntos negros en los estados correspondientes

G: Son unas relaciones asociadas con cada T y, según sea su valor, se conoce la frecuencia con la que se produce dicha transición T.

Ejemplo:

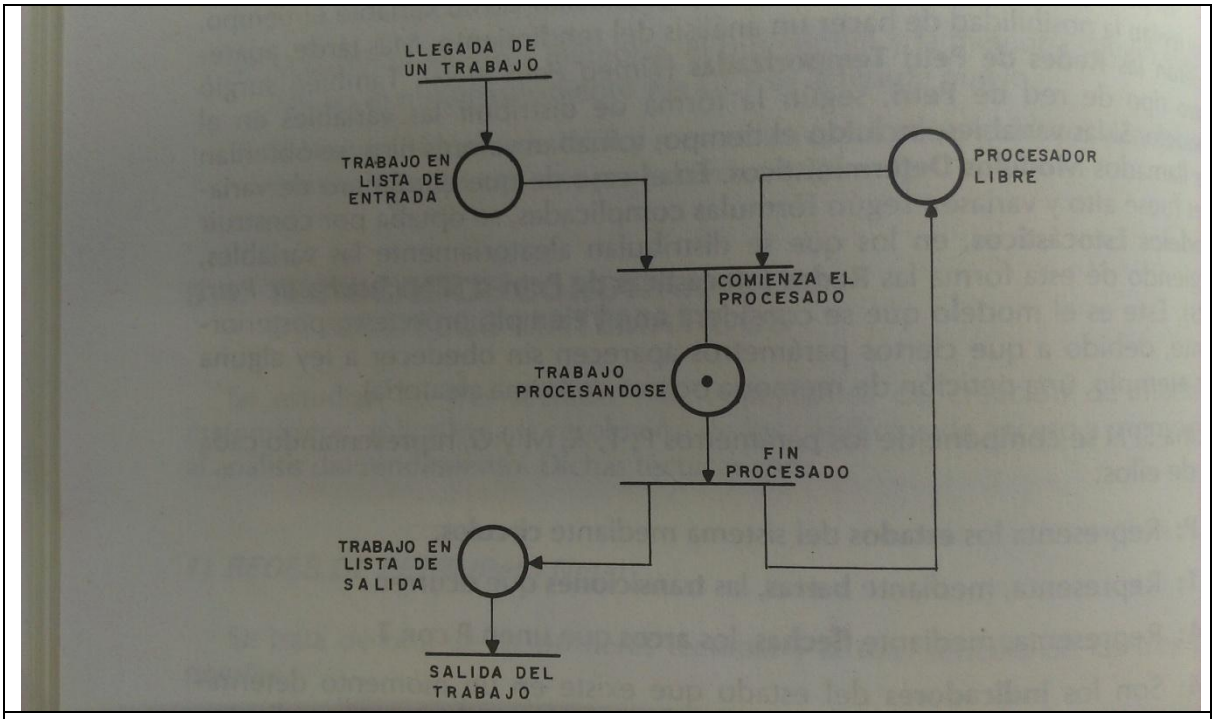
Sea un computador simple, donde la unidad básica de trabajo sea una tarea a realizar. Los estados del sistema pueden ser:

- Hay una tarea o trabajo en la lista de entrada
- Se está procesando un trabajo
- El procesador esta libre y disponible para un nuevo trabajo

Las transiciones correspondientes son:

- Entra un nuevo trabajo al sistema
- Empieza el procesado del trabajo
- Termina el procesado del trabajo
- El trabajo abandona el sistema

La representación de la SPN bajo estas condiciones, se muestra en la siguiente figura



Representación de la red Estocástica de Petri

Aunque el manejo de una SPN sea sencillo, su cálculo matemático es difícil, ya que se trata de expresar el modelo grafico en forma de ecuaciones.

Una solución bastante buena consiste en transformar una SPN en una cadena de Markov, cuya solución matemática es más fácil.

También se han desarrollado las Redes Estocásticas de Petri Generalizadas, que se obtienen considerando dos tipos de transiciones

- Inmediatas
- En el tiempo

Las inmediatas comienzan en el tiempo cero, una vez permitidas, mientras que las transiciones en el tiempo o temporizadas se encienden una vez transcurrido un tiempo aleatorio

2.2.2. Cadenas de markov

Esta técnica se ajusta más a un estudio matemático puro. Es la más desarrollada y la más aplicada hasta la actualidad en el diseño y estudio de los sistemas multiprocesadores.

El modelo gráfico de las cadenas de Markov tiene la ventaja sobre el de las SPN de poder reducir los estados simétricos y el inconveniente de que una variación de las condiciones iniciales obliga a diseñar un nuevo modelo.

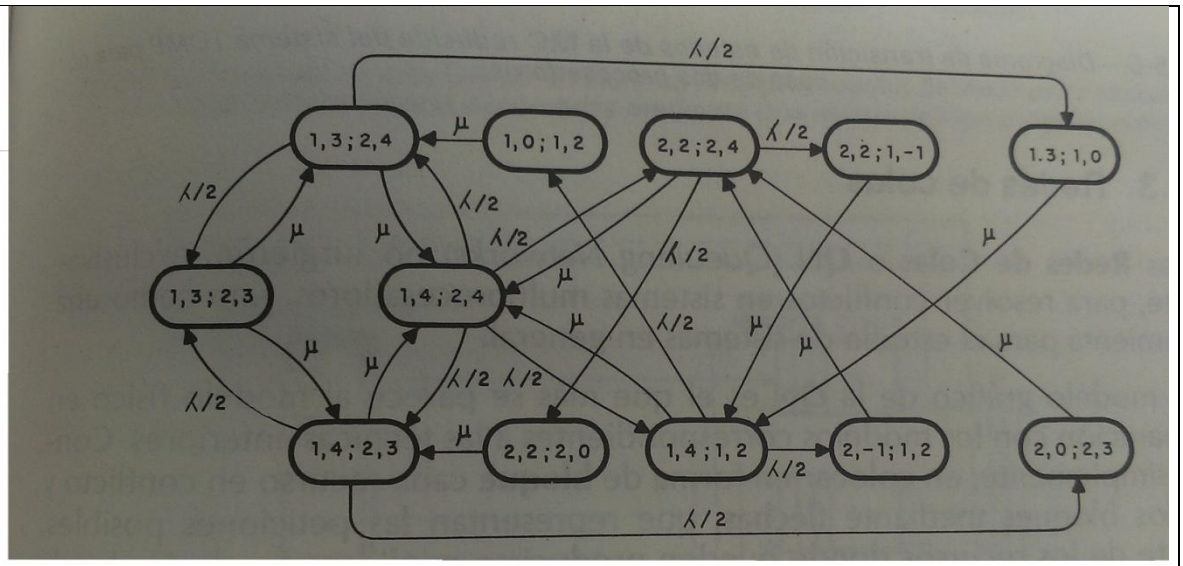
En general, una cadena de Markov consta de unos estados y matriz de transiciones definidos por n parámetros (k_1, k_2,.., k_n), donde cada parámetro k_i denota el estado o situación de un recurso de los que entran en conflicto.

Los estados y la matriz de transiciones son los mismos que en el anterior ejemplo.

La evaluación matemática se lleva a cabo:

1) Calculando la probabilidad de que se produzca una transición de un estado j a otro k. A dicha probabilidad se denomina probabilidad de transición p(j,k)

2) Desarrollando un algoritmo que cree una matriz de todas las probabilidades de transición que hay en el sistema. A partir de esta matriz se puede calcular el rendimiento del sistema, o sea, la probabilidad de que todos los procesadores funcionen a la vez, así como la probabilidad que diferentes grupos de procesadores estén funcionando



Diagramas transición de estados en cadena Markov caso sistema TOMP de dos procesadores

Una vez reducida

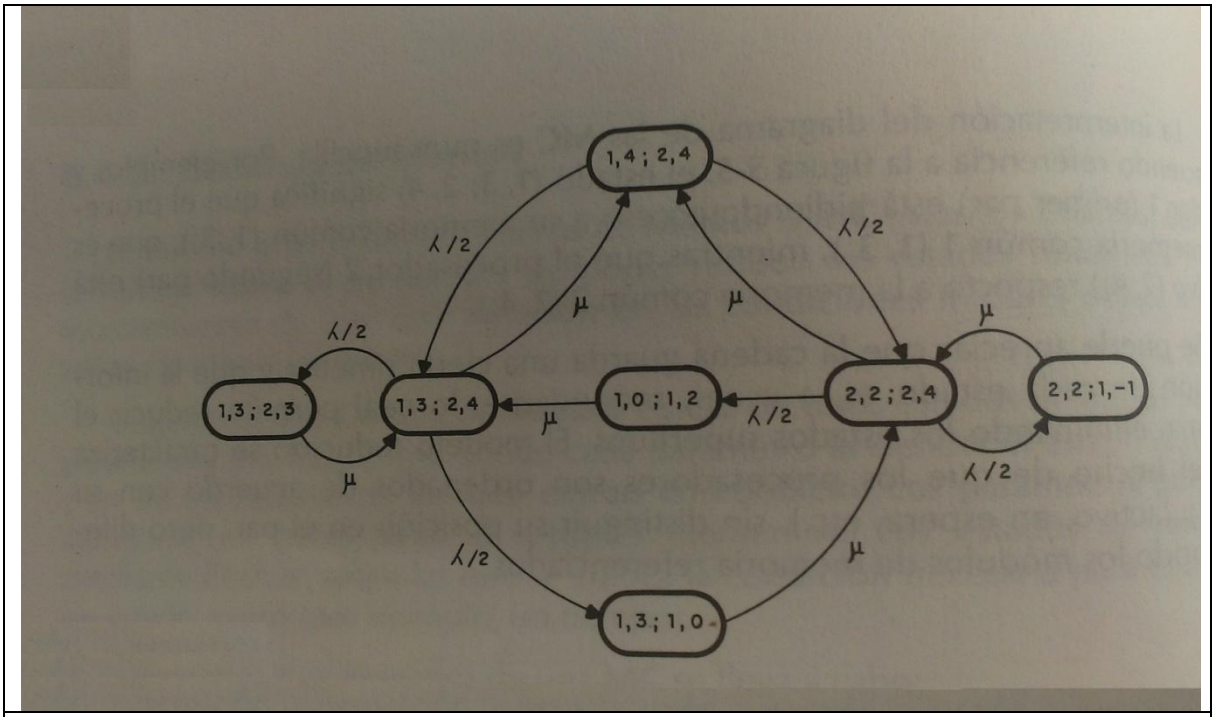
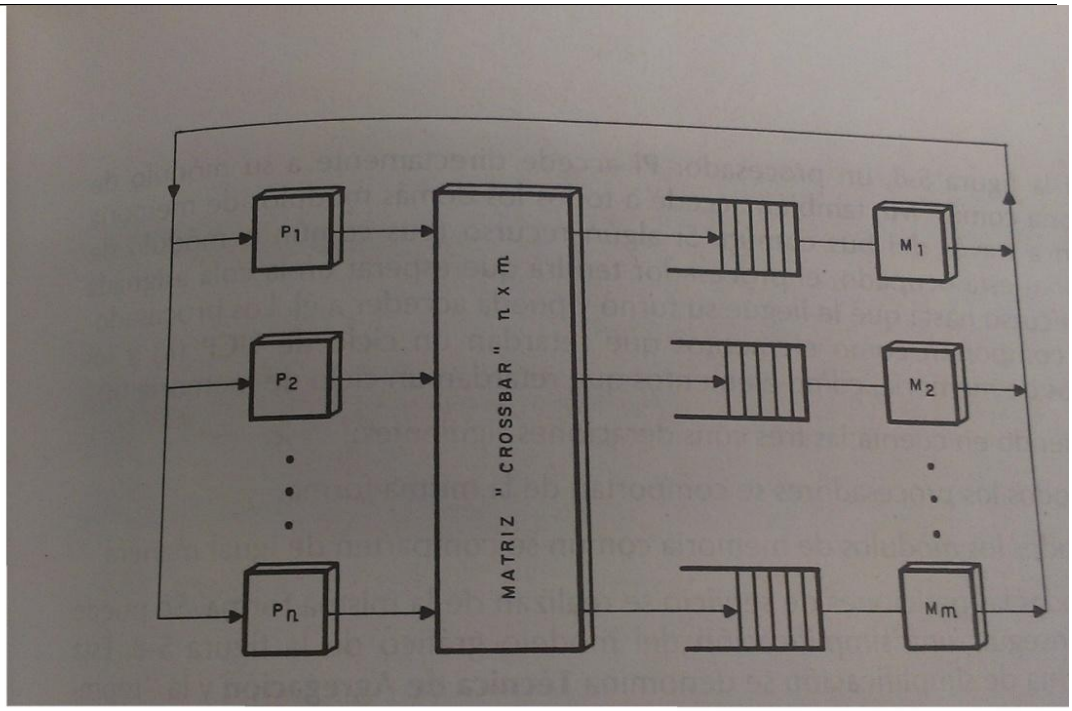


Diagrama de transición de estados de CM reducida para el caso del sistema TOMP de dos procesadores

2.2.3. Redes de espera o colas (Queuing Networks)

Se empezaron a aplicar como una herramienta para el estudio de sistemas en general. (Las anteriores se iniciaron en la planificación de memoria)

Este modelo es de los tres el que más se parece al modelo físico.



Estructura de una cola de espera de un sistema multiprocesador; formado por n procesadores y m módulos de memoria

Sobre este tema solo los he introducido y añadido las soluciones encontradas ya que no se trataran en el resto del trabajo pues que se salen del ámbito de este.

3 . Métodos de programación en paralelo :

Existe diferentes métodos para diseñar o convertir un algoritmo de forma que se este se ejecute en paralelo. Estos dependen del tipo de recursos disponibles a usar para procesar las diferentes tareas:
Puede ser un mismo ordenador con varios núcleos (Memoria compartida) , una red de ordenadores (Memoria distribuida) o usar los procesadores de una GPU (CP-GPU)

En este caso serán tarjetas NVIDIA por lo que la programación es en CUDA, que es la que esta actualmente mas desarrollada. Existe otro "lenguaje" denominado Opencl que ha nacido para poder aprovechar el poder de cálculo de las GPU y otros dispositivos (Como FPGA) sin importar de que marca sean y sin variar el código , pero su rendimiento actualmente es menor .

3.1.Memoria compartida : OpenMP

En el paradigma de programación paralela en sistemas de memoria compartida, se usa en sistemas donde todos los procesadores comparten la misma memoria, es decir, los datos escritos por un procesador son visibles al resto.(Los denominados núcleos insertados en un mismo encapsulado)

Por lo tanto, la comunicación entre ellos es muy rápida: solo hay que usar variables compartidas.

OpenMP es una librería para simplificar el trabajo de programar así, aprovechando el uso de los procesadores con varios núcleos que están un mismo encapsulado de una forma muy simple. .

NOTA: La frase anterior es una simplificación totalmente válida para cualquier maquina común. Existen otros casos mas complejos que se puede aplicar este paradigma y la librería mencionada, pero son maquinas con arquitecturas muy extrañas, algo que va mas allá del alcance de este trabajo.

Como beneficios mencionar la facilidad de usarse, básicamente se crea el código secuencial que resuelva la tarea ,y luego aplicar las directivas OpenMP para paralelizarlo.

Esta diseñada para que sea ella la que controle como serán distribuido los cálculos, haciendo que el problema más importante de este tipo de paradigma, un procesador corrompa la memoria de otro, no aparezca: la librería se encarga de todo.

3.2.Memoria distribuida : OpenMPI

Los sistemas con memoria distribuida son aquellos en los que cada procesador tiene su propia memoria a la que solo accede él. El caso más común es querer aprovechar el cálculo uso de varios computadores , no necesariamente homogéneos, que están unidos mediante una red de computadores..

En este caso cada uno tiene su propia memoria, lo que por una parte beneficia a la hora de que ninguno pueda escribir en la zona del otro por un error del programador, lo cual era una posible pega de la programación con memoria compartida (con openMP no se deja al programador trabajar a un nivel lo suficientemente bajo como para enfrentarse a este error).

Aquí nos encontramos la pega de que el intercambio de datos es mucho mas lento, se hace mediante paso de mensajes ,por lo que el programador debe intentar minimizar los datos a compartirse entre las diferentes maquinas.(Una posible solución para algunos programas en los que los datos no se modifiquen seria guardar todos los datos en todos los ordenadores de igual manera)

OpenMPI es una librería para simplificar la programación en este tipo de situaciones.

3.3.GP-CPU (Cuda)

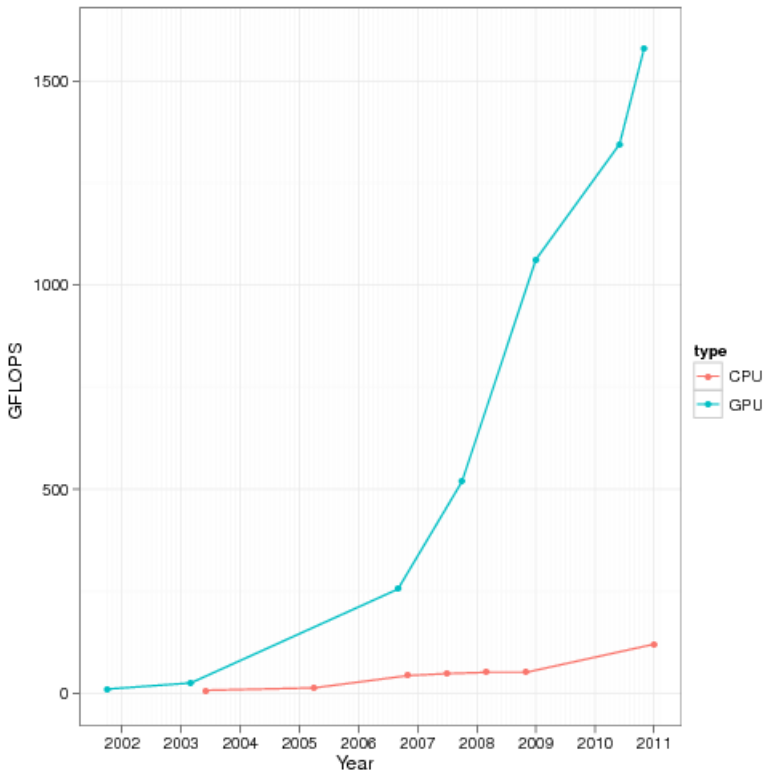
Cuda es un lenguaje de programación creado por NVIDIA para poder aprovechar la potencia de cálculo de sus tarjetas de video .

Esta dentro del paradigma GP-CPU siendo el más novedoso y extraño del resto.

Pese a este inconveniente nos encontramos con que es el que mayor rendimiento se obtiene (Por supuesto siempre que el programa sea paralelizable y lo suficientemente grande, si queremos hacer un sencillo calculo como sumar diez números no se notara diferencia, lógicamente)

Aquí el modelo de comunicación es maestro-exclavo : tu defines unas funciones ,en este caso se denominan kernels,y se los mandas a la tarjeta..

NOTA : Recientemente se está desarrollando una abstracción llamada OpenACC, muy similar a como funciona OpenMP , que simplifica la programación para GPU.Se está avanzando mucho por lo que lo añadiré más información en los anexos.



Diferencia potencia CPU y GPU

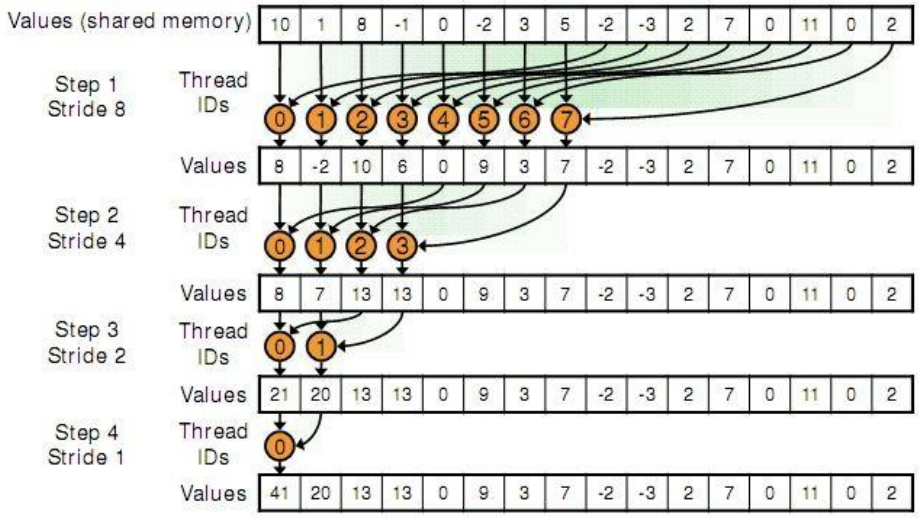
4. Aplicaciones prácticas para programas de estadística:

4.1.Ejemplos sencillos paralelizables

- Suma: (importante recordar que la resta es una suma).

Probablemente sea una de las operaciones más usadas en estadística :no solo en los estadísticos si no para el cálculo de distancias ..

La suma es una operación asociativa y conmutativa, por lo cual su propia naturaleza es paralela.
Pero, además, existen métodos muy útiles para agilizar la forma de calcularlas de forma paralela : el "reduce" (Mas abajo se verá un ejemplo de uso para el cálculo de checksum)



Cada hilo se encarga en esta ocasión de reducir el elemento situado en la posición que coincide con su identificador (`tid`), y el elemento después del último hilo en ejecución (`tid + s`). En los pasos posteriores se repite el mismo comportamiento con la diferencia que ahora el offset `s` es más pequeño debido a que hay menos hilos en ejecución.

Como he comentado anteriormente, la suma es sin duda la operación más usada, por tanto el mayor interés debe ser poder calcularla de la forma más eficiente .
Voy a utilizar el ejemplo de un estadístico muy común, la media, para explicar el interés en optimizar la suma en vez de este otro , el cual es el estadístico mas importante.

La media es la suma de los valores dividido entre el numero total de ellos.
El resto son funciones de las medias, como los momentos que son medias de las diferentes potencias. Podríamos pensar se puede descomponer como la media de medias

$$1/n \sum_{i=1}^{n} x_i = \frac{l/l \sum_{i=1}^{l} x_l + m/m \sum_{i=l+1}^{l+m} x_j}{l+m}$$

Al observar la formula nos encontramos con que aparecen divisiones , lo cual si pensamos en usar la media como nuestra función “primitiva” a la larga nos llevara a tener que deshacer y rehacerlas Además de la perdida de precisión por los redondeos .

Por tanto lo mas practico es evitar hacer operaciones innecesarias. El total de los valores n, no hace falta recalcularlo pues ya lo conocemos.

Imaginemos que tenemos k procesadores. A cada uno se le van a introducir unos datos y reservamos una zona de memoria (un vector) para sus resultados.
Dependiendo de nuestras necesidades podemos optar por dos soluciones diferentes, dependiendo de la cantidad de datos o de lo que busquemos e incluso poder usar ambas de manera conjunta
Pueden pedirme solo las medias entonces solo divido los datos en trozos dependiendo el numero de procesadores

Una de las técnicas seria enviar a la mitad de los procesadores una tanda de datos para los que nos calcule su suma, mientras podemos aprovechar para pedirle a uno de los otros procesadores (ya que la memoria de los datos esta compartida) que use los mismos datos para que nos devuelva las sumas pero de los cuadrados de estos datos y así sucesivamente.

Para el cálculo de las potencias hacemos uso que ya tenemos una copia del valor original “X” y el resultado queda en el vector de resultados que hemos asignado a dicho procesador.
Para pasar a la siguiente potencia usamos la potencia anterior y la multiplicamos por el valor original que siempre lo tendremos en el vector de resultados del procesador que lo calculo.
Y así sucesivamente

4.2. Un ejemplo más complejo: Trabajo con Matrices de diferente tamaño

Todos los cálculos se han hecho con una misma máquina, Acer Aspire con 4 procesadores y 4G RAM

Para mostrar el verdadero incremento al utilizar estas técnicas, voy a usar un ejemplo con el que se trabajan con grandes matrices: el método Stencil. (Anexo casi todo pdf explicación secuencial)

Primero debe buscarse que partes se pueden (y nos interesa) paralelizar

En el código a paralelizar se han encontrado cuatro puntos paralelizables, que son los siguientes:

- El bucle encargado de copiar la matriz de escritura a la de lectura en cada vuelta del bucle principal
- El bucle para crear la frontera periódica de la matriz
-
- El bucle principal encargado de calcular la nueva matriz
-
- El bucle encargado de calcular la suma de comprobación

Aquí mostrare este último escrito para los diferentes modelos. El resto está en el anexo

El bucle encargado de calcular la suma de comprobación

- Secuencial

```
sum = check_sum(matrix1,rows,cols);
      t_end = cp_Wtime();
```

(El código restante es el del siguiente apartado sin los pragmas)

- OpenMP

Aqui utilizo una directiva #pragma omp for y una variable privada temporal tmpSum que contiene la suma parcial para cada hilo, sumas parciales que luego son añadidas a la total, sum, dentro de una sección crítica.

```
m_type sum = 0;
m_type tmpSum = 0;
int i,j;
#pragma omp parallel default(none)\
shared(rows,cols,sum,matrix)\
private(tmpSum,j)
{
tmpSum = 0;
#pragma omp for
for(i=1; i<rows+1; i++)
for(j=1; j<cols+1; j++)
tmpSum += m(i,j);
#pragma omp critical
sum+=tmpSum;
}
```

- OpenMPI
-
- `m_type sum_parcial = check_sum(matrix2,sub_rows+2,cols);`
- `MPI_Allreduce(&sum_parcial,&sum,1,MPI_FLOAT,MPI_SUM,MPI_COMM_WORLD);`

- Cuda

Este es el kernel encargado de calcular el checksum, usando el método de reducción de suma que comente anteriormente.

Recibe como parámetros la matriz de lectura y la matriz de escritura.

```
//! Kernel que realiza la reduccion de un array de entrada y lo
deja en un array de salida
//! @param g_idata      Array de los valores de entrada
//! @param g_odata        Array de los valores de salida (valores
reducidos)
////////////////////////////////////////////////////////////////
////////////////////////
__global__ void gpuFunc_Reduce(m_type* g_idata, m_type* g_odata){
    extern __shared__ m_type sdata[];
    // cada hilo carga un elemento desde memoria global hacia
memoria shared
    unsigned int tid = threadIdx.x;
    unsigned int igl = blockIdx.x * blockDim.x + tid;
    sdata[tid] = g_idata[igl];
    __syncthreads();

    //Hacemos la reduccion en memoria shared
    for (unsigned int s=blockDim.x/2; s>0; s/=2) {
        if (tid < s) {
       // Hacemos la reducciÃ³n sumando los dos elementos que le
tocan a este hilo
           sdata[tid] += sdata[tid+s];
        }
      __syncthreads();
    }

    // El hilo 0 de cada bloque escribe el resultado final de la
reduccion
    // en la memoria global del dispositivo pasada por parametro
(g_odata[])
    if (tid == 0)
        g_odata[blockIdx.x] = sdata[0];
```

NOTAS IMPORTANTES:

CUDA es un lenguaje basado en c para poder aprovechar la potencia de las GPU de la marca NVIDIA. Se ha convertido un estándar, pero actualmente existe otro lenguaje, OpenCL que pretende ser compatible con cualquier GPU (e incluso otros dispositivos para acelerar cálculos).

El objetivo de este trabajo es facilitar el que cualquier persona pueda usar esta potencia computacional extra sin apenas tener que cambiar el código de programación; he desarrollado mas el paradigma “Memoria Compartida” con OpenMP ,en los ejemplos, ya que su uso es muy sencillo : solo hace falta añadir unas directivas pragma en la zona que se quiera paralelizar (principalmente los bucles) y el resto lo hace automáticamente.

Se está desarrollando algo similar a esta metodología (la más sencilla) ,para su uso en GPU CUDA bajo el nombre es OpenACC.

No he entrado en ella ya que al tiempo de escribir esto aun estaba poco desarrollada, pero tras estos meses ya esta lo suficiente maduro para su uso.

Desgraciadamente ya no puedo incluirlo aquí, pero entendiendo OpenMP no debería resultar complicado. Por ello he decidido simplificar las explicaciones de CUDA: puedes usar el código como ejemplo pero recomiendo ver OpenACC.

En los anexos hay mucha más información.

4.3.Optimizando GNU R :

Este apartado está destinado a aplicar los conceptos anteriormente en el programa estadístico GNU R; Es un programa muy potente pero tiene un importante problema: está programado para que funcione de forma secuencial, por lo tanto no aprovecha todos los recursos disponibles.

Mencionar que toda mejora también depende como se escriba el código: antes de cambiar nada en el programa veamos dos ejemplos que hacen lo mismo, uno escrito de forma secuencial y otro vectorizado (Recordemos que GNU R es un lenguaje vectorial)

```
 # Loop
  for (i in 1:length(a)) {
    a[i] <- a[i] + 10
}
))

#  user  system elapsed
# 0.191   0.008   0.216
```

Vectorizado:

```
print(system.time(
  # Vector
  a <- a + 10
))
#  user  system elapsed
#     0       0       0
```

Existen multitud de opciones solventar esto, aquí destacare las que actualmente funcionan mejor:

4.3.1. Usando paquetes externos

Existen multitud de paquetes para cargar en R que nos permiten que nuestro programa se ejecute de manera paralela.
Continuamente van apareciendo nuevas y otras quedan obsoletas, por tanto solo mencionare las que actualmente son más utilizadas.

- Para el uso de los diferentes núcleos de la CPU :

Paralell : contiene una versión de lapply optimizada para distribuirse entre los diferentes núcleos : mcapply.
Solo tienes que cambiar los apply por mcapply

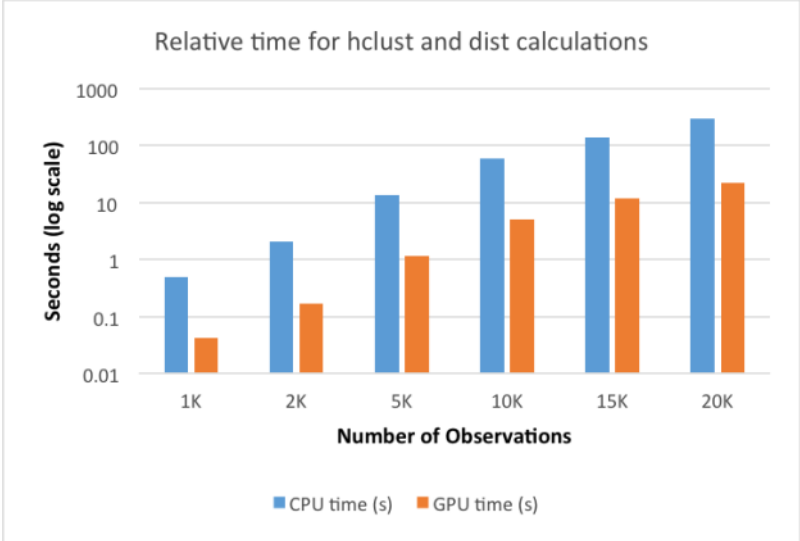
```
inputs <- 1:10000
processs <- function(x) {
  x + 2
}

# Detect number of cores on the host
numCores <- detectCores()
# Changing this to lapply would make it use a single process
result <- mclapply(inputs, processs, mc.cores = numCores)
```

En caso que tengas tu código escrito con bucles (pese a que es un error escribir código así en GNU R, hay ocasiones que es inevitable).Para ello existe el paquete “foreach”

```
library(doParallel)
library(parallel)

numCores <- detectCores()
cl <- makeCluster(numCores)
registerDoParallel(cl)

inputs <- 1:10000
process <- function(x) {
  x + 2
}

results <- foreach(i=inputs) %dopar% {
  process(i)
}
```

La parte importante aquí es %dopar%, indicándole que lo ejecute en paralelo. Si usáramos %do% lo ejecutaría como un simple proceso.

Para mandar que se ejecute el trabajo a la GPU existe el paquete “rpud”, que implementa unos cuantos algoritmos en R que se ejecutaran en una GPU compatible con CUDA. Entre estos algoritmos mencionar suport vector machine, clasificación bayesiana, y modelos lineales .



Observa que el eje Y esta en escala log-10 .Sacado blog David Smith [4]

4.3.2. Compilar R con otras librerías algebra lineal

GNU R viene por defecto compilado con la librería de algebra lineal BLAS.la versión que trae por defecto no viene optimizada y no soporta el uso de varios nucleos.
Escoger otra versión de esta librería puede dar un incremento significante al rendimiento.
Existen varias alternativas:

- OpenBlas

Atlas (esta versión tiene el mismo problema que la versión nativa : no viene optimizada y si deseas usar la versión optimizada debes compilarla por ti mismo)

- MKL :

Una versión optimizada por Intel .Revolution R (La versión de pago de GNU R)viene con ella incluida

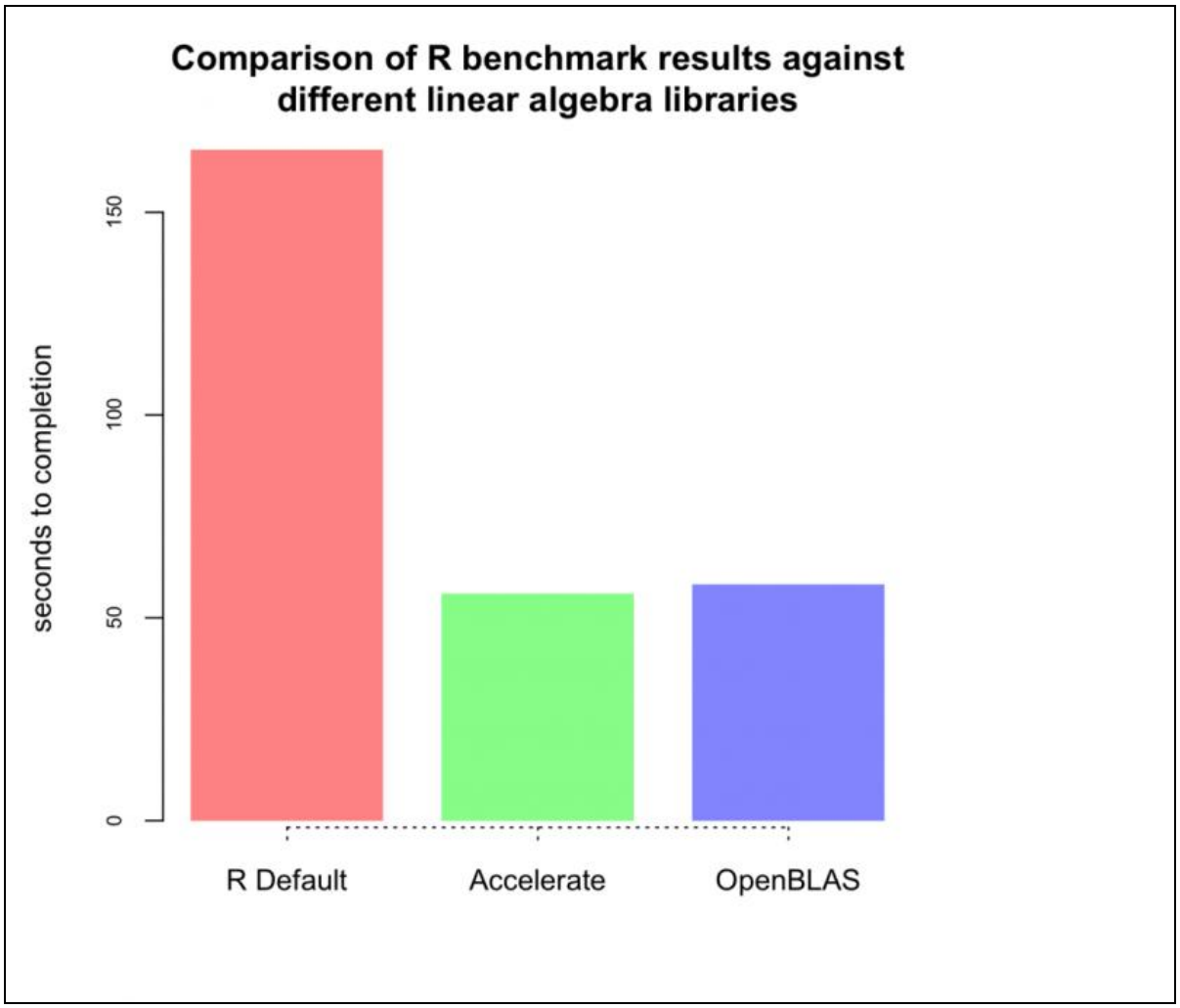
- CuBlas : es una versión optimizada para usarse en GPU CUDA

4.3.3. Enlazar R con las librerías de algebra lineal

En el apartado anterior mencionaba como el que GNU R esté compilado con una u otra librería de algebra lineal hace que haya un incremento en la productividad y aprovechamiento sin necesidad de hacer grandes cambios.
Inicialmente había incluido los pasos para compilarlo (En GNU Linux) con cada uno de los paquetes y un pequeño ejemplo de su mejora.
A medida que avanzaba en la investigación y las consecuentes pruebas me encontré con que lo anterior era un trabajo totalmente innecesario: en vez de tener que cambiar cosas y recompilar para que soporte una de las diferentes librerías bastaba con cambiar el enlace simbólico que los une. Esto se hace en una sola línea de terminal y se obtiene la misma mejoría.
Hay muchos tutoriales que van paso a paso: consiste en encontrar la librería que está usando y la que quieres usar
ej
/usr/lib/libopenblas.so
Y
/usr/lib/libopenblas.so

Hacer una copia para no perderla
>cp /usr/lib/blas.so /usr/lib/blas-OLD.so

>ln –s /usr/lib/libopenblas.so /usr/lib/blas.so


Comparison of R benchmark results against different linear algebra libraries
seconds to completion
0
50
100
150
R Default
Accelerate
OpenBLAS

5. Bibliografía

[1]	"How Multimedia Workloads Will Change Processor Design: Diefendorff y Pradeep K. Dubey.
[2]	Openacc 2.0++ and OpenMP 4.0++ Directive based programing on “accelerators” today and into the future .James Beyer, Ph.D
[3]	The OpenACC™ Application Programming Interface
[4]	R-bloggers – Computing with GPU in R .David Smith
[5]	cuBLAS : The cuBLAS library is freely available as part of the CUDA Toolkit. https://developer.nvidia.com
[6]	HiPLAR (High Performance Linear Algebra in R) delivers high performance linear algebra (LA) routines for the R platform for statistical computing using the latest software libraries for heterogeneous architectures
[7]	Sistemas multiprocesadores – Gomez Pedraz
[8]	Block Cyclic Distribution of Data in pbdR and its Effects on Computational Efficiency. Matthew G. Bachmann
[9]	Introduction to Distributed Computing with pdbR and the UMBC.Andrew m, Raim
[10]	Distributed Data Analysis with Hadoop and R.Jonathan Seidman
[11]	Apuntes asignatura computacion paralela : Diego Llanos,Arturo Escribano y practica Stencil
[12]	R-statistics blog http://www.r-statistics.com/
[13]	http://arstechnica.com/features/2000/04/ps2vspc/5/
[14]	http://brainarray.mbni.med.umich.edu/brainarray/rgpgpu/
[15]	http://www.r-bloggers.com/for-faster-r-use-openblas-instead-better-than-atlas-trivial-to-switch-to-on-ubuntu/
[16]	http://www.onthelambda.com/2013/11/22/compiling-r-from-source-and-why-you-shouldnt-do-it/

6. Anexo

Diferentes paquetes de R para diferentes problemas

- Gmatrix
- Rbdr
- Rcpp
- Rth
- Snow
- Gridr
- Multicore

Nuevas formas poder optimizar un programa para ejecutarse en una GPU: pragmas .OpenACC

Compilar las partes más pesadas de código R
JIT de R : Speed up your R code using a just-in-time (JIT) compiler R-statistics blog

Programing with big data in R

Parallel R fle load

Distributed Data Analysis with Hadoop and R Presentation

Integrating R and Hadoop

Benchmarking Single- and Multicore BLAS Implementations and GPUs For use with R.Dirk Eddelbuettel.

Tutorial – Distributed Data Analysis using R.GridR.Institut Intelligence Analyse- und Informationssysteme Fraunhofer

NOTA

Durante el transcurso de este trabajo , el cual ha sido desarrollado y probado en sistemas GNU Linux, he intentado aplicar las mismas técnicas mencionadas en GNU R para MSWindows pero sin éxito. De casualidad he encontrado un articulo especifico que añado en los anexos :

Parallel Multicore Processing with R (on Windows) _ R-statistics blog

http://www.r-bloggers.com/an-openblas-based-rblas-for-windows-64/