Implementación en GPU del Estadístico *t* para análisis de expresión genética en microarreglos

GPU implementation of *t*-Student algorithm for analysis of genetic expression with microarrays

Eduardo Romero-Vivas*, Fernando Von Borstel Luna*, Isaac Villa-Medina*,**

RESUMEN

Introducción: Los microarreglos de ADN son utilizados para analizar simultáneamente el nivel de expresión de genes bajo múltiples condiciones; sin embargo, la masiva cantidad de datos generados hacen que su análisis sea un candidato ideal para su procesamiento en paralelo. La utilización de Unidades de Procesamiento Gráfico de Propósito General (GPGPU) es una alternativa eficiente y de bajo costo, comparada contra aplicaciones que utilizan CPUs. Objetivo: Implementación de algoritmos basados en la Arquitectura de Dispositivos de Cómputo Unificado (CUDA) para determinar la significancia estadística en la evaluación de los niveles de expresión de genes en microarreglos. Método: Análisis paramétrico *t*-pareado desarrollado en CUDA. Resultados: La evaluación utilizando la implementación en CUDA es 5 a 30 veces más rápida que la implementación en CPU, dependiendo del número de genes a ser evaluados. Conclusiones: Los resultados son comparados contra las implementaciones tradicionales en CPU; se proponen mejoras.

ABSTRACT

Introduction: DNA microarrays are used to analyze simultaneously the expression level of thousands of genes under multiple conditions; however, massive amount of data is generated making its analysis a challenge and an ideal candidate for massive parallel processing. Among the available technologies, the use of General Purpose Computation on Graphics Processing Units (GPGPU) is an efficient cost-effective alternative, compared to a Central Processing Unit (CPU). Objective: This paper presents the implementation of algorithms using Compute Unified Device Architecture (CUDA) to determine statistical significance in the evaluation of gene expression levels for microarray hybridization. Method: A *t*-paired parametric analysis using a GPU implementation developed in CUDA. Results: The gene expression evaluation using the GPU implementation is 5 to 30 times faster than a CPU implementation, depending on the number of genes to be evaluated. Conclusions: The results with respect to traditional implementations are compared, and further improvements are discussed.

INTRODUCCIÓN

Los avances tecnológicos recientes en el área de biología molecular y genómica han desencadenado un crecimiento acelerado en la cantidad de información generada. Ejemplos prominentes de este crecimiento en la cantidad de información se pueden vislumbrar en las bases de datos públicas de secuencias de ADN, tales como GenBank o UniProt, en las que la información se duplica aproximadamente cada 6 meses. Las tecnologías de secuenciación de nueva generación o el uso de microarreglos para el análisis de expresión génica permiten el análisis a gran escala, cubriendo una amplia proporción del genoma de un organismo (a diferencia de las técnicas que hace pocos años solo permitían analizar genes por separado). Los microarreglos de ADN, por ejemplo, permiten analizar simultáneamente el nivel de expresión de miles de genes ante condiciones múltiples; su uso ha revolu-

Recibido: 3 de mayo de 2012
Aceptado: 15 de junio de 2012

Palabras clave:
GPU; microarreglo; CUDA.

Keywords:
GPU; microarray; CUDA.

*Centro de Investigaciones Biológicas del Noroeste, S. C. Instituto Politécnico Nacional 195. Col. Playa Palo de Santa Rita Sur, C. P. 23096, La Paz, B. C. S., México. Tel. (612) 123 8484, ext. 3358. Correo electrónico: fborstel04@cibnor.mx.
**Instituto Tecnológico de La Paz. Boulevard Forjadores de Baja California Sur n. 4720, Apartado Postal 43-B, C. P. 23080, La Paz, B. C. S., México.

cionado la biología molecular, impactando en áreas como la académica, la médica y la farmacéutica, la biotecnológica, la agroquímica y la industria alimenticia. Hoy en día, los costos de análisis de esta información, económicos en tiempo y en recursos, tienden a superar los costos de su generación [1]. Dicho crecimiento en la cantidad de información generada en cada experimento demanda el uso de nuevas tecnologías de análisis que vayan a la par con la dimensión de los datos. La Bioinformática, entendida ésta como la aplicación de matemáticas, estadística y tecnologías de la información para el análisis de señales genómicas y proteómicas, es la respuesta a este reto.

Una de las principales características de los microarreglos es el gran volumen de datos que se generan, por tanto, uno de los grandes retos en este campo involucra el manejo e interpretación de éstos. La dimensión de la información generada y su análisis hace de los microarreglos candidatos ideales para el procesamiento paralelo, aprovechando las arquitecturas de muchos núcleos y multi-núcleos que están revolucionando el cómputo de alto rendimiento. No obstante, el uso de clústers y supercomputadoras sigue siendo exclusivo de laboratorios y universidades con grandes recursos. Mientras tanto, el desarrollo de arquitecturas con muchos núcleos, tales como las Unidades de Procesamiento Gráfico (GPU por sus siglas en inglés) y, en específico, la Arquitectura de Dispositivos de Computo Unificado (CUDA por sus siglas en Inglés) propuesta por NVIDIA en el 2006 [2-4], permiten el desarrollo de algoritmos de análisis bioinformático de alto rendimiento en dispositivos de bajo costo y alto poder de cómputo.

Los trabajos para el análisis de microarreglos utilizando GPUs son escasos. Por ejemplo, un algoritmo basado en GPUs para realizar la clasificación de los genes expresados en un microarreglo ha sido desarrollado recientemente en [5]. En este trabajo se presenta la implementación de algoritmos en CUDA para determinar la significancia estadística en la evaluación de niveles de expresión génica para un experimento de hibridación de microarreglos diseñado en el Centro de Investigaciones Biológicas del Noroeste, S. C. (CIBNOR). De manera similar, se comparan los resultados respecto a implementaciones tradicionales.

MATERIALES Y MÉTODOS

Microarreglos. Los microarreglos de ADN son dispositivos capaces de medir los niveles de expresión de miles de genes de forma paralela. Un microarreglo consiste en una superficie cristalina sólida, generalmente una laminilla de microscopio, a la cual se adhieren moléculas específicas de ADN con el propósito de detectar la presencia y abundancia de moléculas complementarias (ácidos nucleicos) marcadas en una muestra biológica (hibridación vía formación dúplex Watson-Crick). En la mayoría de los experimentos de microarreglos, los ácidos nucleicos marcados derivan del ARN mensajero (mARN) del tejido muestra del organismo, el cual está involucrado en el proceso de generación (codificación) de una proteína, por lo cual, el microarreglo mide la expresión de un gen cuantificando de forma relativa la abundancia de moléculas adheridas [6-8].

La figura 1 muestra el diseño experimental más común en el uso de un microarreglo. Partiendo de dos tejidos con condiciones biológicas distintas, como por ejemplo una condición anormal y una condición normal de control, se procede a extraer material genético de cada muestra. Las muestras se marcan con fluoróforos distintos, Cy5 en rojo para el primer tejido y Cy3 para el tejido de control, y se procede a hibridar ambas laminillas. El uso de estos marcadores permitirá detectar el material genético complementario de la muestra que se ha adherido al microarreglo mediante la emisión de luz, puesto que éstos son iluminados por un láser en rojo y verde respectivamente. Ambas imágenes se combinan para obtener una imagen a color donde los genes sobreexpresados adquieren tonalidades de rojo, los genes inhibidos tonalidades en verde y los genes que han permanecido en la misma condición en ambas muestras se presentan en amarillo. A continuación se realiza una estimación de la intensidad de la señal en cada caso, realizando correcciones contra el fondo oscuro y la normalización de las señales [6]. La sobreexpresión o subexpresión de un determinado gen se puede representar como una fracción definida en la ecuación 1. Dado que genes sobrerregulados por un factor de 2 darían una relación de 2, y que genes subregulados darían un valor de 0,5, es preferible usar una transformación logarítmica con base 2, de tal forma que un gen sobreexpresado al doble dará un valor de 1, en tanto que un gen que sea subexpresado dará un valor de -1 (haciendo intuitiva su interpretación y reflejando la simetría natural del fenómeno biológico [7,8]).

$$ratio = \frac{IB - BkgB}{IA - BkgA}. \quad (1)$$

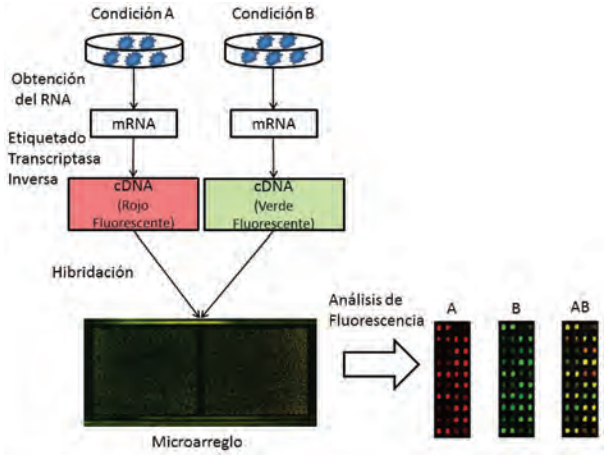

Figura 1. Diseño experimental y uso de los microarreglos.

Análisis estadístico de expresión diferencial

Para cada gen en el diseño presentado se tiene una medida de expresión que compara ambas muestras para un experimento dado. Sin embargo, con la finalidad de representar la variabilidad existente entre una población de organismos, se requiere contar con repeticiones del experimento para distintos individuos para poder identificar los genes que son diferencialmente expresados de forma consistente. Fijar un umbral de expresión y promediar las lecturas para el total del número de organismos no sería apropiado dado que no reflejaría el grado en que los niveles de expresión varían para cada individuo, ni tomaría en cuenta el tamaño de la muestra (es decir, el número de organismos involucrados en el estudio). Por ello se determinará si un gen está diferencialmente expresado mediante una prueba de hipótesis. La hipótesis nula para este experimento es que no hay diferencia en expresión para ambos tejidos. Si esta hipótesis fuese cierta, la variabilidad en los datos solo representaría la variabilidad entre individuos, o bien un error en las mediciones. La selección de genes diferencialmente expresados no se hará, por tanto, con base a su proporción definida en la ecuación 1 sino con base a un valor p predefinido (p = 0,001), es decir, a la probabilidad de observar un cierto nivel de cambio aleatoriamente [9].

Para el propósito de este estudio se ha seleccionado la prueba t pareada calculada como

$$t = \frac{\overline{X}}{S/\sqrt{n}}. \qquad (2)$$

Donde $\overline{X}$ es el promedio del logaritmo de las proporciones definidas en la ecuación 1. S es la desviación estándar calculada con la ecuación 3 y n es el número de réplicas biológicas del experimento.

$$S = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{n-1}}. \qquad (3)$$

El valor de p se calcula a partir de la estadística t por comparación con una distribución t con un número apropiado de grados de libertad (en este, caso el número de réplicas menos uno).

Diseño del microarreglo

Como parte del proyecto SAGARPA-CONACYT 2009-II intitulado "Aplicación de la genómica funcional como estrategia para la mejora continua de la industria del camarón", se diseñó un microarreglo especifico para camarón a partir de secuencias únicas provenientes de bases de datos públicas (GenBank) y librerías sustractivas generadas en el CIBNOR. La selección de secuencias, pre-procesamiento, ensamblaje y diseño de sondas se llevó a cabo en el CIBNOR, en tanto que la impresión física del microarreglo se encargó a la compañía Biodiscovery, LLC (dba MYcroarray). Los experimentos reto ante diversas condiciones biológicas se efectuaron en el CIBNOR, mientras que el proceso de hibridación y escaneo del microarreglo se realizó en la Unidad de Microarreglos de DNA en el Instituto de Fisiología Celular de la UNAM.

La figura 2 muestra un ejemplo de la imagen del microarreglo generada para un experimento dado y un acercamiento a la imagen. La imagen del microarreglo mostrada es el resultado de combinar las imágenes de la laminilla de control y la laminilla de la condición reto, y contiene 61 440 genes ordenados en 160 filas y 384 columnas divididas en dos bloques. Cada punto representa una secuencia de 70 bases, representativa y única para el gen de interés. En el acercamiento a una sección del microarreglo de nueve por nueve genes se muestra que a cada gen del microarreglo corresponde un conjunto de puntos de la imagen donde se forma un círculo, en el cual no todos los puntos tienen la misma intensidad.

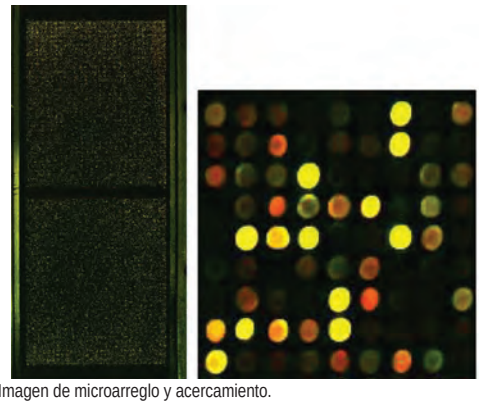

Figura 2. Imagen de microarreglo y acercamiento.

Para determinar el nivel de expresión de cada gen a partir del conjunto de puntos que forman la imagen se utilizó el programa SpotFinder de análisis de imágenes de microarreglos, el cual genera una tabla con los valores máximos, mínimos y promedio para la intensidad y para el fondo. La figura 3 muestra la salida del archivo correspondiente en formato *.mev. Un archivo similar se genera para cada par de laminillas de cada réplica.

```
#Version: V1.0
#Date: mar 25. oct 13:25:26 2011
#Input_row_count: 13056; bad/undetected spots:7761
#Created by: TIGR Spotfinder 3.2.1, Windows
#TIFF files processed: ch A- C:/Cy3.TIF; ch B- C:/Cy5.TIF;
# Segmentation method: Otsu, Min size: 3, Max size: 100; Top bkg cutoff: 5%
#Post-processing settings: QCfilter: ON, Background correction: ON, Keep flagged values: OFF, QC threshold= 1*medianBKG+1*StdDevBkg
#<QRTYPE> A=QUERY=Cy3=; B=REFERENCE=Cy5=
# Type your  comments here.
UID  RA  RB  R  C  MR  MC  SR  SC  FlagA  FlagB  SA  SF  QC  QCA QCB BkgA    BkgB    SDA SDB SDBkgA SDBkgB MedA    MedB    MRA MRB MedBkgA MedBkgB X   Y   PValueA PValueB
1    0   0   1  1   1   1   1   1   X     X     79  1.000   0.00000 0.00000 0.00000 14892   10081   93.4  92.7   58.9   59.9    0   0   0   0   204 137 112 89  0.0000  0.0718
2    0   0   1  2   1   1   1   2   X     X     91  1.000   0.00000 0.00000 0.00000 18382   10529   88.2  91.2   55.6   51.4    0   0   0   0   202 119 129 89  0.0000  0.0013
3    0   0   1  3   1   1   1   3   X     X     78  1.000   0.00000 0.00000 0.00000 18018   11700   141.6 169.4  76.7   84.4    0   0   0   0   231 150 147 89  0.0000  0.0000
4    0   0   1  4   1   1   1   4   X     X     110 1.000   0.00000 0.00000 0.00000 21670   16170   153.0 170.2  73.0   79.7    0   0   0   0   197 147 164 89  0.0000  0.0000
5    0   0   1  5   1   1   1   5   X     X     76  1.000   0.00000 0.00000 0.00000 13580   9956    109.2 105.7  64.0   58.6    0   0   0   0   205 131 182 89  0.0000  0.0000
6    0   0   1  6   1   1   1   6   X     X     90  1.000   0.00000 0.00000 0.00000 17820   11790   95.7  115.5  58.2   57.6    0   0   0   0   198 131 199 89  0.0000  0.0001
7    0   48962   1  7   1   1   1   7   X     C     82  1.000   0.28267 0.18695 0.37879 19106   11480   161.8 238.8  85.8   100.0   0   0   817 0   361 233 140 217 89  0.0000  0.0000
8    0   47125   1  8   1   1   1   8   X     C     97  1.000   0.16727 0.09280 0.24174 21340   11543   121.6 162.7  79.7   85.9    0   0   478 0   486 220 115 234 89  0.0000  0.0000
9    0   0   1  9   1   1   1   9   X     X     70  1.000   0.00000 0.00000 0.00000 14560   9660    114.8 117.6  61.3   57.9    0   0   0   0   203 138 252 89  0.0000  0.0000
10   0   0   1  10  1   1   1   10  X     X     83  1.000   0.00000 0.00000 0.00000 16849   10292   133.0 132.0  58.5   63.7    0   0   0   0   203 124 269 89  0.0000  0.0000
11   240641  184403  1  11  1   1   1   11  C     C     62  1.000   0.35303 0.35303 0.35303 17236   9548    1619.7 1083.7 439.6  430.4   3367    2797    3881    2974    276 154 287 89  0.0000  0.0000
12   270472  203753  1  12  1   1   1   12  C     C     81  1.000   0.46116 0.46116 0.46116 20898   11502   1209.1 819.8  490.7  499.3   2967    2469    3339    2515    258 142 304 89  0.0000  0.0000
13   40204   167497  1  13  1   1   1   13  C     C     86  1.000   0.30587 0.26962 0.35012 20382   12040   978.2  1816.6 124.7  258.3   312 1705    468 1848    237 140 322 89  0.0001  0.0000
14   94821   159180  1  14  1   1   1   14  C     C     94  1.000   0.43159 0.39829 0.46488 22372   13066   251.2  559.9  122.0  223.4   368 1547    370 1693    238 139 339 89  0.0000  0.0000
15   707662  623684  1  15  1   1   1   15  C     C     314 1.000   0.66152 0.66152 0.66152 28044   14934   1569.2 1184.8 450.1  408.5   6137    5460    6208    5471    296 131 357 89  0.0000  0.0000
16   860559  690706  1  16  1   1   1   16  C     C     103 1.000   0.50742 0.50742 0.50742 24926   15965   2517.4 2340.9 1152.5 1237.7  8647    8664    9326    8648    242 155 374 89  0.0000  0.0000
17   0   0   1  17  1   1   1   17  X     X     87  1.000   0.00000 0.00000 0.00000 16488   7239    116.6  87.4   58.2   56.3    0   0   0   0   184 127 392 89  0.0000  0.0034
```

Figura 3. Archivo de salida con los niveles de intensidad para cada gen.

Diseño experimental

Con la finalidad de evaluar el uso de los algoritmos de procesamiento paralelo para el análisis de expresión genética en microarreglos, se implementó el análisis paramétrico *t* pareado en las tarjetas de procesamiento gráfico mediante rutinas desarrolladas en CUDA. A partir de los datos procedentes de la hibridación de un microarreglo de 61 440 genes, se generaron varios sets de datos para evaluación variando el número de genes utilizados para el análisis y el número de réplicas del experimento.

El equipo de cómputo en el que se desarrolló el proyecto tiene las siguientes características: procesador Intel Core2Duo E8400 a 3,00 GHz con 2,0 GB de memoria RAM, un disco duro de 100 GB, sistema operativo Fedora 12 con una tarjeta gráfica GeForce 9800 GT (112 CUDA núcleos, capacidad de cómputo 1,1 CUDA con 1 024 MB de memoria dedicada e interfaz de memoria de 256-bits).

Implementación en CUDA

La figura 4 muestra un diagrama de flujo de las operaciones y funciones en CUDA para efectuar la prueba *t*, tal como se define en la ecuación 2.

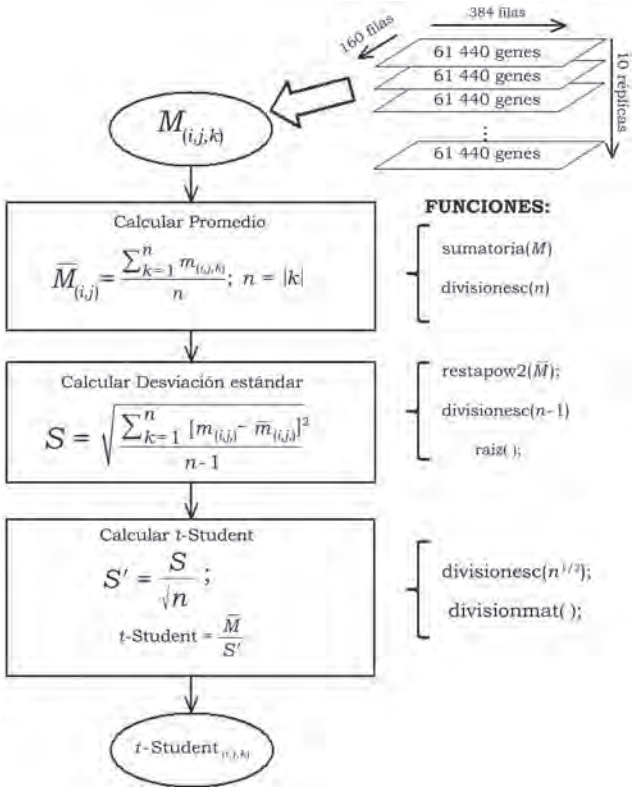

Figura 4. Diagrama de flujo del cálculo de *t* en CUDA.

La matriz tridimensional *M* conformada por los datos de los microarreglos se mapea en memoria global en un arreglo bidimensional, justo como se ilustra en la figura 5.

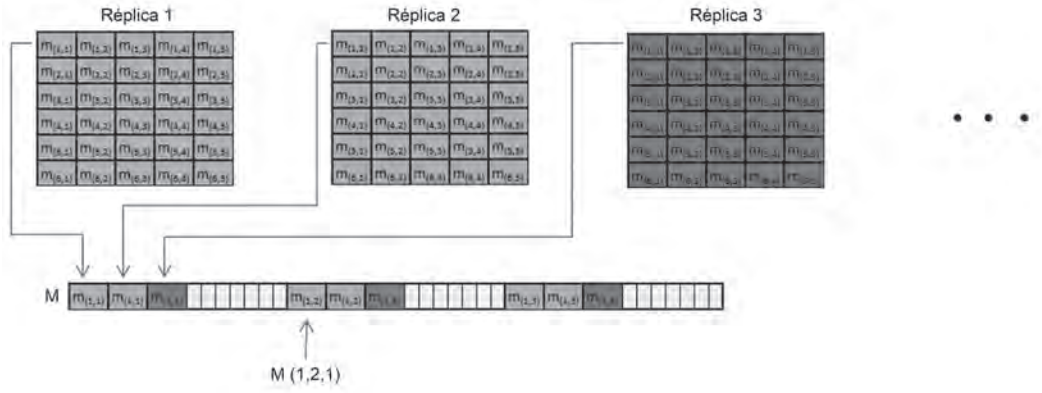

Figura 5. Mapeo de la matriz de datos a memoria global.

A continuación se presentan las funciones requeridas para el cálculo.

Paso 1. Obtener el promedio de los datos. Para realizar esta operación se requirió aplicar un algoritmo que permitiera sumar los elementos de un renglón, para posteriormente dividirlo entre el número de columnas, obteniendo así el promedio. Las funciones requeridas son:

- sumatoria(). Sumatoria por renglón. Algoritmo que recibe como parámetros una matriz con los datos a procesar y un vector. El algoritmo calcula la suma de todas las columnas de cada renglón y almacena el resultado en el vector que se le pasa como parámetro. Cada hilo de cada bloque se encarga de cargar un elemento de la matriz en el búfer compartido de cada bloque y las operaciones se realizan empleando dicho búfer.

```
sumatoria(Matriz,resultado,n_columnas)
{
    tx = threadIdx.x; ty = threadIdx.y;
    bx = blockIdx.x; by = blockIdx.y;
    idxb = by*gridDim.x + bx;
    size = hx*hy;
    idx = ty*n+tx;
    row = ((idxb*size) + (idx));
    buffer[ty*n+tx] = vector[idxb*size + idx];
    __syncthreads();
    for(i=0 to n_columnas do)
    {
        resultado[row] += buffer[ty*n+i];
    }
}
```

- divisionesc(). División por un escalar. Algoritmo que divide un vector entre un valor escalar; ambos se reciben como parámetros. Cada hilo de cada bloque carga un elemento del vector en el buffer para posteriormente realizar la operación.

```
divisionesc(vector,resultado,valor_n)
{
    tx = threadIdx.x;
    idx = tx + blockIdx.x*blockDim.x;
    buffer[tx] = vector[idx];
    __syncthreads();
    resultado[idx] = buffer[tx]/valor_n;
}
```

- promedio(). Promedio. Algoritmo que se auxilia de los algoritmos anteriores para obtener el promedio de cada renglón.

```
promedio(matris,resultado,n_columnas,n_renglones)
{
    definir dimGrid y dimBlock de acuerdo a su arquitectura CUDA
    gpumatris = cudaMalloc(n_columnas*n_renglones);
    gpuresultado = cudaMalloc(n_renglones);
    cpuresultado = cpumalloc(n_renglones);
    cudaMemcpy(gpumatris,matris);
    sumatoria<<<dimGrid,dimBlock>>>(gpmatris,gpuresultado,n_columnas);
    cudaSyncThreads
    cudaMemcpy(cpuresultado,gpuresultado);
    cudaFree(gpuresultado); cudaFree(gpumatris);
// ----------------------------promedio
    gpuresultado = cudaMalloc(n_renglones);
    gpuvector = cudaMalloc(n_renglones);
    cudaMemcpy(gpuvector,cpuresultado);
    divisionesc<<<dimGrid2,dimBlock2>>>(gpuvector,gpuresultado,n_columnas);
    cudaMemcpy(resultado,gpuresultado);
    cudaFree(resddiv); cudaFree(vectorddiv);
}
```

Paso 2. Obtención de la desviación estándar. Se requiere obtener la sumatoria del cuadro de la diferencia entre las muestras y el promedio, además de calcular la raíz cuadrada. Para ello se emplearon los siguientes algoritmos:

- restapow2(). Diferencia cuadrada. Algoritmo que recibe como parámetro una matriz y dos vectores, uno de datos y en el otro se almacenará el resultado. El algoritmo resta a los elementos de cada renglón el

valor que le corresponde en el vector. Los eleva al cuadrado y almacena el resultado en el vector correspondiente.

```
restapow2(matris,vector,resultado,n_columnas)
{
    tx = threadIdx.x; ty = threadIdx.y;
    bx = blockIdx.x; by = blockIdx.y;
    idxb = by*gridDim.x + bx;
    size = hx*hy;
    idx = ty*n+tx;
    row = idxb*size + idx;
    pos = idxb*hy + ty;
    buffer[idx] = matris[row];
    buffer2[idx] = vector[pos];
    __syncthreads();
    for(i=0 to n_columnas do)
    {
        tmpresta = buffer[ty*n+i] - buffer2[idx];
        resultado[row] += tmpresta*tmpresta;
    }
}
```

- raiz(). Raíz cuadrada. Algoritmo que recibe como parámetros dos vectores, un vector de datos y el vector donde se guardarán los resultados. El algoritmo obtiene la raíz cuadrada de cada elemento del vector de datos.

```
raiz(vector,resultado)
{
    tx = threadIdx.x;
    idx = tx + blockIdx.x*blockDim.x;
    buffer[tx] = vector[idx];
    __syncthreads();
    resultado[idx] = sqrt(buffer[tx]);
}
```

Paso 3. Calcular el valor de *t*. Para terminar el cálculo se desarrolló el siguiente algoritmo que se encarga de hacer una división entre 2 vectores elemento a elemento.

- divisionmat(). División de vectores. El algoritmo realiza la división elemento a elemento de 2 vectores y almacena el resultado en un tercer vector.

```
divisionmat(vector,vector2,resultado)
{
    tx = threadIdx.x;
    idx = tx + blockIdx.x*blockDim.x;
    buffer[tx] = vector[idx];
    buffer2[tx] = vector2[idx];
    __syncthreads();
    if(idx<n_renglones)
    {
        A = buffer[tx];
        B = buffer2[tx];
        resultado[idx] = A/B;
    }
}
```

RESULTADOS

Se comparó el resultado en tiempo de cómputo para la implementación en GPU contra el tiempo obtenido en una implementación en serie utilizando CPU, así como variando el número de genes involucrados en el análisis y el número de réplicas en cada experimento.

Las figuras 6 y 7 muestran el tiempo de procesamiento para la implementación en CPU y GPU respectivamente para diferente número de genes analizados y número de réplicas.

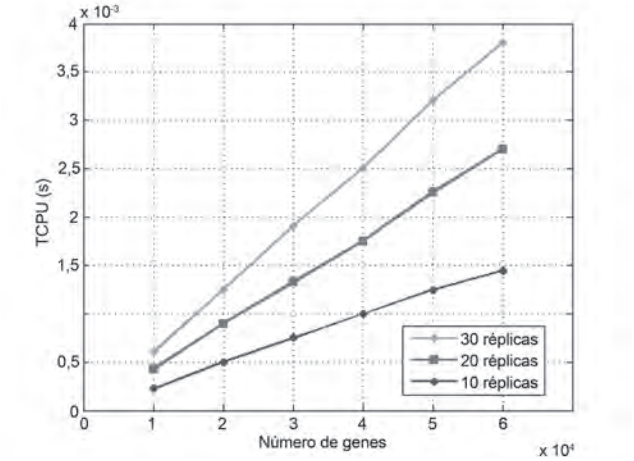

Figura 6. Tiempo de cómputo utilizando el CPU.

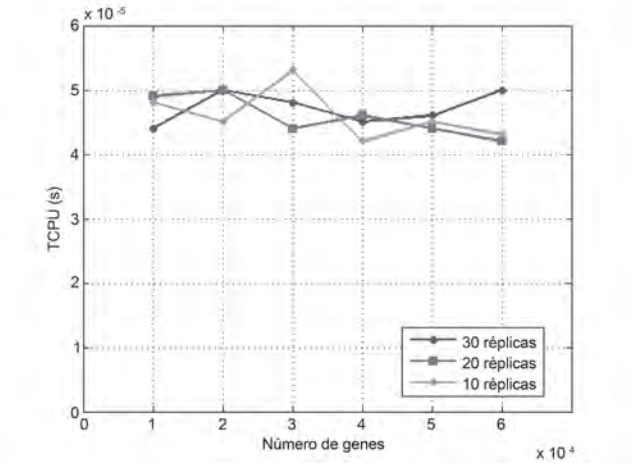

Figura 7. Tiempo de cómputo utilizando el GPU.

DISCUSIÓN

La figura 6 muestra cómo varía el tiempo de ejecución de la prueba *t* con respecto al número de genes involucrados en cada réplica y con respecto al número de

réplicas *n*. Para un número de réplicas dado, se puede observar un incremento lineal al aumentar el número de genes involucrados. Para un mayor número de réplicas, la pendiente se hace mayor al aumentar el número de valores a utilizar para cada cálculo. Este comportamiento corresponde a lo que cabría esperar de la implementación en serie desarrollada utilizando un CPU. Nótese que la velocidad de análisis depende no solo de la cantidad de genes a analizar, sino del arreglo de los datos respecto a las operaciones a realizar: el análisis de 30 réplicas para 20 000 genes es ligeramente más rápido que el análisis de 10 réplicas de 60 000 genes, aun cuando el total de genes involucrados es el mismo (600 000). Esto se debe a que en el primer caso realiza por ejemplo una suma con 30 valores, en tanto que el segundo realizara 3 sumas de 10 valores. La figura 7 muestra los tiempos correspondientes para el mismo cálculo pero ahora implementado en GPU. Se puede observar que los tiempos de proceso permanecen aproximadamente iguales -en el orden de las cienmilésimas de segundo-, independientemente del aumento en el número de genes o en el número de réplicas. Esto se debe a que, tal como se muestra en la figura 5, la GPU realiza una sola operación sobre múltiples datos. La diferencia entre analizar 30 réplicas para 20 000 genes y 10 réplicas para 60 000 genes es, por tanto, menor en GPU que en CPU. La figura 8 nos muestra la ventaja de utilizar el cómputo en paralelo sobre la implementación serial. El proceso se puede realizar de 5 a 30 veces más rápido dependiendo del número de genes involucrados en comparación con la implementación utilizando CPU.

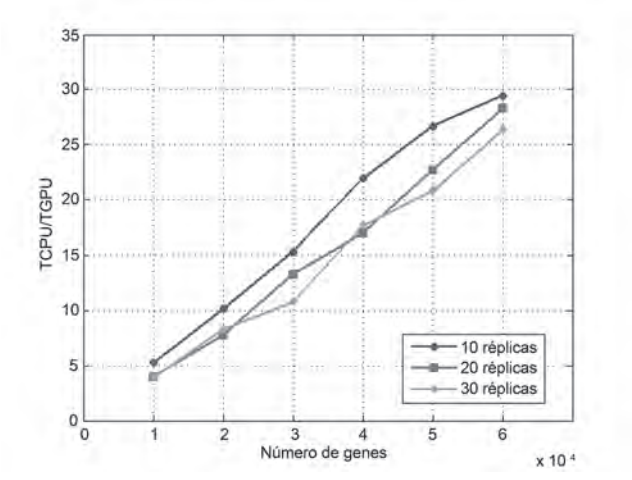

Figura 8. Proporción de tiempo de cómputo CPU/GPU.

CONCLUSIONES

A pesar de haberse realizado el cómputo en GPU de 5 a 30 veces más rápido, el orden de los tiempos empleados en CPU y GPU pareciera, a primera vista, no justificar el uso de una implementación en paralelo, ya que ambas se realizan en fracciones de segundo. Sin embargo, hay que tomar en cuenta que como aproximación inicial se ha implementado la estadística paramétrica más básica *t* en un estudio simple con datos pareados. La técnica de microarreglos se usa en experimentos más complejos, en donde puede haber múltiples grupos en los que más de una condición es analizada. Este tipo de experimentos requiere de análisis más sofisticados conocidos como las pruebas de ANOVA y los modelos lineales generalizados. Ambas técnicas son similares a la prueba *t* en cuanto a que requieren que la variabilidad en los datos sea normalmente distribuida. Sin embargo, esta condición se puede relajar para ambas técnicas utilizando análisis *bootstrap*, para los cuales se requiere generar -a partir de los datos que se tienen- nuevos sets de datos de la misma dimensión, siendo común generar millones de estos sets para generar las distribuciones correspondientes [8]. El paquete SAM [10], disponible como librería para Excel, por ejemplo, permite realizar este tipo de análisis en CPU, no obstante, tiende a ser lento para sets de datos amplios [8] y [10]. En estos casos, es plenamente justificable la ventaja en velocidad de análisis que presentan las implementaciones en GPU.

AGRADECIMIENTOS

Los autores agradecen el apoyo del proyecto SAGARPA-CONACYT 2009-II intitulado “Aplicación de la genómica funcional como estrategia para la mejora continua de la industria del camarón”.

REFERENCIAS

[1] Sboner, A., Mu, X. J., Greenbaum, D., Auerbach, R. K. and Gerstein, M. B. (2011). The real cost of sequencing: higher than you think! *Genome Biology* 12: pp. 125-135. Recuperado de http://genomebiology.com/2011/12/8/125

[2] NVIDIA (2011). *Developer zone*. Recuperado de http://developer.nvidia.com/object/cuda.html

[3] Sanders, J. and Kandrot, E. (2010). *CUDA by example: An introduction to General-Purpose GPU Programming*. 1st ed. Addison-Wesley Professional. Ann Arbor, MI.

[4] Kirk, D. B. and Hwu, W. W. (2010). *Programming massively parallel processors: A hands-on approach.* 1st ed. Morgan Kaufmann Publishers Inc. San Francisco, CA.

[5] Benso, A, Di Carlo, S., Poltano, G. and Sevino, A. (2010). GPU acceleration for statistical gene classification. *IEEE Intl Conf. On Automation Quality and Testing Robotics* (AQTR 2010) 2: pp. 1-6.

[6] Churchill, G. A. (2002). Fundamentals of experimental design for cDNA microarrays. *Nat. Genet*. 32: pp. 490-495.

[7] Holloway, A. J., Van Laar, R. K., Tothill, R. W. and Bowtell, D. D. L. (2002). Options Available –From Start to Finish– For Obtaining Data from DNA Microarrays II. *Nature Genetics Supplement* 32: pp. 482-489.

[8] Dov, S. (2003). *Microarray bioinformatics*. 1st ed. Cambridge University Press. Cambridge, PA.

[9] Pan, W. (2002). A comparative review of statistical methods for discovering differentially expressed genes in replicated microarray experiments. *Bioinformatics* 18(4): pp. 546–554.

[10] Significance Analysis of Microarrays (SAM) (2011). *Supervised learning software for genomic expression data mining*. Recuperado de http://www-stat.stanford.edu/~tibs/SAM/