Paralelización de un Esferizador Geométrico

Paralellization of a Geometric Spherizer

Simena Dinas, Ing., José M. Bañon P., PhD. y Olmedo Arcila G., Ing.
Universidad del Valle
simdinas@univalle.edu.co, banon@univalle.edu.co, olarcila@univalle.edu.co

Recibido para revisión 26 de Marzo de 2007, aceptado 15 de Junio de 2007, versión final 22 de junio de 2007

***Resumen*—En este artículo se describe la paralelización de un esferizador geométrico utilizado en un detector jerárquico de colisiones. La paralelización se basa en la computación paralela con la utilización de la herramienta PVM (Parallel Virtual Machine). Se discute la estrategia utilizada junto a la implementación. Finalmente, se muestran resultados experimentales y se discuten los resultados obtenidos.**

***Palabras Clave*—Paralelización, Detector de Colisiones, Esferizador Geométrico, Computación paralela, Parallel Virtual Machine .**

***Abstract*— In this paper we present the parallelization of a geometric spherize to be used in a collision detection software. The PVM software (Parallel Virtual Machine) is used to parallelize the geometric spherizer. Finally, we discuss the implementation and the main experimental results obtained.**

***Key words*— Parallelization, Collision Detection, Geometric Spherize, Parallel Computing, Parallel Virtual Machine**

I. INTRODUCCIÓN

La detección de colisiones es un proceso importante en muchas aplicaciones de diversas disciplinas tales como la robótica móvil, la computación gráfica, la realidad virtual, el diseño asistido por computador etc,. La detección de colisiones es vital en la planificación del movimiento de los robots, y representa la componente más costosa desde el punto de vista computacional. Por ello se esta haciendo un gran esfuerzo internacional en el desarrollo de detectores de colisiones rápidos y eficientes.

La utilización de la computación paralela ha sido muy poco utilizada en las aplicaciones de la detección de colisiones [4]. En este trabajo se presenta la paralelización del esferizador geométrico [6] y [5] a través de la herramienta llamada PVM (Parallel Virtual Machine), que permite ejecutar el algoritmo en varios computadores conectados en red, los cuales cada uno realiza una porción de tarea para que el tiempo total de ejecución resulte óptimo. El esferizador geométrico está siendo utilizado para calcular las representaciones esféricas de poliedros en el desarrollo de un detector jerárquico de colisiones. Se discute la estrategia utilizada junto a la implementación en PVM y se dan los resultados obtenidos en una aplicación a un conjunto de lámparas.

La organización de este artículo se realiza de la siguiente manera. En la sección II se presentan los antecedentes del trabajo y se describe el esferizador geométrico utilizado. Generalidades sobre la computación paralela y PVM se presenta en la sección III. En la sección IV se describe la implementación del algoritmo paralelo del esferizador geométrico. En la sección V se dan las pruebas realizadas y los resultados. Finalmente, en la sección VI se dan las principales conclusiones.

II. EL ESFERIZADOR GEOMÉTRICO

A. Antecedentes Científicos

En la literatura existente muy pocos trabajos han abordado la difícil tarea de paralelizar los algoritmos de detección de colisiones. En particular, [4] obtienen en un computador paralelo la paralelización de un detector de colisiones en basado en una jerarquía de esferas. Recientemente, [5] desarrollan un esferizador geométrico basado en representaciones externas e internas de esferas y demostraron su desempeño frente a otras representaciones esféricas.

B. Descripción del Esferizador

El Esferizador geométrico [4] construye una representación del objeto basada en una jerarquía binaria de esferas exteriores e interiores al objeto. Las representaciones exterior e interior son importantes en el diseño de volúmenes limitantes para algoritmos jerárquicos de detección de colisiones. Por una parte, la utilización de representaciones exteriores permite garantizar que dos objetos no interseccionan, son útiles como tests de rechazo. Por otra parte, la utilización de la representación interior permite

asegurar que si dos esferas interiores de dos objetos se tocan entonces hay intersección entre los objetos. Son útiles como tests de aceptación de la colisión.

El algoritmo procede de la manera siguiente. Primero, calcula la esfera mínima que contiene al objeto, la cual representa el nivel 0 de la jerarquía y es almacenada en la raíz del árbol binario de la jerarquía. Seguidamente, se subdivide el objeto en dos subobjetos y se construye las esferas mínimas de cada una de las dos partes. Tales esferas constituyen el nivel 1 de la representación jerárquica de esferas. El algoritmo aplica recursivamente la misma subdivisión a cada una de las partes, obteniéndose de esta manera un árbol binario de esferas balanceado. El algoritmo se aplica hasta que se llega al nivel que representa la resolución requerida. Dado que por construcción la unión de todas las esferas de un mismo nivel contiene al objeto, cada nivel es una representación esférica del objeto.

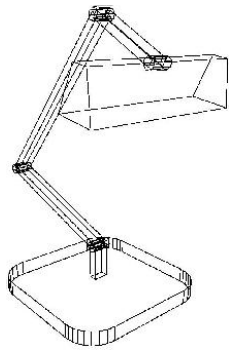

Figura 1. Esqueleto de la Lámpara

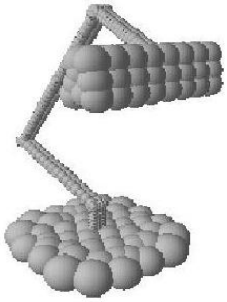

Figura 2. Lámpara de Nivel 6

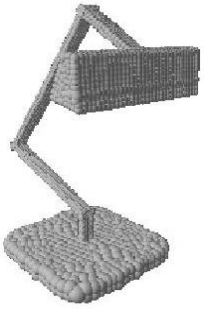

Figura 3. Lámpara con 5000 esferas

A partir del árbol binario de esferas es posible construir dos representaciones esféricas del interior y de la frontera del objeto respectivamente. La representación del interior del objeto está formada por esferas contenidas en el interior del objeto. La representación de la frontera del objeto, ó representación exterior, está formada por esferas que cubren las caras del objeto. Las Figuras 1, 2 y 3 muestran las representaciones esféricas de una lámpara obtenida por el esferizador mencionado. La lámpara, cuyo esqueleto se muestra en la figura 1, consta de diez piezas articuladas. La subdivisión esferica de nivel 6 se representa en la gráfica 2, mientras que la gráfica 3 muestra el resultado de tener 5000 esferas en cada una de las 10 piezas articuladas.

III. COMPUTACIÓN PARALELA Y PVM

A. Procesamiento Paralelo

Para aumentar el rendimiento de los computadores aparecen los sistemas multiprocesadores ó sistemas paralelos. Tales sistemas están formados por varios procesadores en comunicación, los cuales comparten el bus, el reloj, la memoria y los dispositivos periféricos. En general, la computación paralela es la capacidad de una aplicación o programa de ejecutar 2 o más de sus unidades funcionales o tareas simultáneamente entre los procesadores existentes.

B. La Máquina Virtual Paralela: PVM

Fué creada en 1989 por el Oak Ridge National Laboratory. En general, PVM es un paquete de software, para simular una máquina virtual que permite explotar de manera eficiente los recursos de una red de computadores heterogéneos. PVM está diseñado para unir recursos de cómputo y proveer a los usuarios de una plataforma paralela, de bajo costo, para correr aplicaciones, independientemente del número y tipo de computadoras que usen y donde se encuentren localizadas [3]. La filosofía de PVM es: *Mantener una interfaz de usuario simple y fácil de entender dejando a PVM el trabajo duro: mejorar la ejecución.* [2].

C. Métricas de Paralelización

Para medir la calidad de los algoritmos tanto en máquinas paralelas reales como en máquinas virtuales paralelas se utilizan los siguientes parámetros: i) el Speedup, ii) la Eficiencia y iii) la Paralelicibilidad. El $Speedup_p$ se define como:

$$Speedup_p = T_{sec} / T_{par} \quad (1)$$

donde T_{sec} es el tiempo que un computador paralelo ejecuta con un solo procesador el algoritmo secuencial más rápido, y T_{par} es el tiempo que toma al computador paralelo ejecutar el algoritmo paralelo en p procesadores. Esencialmente, el $Speedup_p$ es la ganancia del proceso paralelo con p procesadores frente al secuencial [2] y [7]. El valor óptimo del $Speedup_p$ es el crecimiento lineal respecto al número de procesadores [1]. La $Eficiencia_p$ se define como:

$$Eficiencia_p = Speedup_p / p \quad (2)$$

La Eficiencia se refiere al porcentaje de tiempo empleado en los procesos efectivos y [8]. Su valor óptimo es 1. La Paralelicibilidad$_p$ se define como:

$$Paralelicibilidad_p = T_1 / T_p \quad (3)$$

donde T_1 es el tiempo en que el computador paralelo ejecuta un algoritmo paralelo usando un procesador y T_p es el tiempo que toma al mismo computador paralelo ejecutar el mismo algoritmo paralelo usando *p* procesadores. La Paralelicibilidad es la ganancia proporcionada por el aumento de utilización de procesadores dentro de un sistema paralelo con el mejor algoritmo paralelo encontrado o el óptimo.

D. Paradigmas de Computación Paralela

La computación paralela se hace útil cuando los algoritmos presentan una estructura que permite que las tareas o las unidades funcionales se puedan dividir y ejecutar independientemente. De acuerdo a la estructura del algoritmo podemos distinguir tres paradigmas diferentes de utilización de PVM para paralelizar el algoritmo [2], [8]. y [9] : i) árbol, ii) Crowd, y iii) el Híbrido. El paradigma árbol es usado para la estructura de "divide y conquista"; consiste en dividir el problema en dos subproblemas que poseen la misma estructura del problema original. Posteriormente y de forma recursiva, los subproblemas se subdividen en subsubproblemas, y así sucesivamente, se realiza una subdivisión n-aria del problema hasta llegar a un nivel en donde resolver los subproblemas no sea tan costoso.

El paradigma Crowd consiste en la ejecución repetitiva del mismo algoritmo o proceso sobre conjuntos iguales o diferentes de datos. Está subdividido en dos categorías: i) Maestro-Esclavo: el maestro se encarga de las tareas de inicialización de los datos, de lanzar las tareas, de recuperar los valores, hacer cálculos finales y desplegar resultados y el esclavo se encarga de ejecutar métodos específicos que son requeridos por el maestro, y ii) Nodo- único: Este es el caso de un código ejecutable que se lanza a sí mismo y ejecuta una tarea definida generalmente de acuerdo a su condición (padre o hijo).
Finalmente, PVM soporta los dos paradigmas de forma combinada. Cuando se hace referencia a este tipo de combinación, se habla del paradigma híbrido.

IV. IMPLEMENTACIÓN

A. Paralelización del Esferizador Geométrico

En esta sección se presenta la implementación del esferizador teniendo en cuenta su estructura. Se presenta la estructura del esferizador enfatizando en las unidades funcionales que se van a paralelizar.

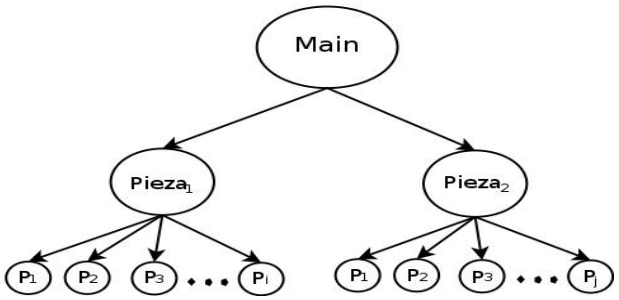

Figura 4. Grafo Secuencial del Esferizador Geom'etrico

En el primer nivel de la figura 4 aparece la unidad funcional principal *main*, que es la encargada de la captura de los datos, la creación de los procesos hijos, el envío y recepción de datos a los procesos hijos, y la visualización de los resultados. La unidad funcional *pieza$_i$* recibe como parametro un objeto (e.g. una lámpara) y se hace una subdivisión de cada una de las piezas que la conforman, creando las siguientes unidades funcionales, es decir, las subpiezas P_i que la conforman y que posteriormente se convierten en las unidades funcionales del último nivel.

Para realizar la paralelización, se implementó un algoritmo híbrido. La combinación de la estructura del problema en forma de árbol y la técnica de maestro-esclavo del paradigma crowd, que permite usar la replicación de tareas dió como resultado un algoritmo paralelo del Esferizador Geométrico que cuenta con paralelismo de datos en todos sus niveles, puesto que las unidades funcionales creadas son exactamente iguales y realizan el procesamiento sobre un conjunto de datos diferentes. A continuación se muestra el pseudocódigo del algoritmo paralelo.

B. Pseudocódigo del Algoritmo Paralelo del Esferizador Geométrico

```
main (maestro)
    Leer_Datos();
    Levantar_Procesos_Esclavos();
    for i <- 1 to n do
        construirArbol(esclavos);
    for i <- 1 to n do
        Enviar_Pieza_A_Cada_Esclavo();
    for i <- 1 to n do
        Recibir_Pieza_De_Cada_Esclavo();
construirArbol (Esclavo Nivel 1)
    Recibir_Datos();
    for i <- 1 to n do
        construir(subpieza);
    for i <- 1 to m do
        Enviar_Subpieza_A_Cada_Esclavo();
    for i <- 1 to m do
        Recibir_Subpieza_De_Cada_Esclavo();
    Reconstruir_Pieza()
    Enviar_Pieza_Al_Padre();
construir (Esclavo Nivel 2)
    Recibir_Datos();
    Construir_Arbol_Subpieza();
    Enviar_Subpieza_Al_Padre();
```

V.EXPERIMENTACIÓN Y DISCUSIÓN DE LOS RESULTADOS

En esta sección se describen las pruebas realizadas para comparar el algoritmo secuencial del esferizador con el algoritmo paralelo del esferizador implementado en PVM. La prueba consiste en aplicar el esferizador a un conjunto de 10 lámparas. Cada lámpara se representa con 1600 esferas repartidas en las diez piezas de acuerdo con la distribución referida en la tabla 1. Cada prueba se realiza 10 veces y de ellas se calcula el tiempo de ejecución promedio y mínimo.

Tabla 1. Composición de las dos Lámparas a Paralelizar

Pieza	#Esferas
Base (1)	300
Reflector (1)	300
Tronco (4)	150
Articulación (4)	100

Para las pruebas se usó un cluster de 40 computadores de los cuales 23 eran IBM con procesador Pentium IV 1.6 GHz y 256 MB de memoria RAM y 17 eran DELL con procesador Pentium IV 2.4 GHz y 512 MB de memoria RAM. Ambos ambientes con plataforma PC-Linux, bajo el compilador g++ 2.95.3 e interconectados por medio de switchs, con transferencia de datos 100Mbps.

A. Algoritmo Secuencial

En la tabla 2 se presentan los tiempos promedio y mínimo, del algoritmo secuencial.

Tabla 2. Tiempo Promedio y Mínimo de Ejecución del Algoritmo Secuencial

Sec	TProm	TMin
	21.10	21.27

B. Algoritmo Paralelo

En la tabla 3 se representan los tiempos de ejecución, promedio y mínimo, así como el número de equipos utilizados obtenidos con el algoritmo paralelo implementado en PVM.

Tabla 3. Tiempo Promedio y Mínimo de Ejecución del Algoritmo Paralelo

# p	TProm	TMin
1	-----	-----
2	10.11	9.81
3	8.09	7.85
4	6.35	5.95
5	5.25	4.96
6	4.62	4.13
7	3.94	3.67
8	3.64	3.37
9	3.68	3.29
10	3.27	3.12
11	2.83	2.59
12	2.80	2.64
13	2.58	2.36
14	2.45	2.19
15	2.36	2.20
16	2.28	1.97
17	2.09	1.91
18	2.21	1.87
19	2.21	1.87
20	2.00	1.69
21	2.04	1.63
22	2.14	1.67
23	1.94	1.72
24	1.96	1.69
25	2.01	1.57
26	2.04	1.48
27	2.13	1.65
28	2.28	1.73
29	2.37	1.56
30	2.91	2.13
31	2.40	1.44
3	2.32	1.80
33	2.33	1.34
34	2.10	1.45
35	2.40	1.41
36	2.36	1.23
37	2.17	1.50
38	2.39	1.51
39	2.10	1.37
40	2.03	1.58

C. Discusión de Resultados

Analizando los resultados consignados en las tablas 2 y 3 se observa que con la implementación paralela del esferizador se obtienen ganancias del 90% con tan solo 23 procesadores; Después de 23 procesadores, el sistema no presenta mejoras significativas debido al manejo de los recursos.

Dicho de otra manera, si un algoritmo a_n crea n procesos hijos iguales con un tiempo T cada uno, el tiempo de ejecución secuencial del algoritmo a_n es:

$$T_{secuencial} = T_{a_n} \quad n \quad T \tag{4}$$

donde Ta_n es el tiempo que demora la ejecución del algoritmo a_n sin la ejecución de sus procesos hijos. El tiempo de ejecución de a_n paralelizando los *n* procesos hijos, es aproximadamente:

$$T_{paralelo} = T_a \quad T \tag{5}$$

En un sistema paralelo con *n* o más procesadores, mientras que en un sistema en Cluster con *n* o más procesadores, se hace necesario agregar el costo del tiempo de espera o inactividad (t_{idle}) y el tiempo de comunicación (t_{comm}) es decir:

$$T_{cluster} = T_a \quad T \quad T_{comm} \quad T_{idle} \tag{6}$$

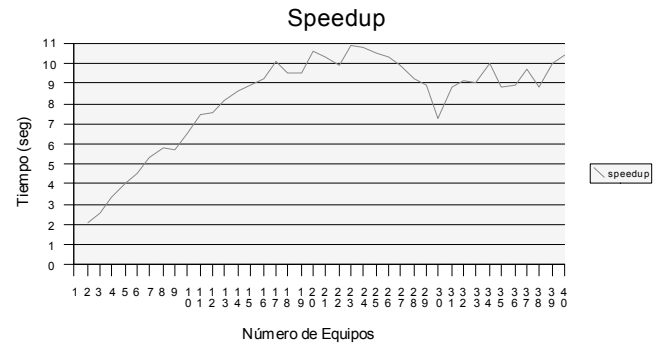

Figura 5. Speedup del Esferizador Geométrico

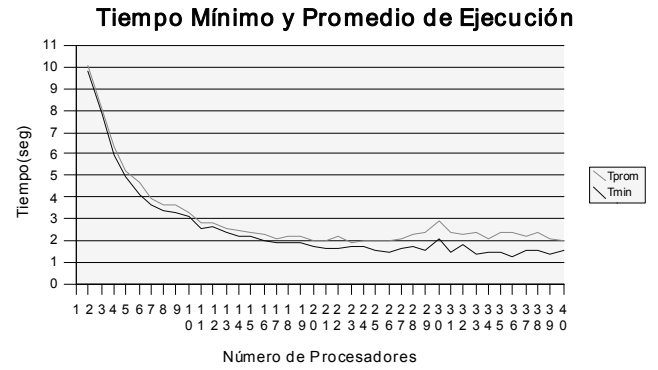

Figura 6. Tiempo Mínimo y Promedio de Ejecición del Esferzador Geometrico Paralelo

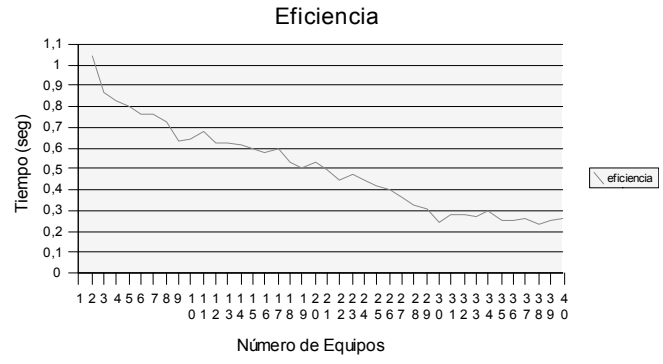

Figura 7. Eficiencia del Esferizador Geométrico

En un Cluster, el tiempo de comunicación depende, entre otras cosas, de la infraestructura de la red, si la conexión es por medio de switch, el intercambio de paquetes se hacen punto a punto, por lo tanto el tiempo de comunicación se reduce al tiempo de lectura del proceso receptor. Por otra parte, si los dispositivos de comunicación son concentradores, el sistema de multicast que aumenta la posibilidad de colisiones y por ende pérdida de datos. En olas figuras 5, 6 y 7, se grafican los resultados condensados en las tablas 2 y 3.

Hasta completar los 17 equipos, este algoritmo tiene un desempeño deseable, tanto los tiempos que alcanza como la ganancia con respecto al procesamiento paralelo, muestran un buen resultado. Si se tiene en cuenta que el número de procesos o tareas que se crean con este algoritmo es de ciento once (111) incluido el programa principal o el maestro, cada uno de los 17 procesadores tiene un promedio de 6.5 tareas . A partir de los 17 procesadores tanto la Eficiencia como el Speedup se vuelven constantes, por lo tanto la solución es no aumentar más equipos a la máquina virtual por que generan más tareas para PVM y sobrecosto en el tiempo de lectura de los buffers que están ejecutando procesos. Para este algoritmo no fue posible calcular la Paralelicibilidad , es necesario para esta medida ejecutar el algoritmo paralelo en un sólo procesador de la máquina virtual y PVM no soporta manejar tantas tareas en un sólo procesador.

VI. CONCLUSIONES

La implementación de la paralelización mejoró el tiempo del algoritmo secuencial hasta en un 90 %.

La gráfica de los tiempos de ejecución en paralelo de la aplicación permite detectar el número de equipos necesario para obtener un buen resultado; tal como se ve en la prueba, agregar procesadores a la máquina virtual aumenta el tiempo de ejecución, por tanto, los recursos adicionales degradan el desempeño.

Las métricas permiten detectar qué tan eficiente es agregar procesadores a la máquina virtual, hasta que punto aplicar la paralelización es mejor y cual es el comportamiento del algoritmo frente a la paralelización.

El procesamiento paralelo mediante sistemas cluster es una alternativa interesante en términos de costo-beneficio para la ejecución de aplicaciones que requieren de un alto nivel de procesamiento, que consumen considerablemente los recursos de la máquina, que requieren de resultados en tiempo real (sin ser aplicaciones críticas) o que cuentan con algoritmos computacionalmente complejos.

Un sistema basado en cluster unido con una herramienta de desarrollo de aplicaciones paralelas y más particularmente con PVM, dá como resultado un ambiente de ejecución adecuado para aplicaciones que requieren grandes cantidades de procesamiento a bajo costo.

REFERENCIAS

[1] Wolfgang Ertel. On the Definition of Speedup. In Parallel Architecture and Lenguage Europe, 1993. URL ftp.icsi.berkeley.edu/pub/techreport/1993/tr-93-069.ps.gz.

[2] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. PVM 3 User's Guide and Reference Manual. ORNL/TM-12187 Engeneering Phisics and Mathematics Division, Oak Ridge National Laboratory, Septiembre 1994a. URL http://citiseer.ist.psu.edu/ geist94pvm.html

[3] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. PVM: Parallel Virtual Machine A User´s Guide and Tutorial for Network Parallel Computing. MIT Press, Cambridge, MA, USA, 1994b. ISBN 0-262-57108-0 (Paperback). URL http://www.netlib.org/pvm3/book/pvm-book.html.

[4] Begoña Martinez-Salvador, Angel P. del Pobil, and Miguel Perez-Francisco. Very Fast Collision Detection for Practical Motion Plannig Part II: The Parallel Algorithm. Proceeding of the IEEE International Conference on Robotics and Automation., pages 644{649, 1998.

[5] Carlos Andres Muñoz, Olmedo Arcila Guzmán, and José María Bañon. Nuevas Heurísticas para la Representación Eficiente de Objetos por Jerarquías de Esferas. CLEI 2001.

[6] Carlos Andres Muñoz, Olmedo Arcila Guzmán, and José María Bañon. Una Nueva Representación Jerárquica Basada en Esferas Sencilla, Rápida y EÂ¯ciente Información Tecnológica, 13 (4):141{147, 2002.

[7] Sartaj Sahni and Ventak Thanvantri. Parallel Computing: Performance Metrics and Models. Computer & Information Sciences Departament, University of Florida, 1996. URL ftp.cis.ufi.edu/pub/tech-report/tr96/tr96-008.ps.Z.

[8] B. K. Schmidt and V. S. Sunderam. Empirical Analysis of Overheads in Cluster Environments. Concurrency: Practic and Experience, 1994. URL http://suif.stanford.edu/bks/publications/empanal.ps.

[9] M. R. Steed and M. J. Clement. Performance Prediction of PVM Programs. The MIT Press Cambridge,Massachusetts, London, England, 1995. URL http://citeseer.ist.su.edu/steed95performance.html.

Simena Dinas (Cali, Colombia, 1979). Ingeniera de Sistemas de la Universidad del Valle, Cali, Colombia (2005). Estudiante de Maestria en Ingeniería de Sistemas y Computación en la Universidad del Valle, Cali, Colombia (2005)

Dr. José M. Bañón (Pamplona, España, 1954). Profesor Titular de la Universidad del Valle en Cali. Físico por la Universidad Complutense de Madrid, España (1977). Especialista en Ingeniería Nuclear por el Instituto de Estudios Nucleares de Madrid, España (1979). Doctor de Estado en Física por la Universidad Paris VI , Francia (1985).
El estuvo como Profesor visitante en el Laboratorio de Robótica de la Universidad de Stanford, USA (1989-1991). Es coautor de varias comunicaciones al Congreso Latinoamericano de Estudios en Informática (CLEI). Su principal campo de investigación es la Robótica Computacional. Ha participado en varios Encuentros de Geometría Computacional en España.

Olmedo Arcila Guzman (Palmira, Colombia, 1972). Ingeniero de Sistemas de la Universidad del Valle, Cali, Colombia (2000). Estudiante de Doctorado en la Universidad del Valle, Cali, Colombia. Es coautor de varias comunicaciones al Congreso Latinoamericano de Estudios en Informática (CLEI). Ha participado en varios Encuentros de Geometría Computacional en España.