# 66:20 Organización de Computadoras
# Trabajo práctico 2: Profiling y optimización

## 1. Objetivos

Familiarizarse con las técnicas y herramientas de profiling y optimización de software, evaluando y optimizando un programa que implementa la simulación de un ecosistema de presas y depredadores [2].

## 2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

## 3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido. Por este motivo, el día de la entrega deben concurrir todos los integrantes del grupo.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

## 4. Recursos

Utilizaremos las herramienta `gprof` [3] y/o `cachegrind` [4] para evaluar la performance de las diversas áreas del programa y ver en cuáles se está empleando más tiempo.

## 5. Introducción

El planeta WA-TOR [1] está completamente cubierto de agua, y en él habitan principalmente plancton (que se supone infinito), peces (que viven

del plancton), y tiburones (que comen peces). Este planeta es toroidal, de manera que su superficie se puede representar como una matriz rectangular, en la que al salir por un lado se vuelve a ingresar por el lado opuesto. Este planeta está modelado de la siguiente manera:

- Cada celda puede contener o bien un pez, o un tiburón, o estar vacía.
- En cada iteración del algoritmo, el habitante de cada celda puede intentar moverse:
  - Si el habitante es un tiburón, y tiene algún pez en los casilleros adyacentes, se mueve a uno de estos casilleros elegido al azar, comiéndose al pez. De otra manera, si hay un casillero libre, se mueve a éste.
  - Si el habitante es un pez, y tiene algún casillero vacío adyacente, se mueve a alguno de estos al azar.
  - Al alcanzar una cierta edad, tanto los peces como los tiburones se reproducen, pero sólo si se pueden mover en ese turno. Al moverse, dejan detrás a otro habitante de su misma especie, y tienen que esperar una cantidad de turnos para volver a reproducirse.
  - Si un tiburón pasa cierta cantidad de turnos sin comer, muere y su casillero queda vacío.

Los parámetros de esta simulación son entonces la edad a la que se reproducen los peces, la edad a la que se reproducen los tiburones, y la cantidad de tiempo que pueden pasar los tiburones sin comer.

El objetivo del trabajo es evaluar el desempeño de un programa que simula la evolución de un grupo de peces y tiburones en el planeta WATOR, y de ser posible mejorarlo.

# 6. Programa

Se trata de una versión en lenguaje C de la simulación del planeta WATOR. El programa recibe un nombre de archivo en el que se van dejando las cantidades de peces y tiburones en cada turno, y simula 1000 turnos (o *cronones*) en un planeta de 32x32 celdas.

# 7. Mediciones

## 7.1. Profiling

Si bien la medida más simple y más definitiva de evaluar las mejoras en desempeño de un programa es el tiempo de ejecución, es difícil saber cómo optimizarlo si no tenemos claro cómo se distribuye el tiempo de ejecución

entre las diferentes partes de éste. Una manera recomendada de proceder a la optimización de un programa consiste en ver en qué emplea más tiempo, y si esa sección de programa se puede hacer más eficiente. Por eso será necesario entonces detectar y documentar los principales cuellos de botella. Para esto, usaremos dos herramientas: cachegrind [4] y gprof [3].

### 7.2. Caso de prueba

El caso que queremos analizar en cuanto a la performance del cache es el de una computadora con un cache L1D 2WSA de 256B, con 64 bytes por línea. Se simula en el `cachegrind` como

```
$ valgrind --tool=cachegrind --D1=256,2,64 ./wator wator.txt
```

El miss rate del programa se puede ver con el cachegrind de la manera en que acabamos de ver. El `gprof` no permite la definición del cache donde corre el programa, pero permite ver cómo se reparte el tiempo de ejecución entre las distintas funciones y cuánto se llama a cada una. La función `time` de Linux también permite ver cuánto tiempo pasa el programa en modo usuario, cuánto en modo kernel, y cuánto dura la ejecución del programa.

En clase veremos algunos ejemplos de utilización de estas herramientas.

### 7.3. Documentación

Es necesario que el informe incluya una descripción detallada de las técnicas y procesos de medición empleados, y de todos los pasos involucrados en el mismo, ya que forman parte de los objetivos principales del trabajo.

## 8. Informe

El informe deberá incluir:

- Este enunciado;
- Análisis de la performance del programa tal como está presentado, para el caso de prueba.
- Optimizaciones planteadas y análisis de los cambios en performance.
- El código fuente completo del programa modificado, en dos formatos: digital[1] e impreso en papel.

[1] No usar diskettes: son propensos a fallar, y no todas las máquinas que vamos a usar en la corrección tienen lectora. En todo caso, consultá con tu ayudante.

## 9. Fecha de entrega

La última fecha de entrega y presentación es el jueves 13 de Diciembre de 2012.

## Referencias


[1] "Sharks and fish wage an ecological war on toroidal planet WA-TOR", A.K.Dewdney, Scientific American,`http://home.cc.gatech.edu/biocs1/uploads/2/wator_dewdney.pdf`.

[2] WA-TOR, Wikipedia, `http://en.wikipedia.org/wiki/Wa-Tor`.

[3] GNU profiler, `http://sourceware.org/binutils/`.

[4] Cachegrind (parte del Valgrind), `http://valgrind.org/docs/`.