Universidad Nacional de La Plata
Facultad de Informática

Magister en Cómputo de Altas Prestaciones

Infraestructura para el Análisis de Rendimiento

Alumno: Andres More

Director: Dr Fernando G Tinetti

Resumen

- En HPC las aplicaciones son construidas por los especialistas del dominio del problema, necesitando un método sistemático y automático para soportar el análisis de rendimiento.
- Se trabajó en un generador automático de reportes de rendimiento para aplicaciones utilizando tecnología OpenMP sobre GNU/Linux.
- La infraestructura brinda: (1) información de contexto del programa y el sistema, (2) resume el comportamiento, perfil de ejecución, cuellos de botella y la utilización de los recursos disponibles.

Motivación

- En HPC los desarrolladores no son especialistas en rendimiento.
- El código optimizado puede ejecutarse órdenes de magnitud mejor.
- Mayor complejidad de implementación, depuración y optimización.
- Esto impacta en la investigación y desarrollo de un grupo de trabajo.

Objetivos

- Desarrollo de soporte para el análisis de aplicaciones HPC.
- Definición e implementación de un procedimiento sistemático.
- Generación de un reporte de rendimiento.

Metodología

- Revisión del estado del arte en el análisis de rendimiento
- Formulación de un procedimiento para simplificar la tarea
- Identificación de la información a resumir y graficar
- Automatización del procedimiento sistemático
- Aplicación sobre núcleos de cómputo conocidos
- Documentación de la experiencia

Contribuciones

- Estudio de Multiplicaciones de Matrices (Reporte Técnico)
- *Optimizing Latency in Beowulf Clusters* (Artículo)
- Comparación de Implementaciones de BLAS (Reporte Técnico)
- *Intel Cluster Ready* e *Intel Cluster Checker* (Sección Libro)
- *Intel Xeon Phi Coprocessor Programming* (Reseña Libro)
- *Lessons Learned from Contrasting BLAS Implementations* (Artículo)
- *Hotspot: Framework to Support Performance Optimization* (Artículo)

Análisis de Rendimiento

- **Rendimiento**: trabajo comparado contra recursos
- **Paralelismo**: trabajo conjunto
- **Ley de *Amdahl***: mejora en función del componente serial
- **Ley de *Gustafson***: mejora en el problema
- **Métricas**: latencia, cantidad de instrucciones, rendimiento sobre costo o energía consumida, cualquier recursos utilizado.
- **Técnicas de Análisis**: iteraciones de medir, localizar, optimizar y comparar.

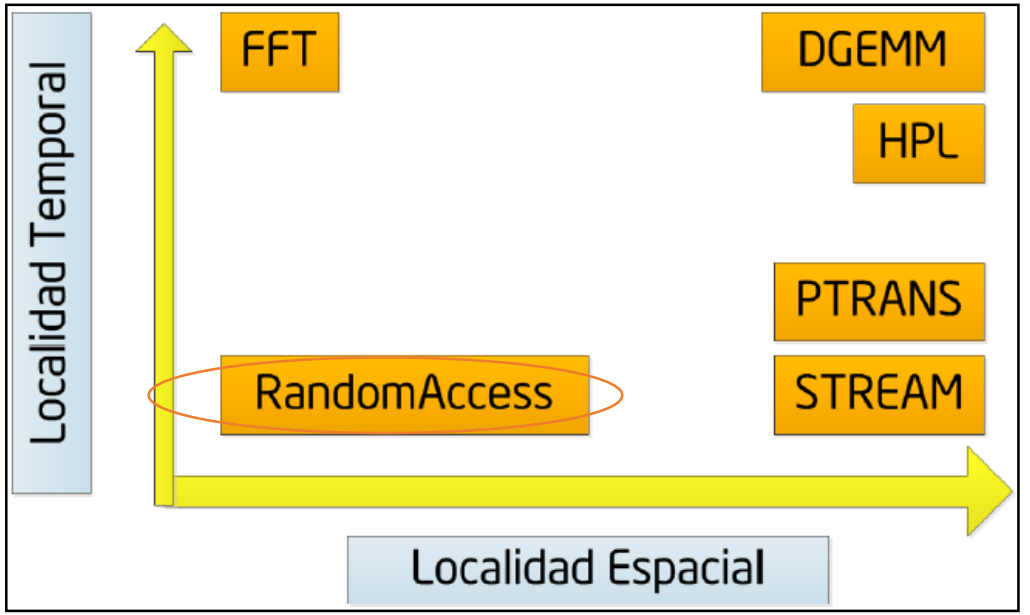
	Porcentaje de Paralelismo					
Número de Procesadores	0.1	0.3	0.5	0.8	0.9	0.95
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.05	1.14	1.33	1.60	1.82	1.90
4	1.08	1.23	1.60	2.29	3.08	3.48
8	1.10	1.28	1.78	2.91	4.71	5.93
16	1.10	1.31	1.88	3.37	6.40	9.14
32	1.11	1.32	1.94	3.66	7.80	12.55
64	1.11	1.33	1.97	3.82	8.77	15.42
128	1.11	1.33	1.98	3.91	9.34	17.41
256	1.11	1.33	1.99	3.95	9.66	18.62
512	1.11	1.33	2.00	3.98	9.83	19.28
1024	1.11	1.33	2.00	3.99	9.91	19.64
2048	1.11	1.33	2.00	3.99	9.96	19.82
4096	1.11	1.33	2.00	4.00	9.98	19.91
8192	1.11	1.33	2.00	4.00	9.99	19.95
16384	1.11	1.33	2.00	4.00	9.99	19.98
32768	1.11	1.33	2.00	4.00	10.00	19.99
65536	1.11	1.33	2.00	4.00	10.00	19.99

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

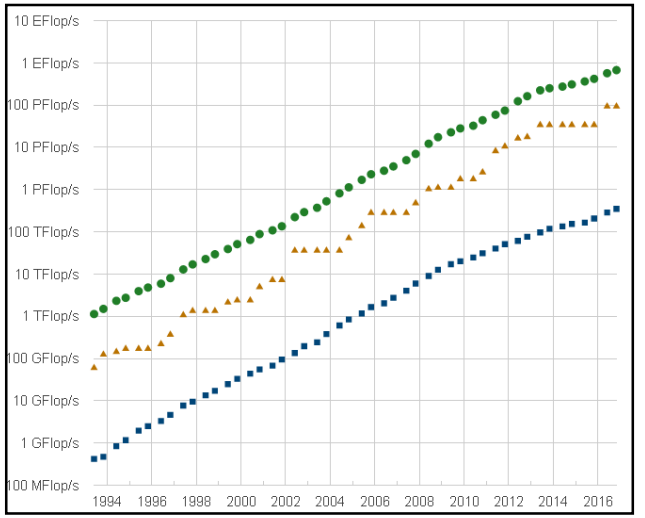
$$Speedup(P) = P - \alpha \times (P-1)$$

Herramientas

- Pruebas de Rendimiento
  - STREAM, Linpack, IMB, HPC Challenge
- Utilización y Aplicación de Herramientas
  - Capacidad del Sistema
  - Medición y Perfil de la Ejecución
  - Comportamiento de Aplicación, Librerías y Sistema
  - Vectorización y Contadores de *Hardware*



Característica	Herramientas
Capacidad del sistema	Benchmark HPCC
Medición de ejecución	`time`, `gettimeofday()`, `MPI_WTIME()`
Perfil de ejecución	profilers: `gprof`, `perf`
Comportamiento de la aplicación	profilers: `gprof`, `perf`
Comportamiento de librerías	profilers: valgrind, MPI vampir.
Comportamiento del sistema	profilers: oprofile, perf
Vectorización	compilador: gcc
Contadores en *hardware*	oprofile, PAPI, perf



Problema

- Análisis de Rendimiento
  - **Interacción Humana**: existen errores involuntarios, requiere estricta disciplina
  - **Manejo de Herramientas**: correcto uso, aplicación de estadística
  - **Recopilación y Representación de Datos**: como obtener métricas y su representación
  - **Optimización Temprana**: mejoras sin impacto significativo
  - **Implementación Teórica de Algoritmos**: implementación directa versus librerías
- Métodos de Optimización
  - **Código**: soporte en la predicción de saltos, utilización de inlining y unrolling
  - **Ejecución**: uso de instrucciones vectoriales, reducción de la utilización de registros
  - **Memoria**: disminución de transferencia, fallas de cache, alineación, localidad
  - **Precarga**: uso de instrucciones explicitas cuando la predicción no es efectiva
  - **Punto Flotante**: reglas de redondeo y representación

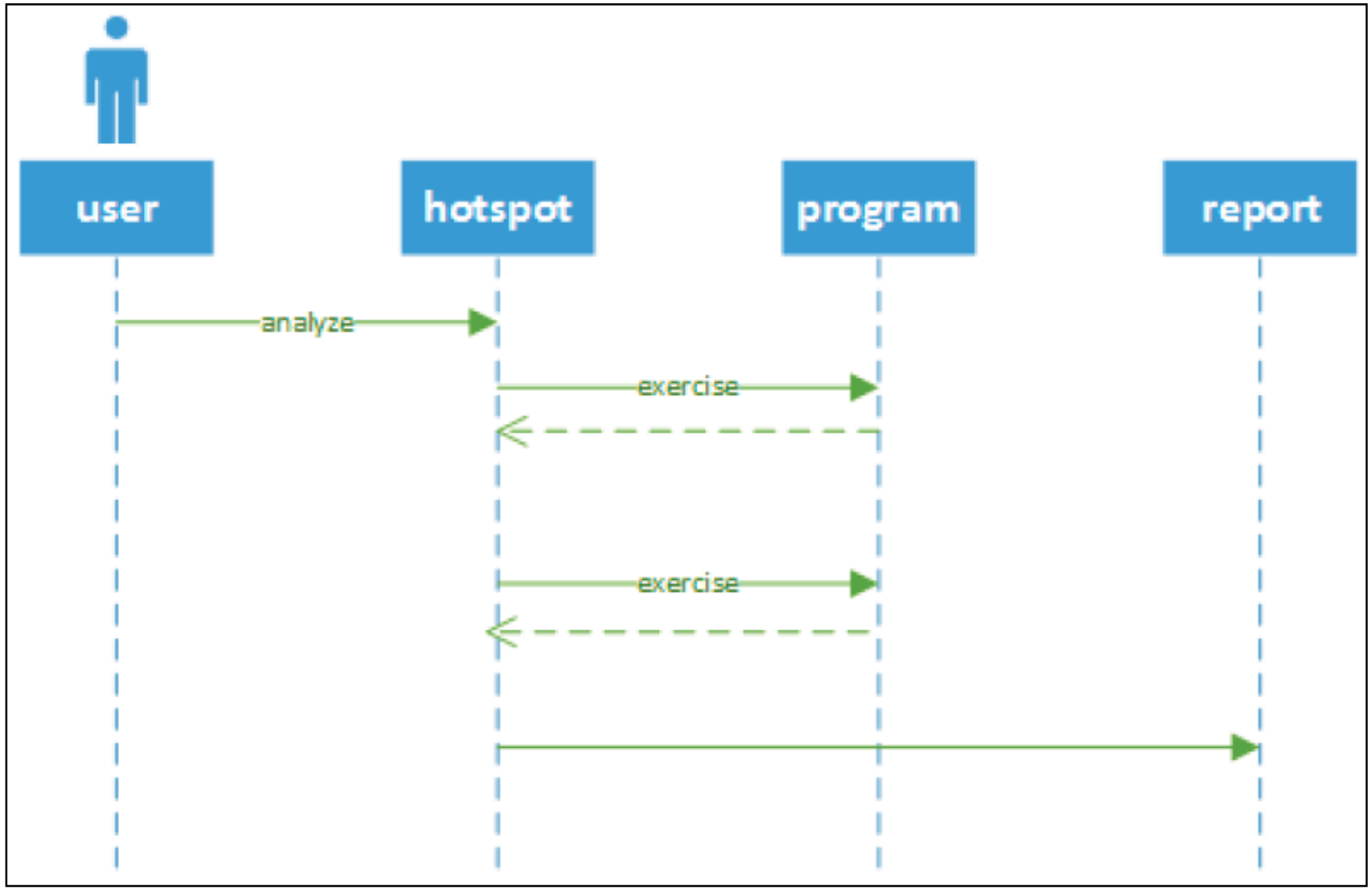
Infraestructura de Soporte

- **Reusabilidad**: aplicable a cualquier aplicación, depende solamente de herramientas del sistema, permite uso iterativo.
- **Configuración**: como compilar, configurar y ejecutar la aplicación.
- **Portabilidad**: implementación portable, de código abierto.
- **Extensibilidad**: incorporación de nuevas herramientas, gráficos o secciones utilizando los mismos comandos a ejecutar manualmente.
- **Simplicidad**: reutilización de herramientas, generación de archivos de soporte, generación de un reporte completo entre días de trabajo.

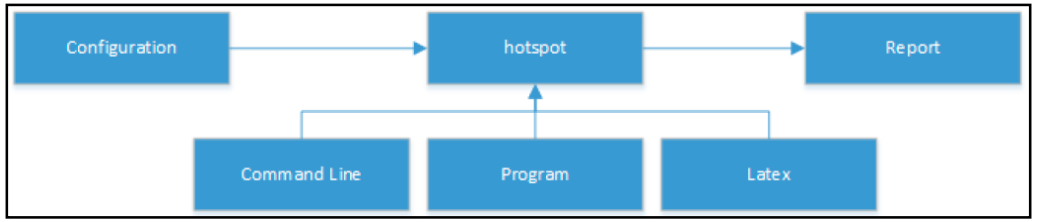
Procedimiento

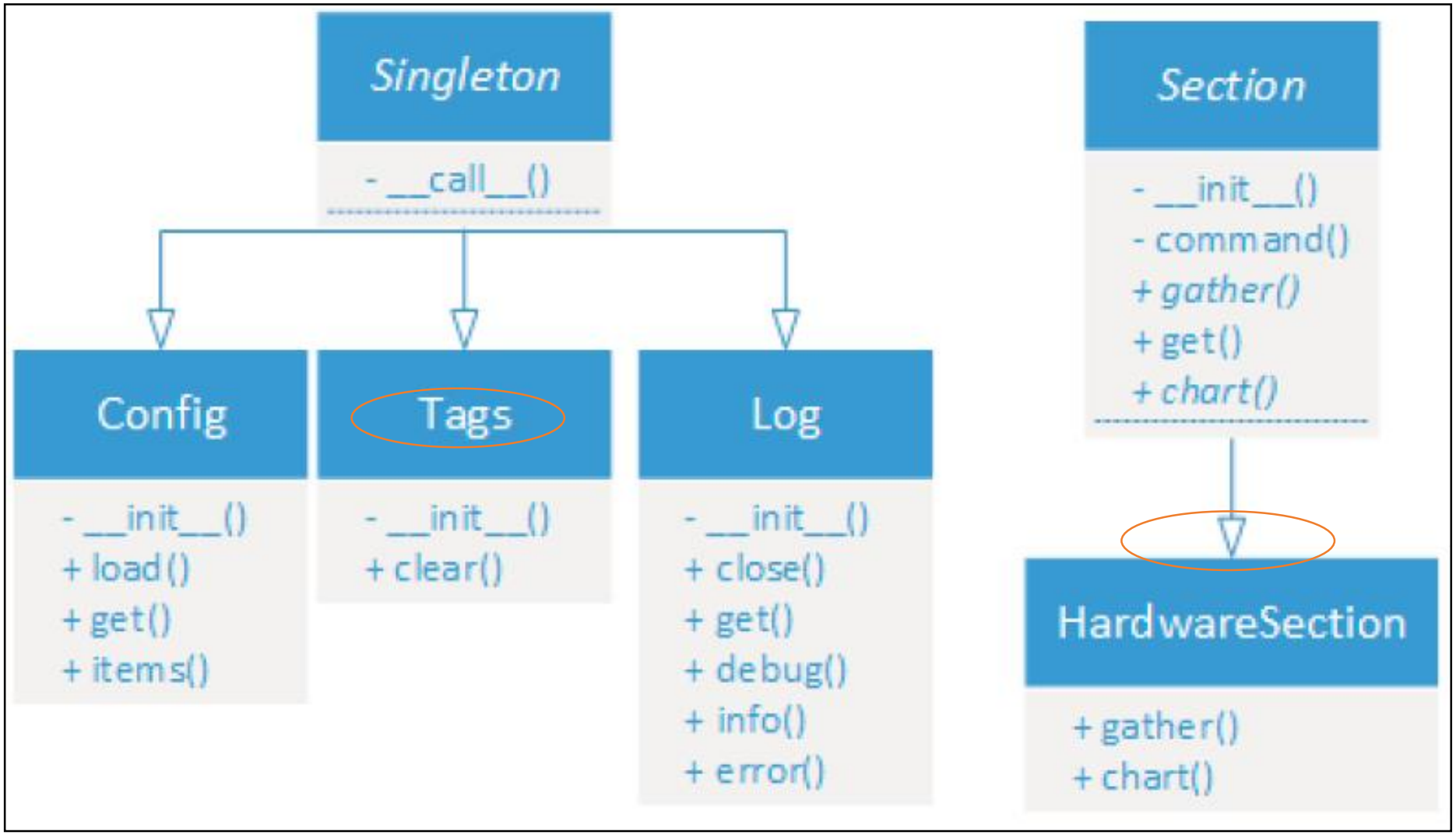
- **Pruebas de Referencia**: dimensiona capacidad
- **Compilación del Binario**: instrumentación
- **Estabilidad de Resultados**: asegura impacto
- **Utilización de Recursos**: patrón de escalamiento
- **Perfil de Código**: comportamiento
- **Optimización**: interacción humana

Arquitectura y Funcionamiento



Diseño de Alto y Bajo Nivel





Implementación

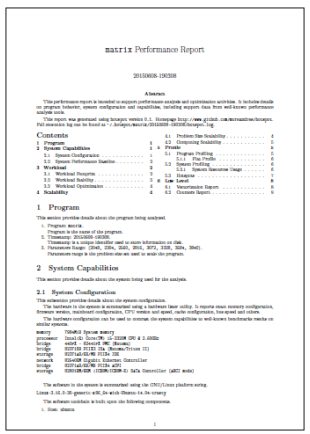
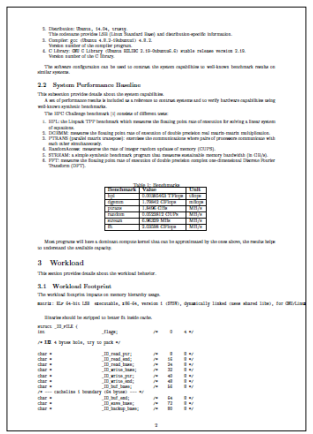
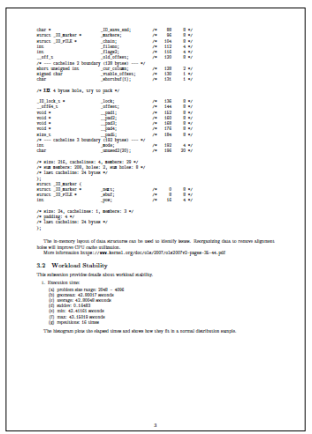
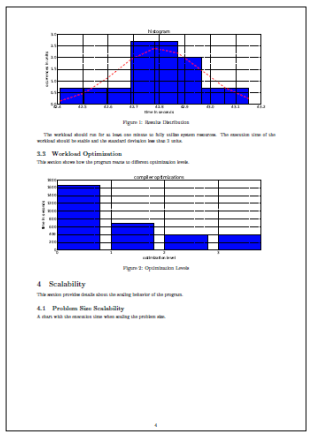
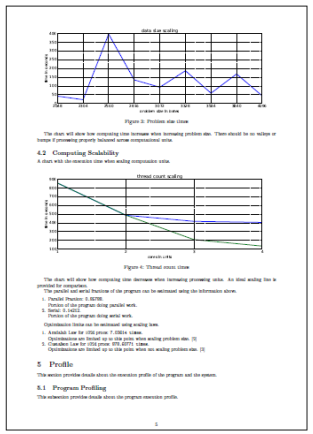
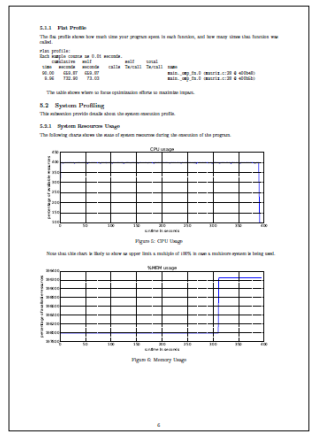
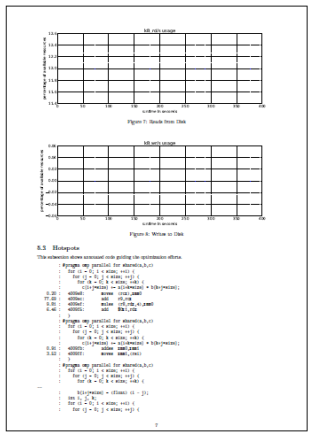
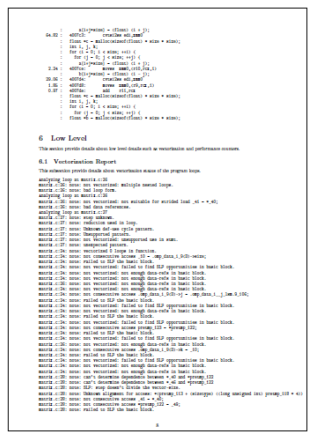
- **Desarrollo**: Ubuntu 14.04.1 LTS, Python 2.7.6, publicado en PIP como hotspot 0.3 con licencia GPLv2. Usa matplotlib, numpy. Pylint 8.4/10.
- **Configuración**: definición de como se compila, limpia, y ejecuta la aplicación bajo diferentes tamaños del problema a resolver.

```
$ hotspot --help
usage: hotspot [-h] [-v] [--config CONFIG] [--debug]

Generate performance report for OpenMP programs.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
  --config CONFIG, -c CONFIG
                        path to configuration
  --debug, -d           enable verbose logging

Check https://github.com/moreandres/hotspot for details.
```

```
# hotspot configuration file

[hotspot]
# python format method is used to pass parameters

# range is a seq-like definition for problem size
range=1024,2048,256

# cflags are the compiler flags to use when building
cflags=-O3 -Wall -Wextra

# build is the command used to build the program
build=CFLAGS='{0}' make

# clean is the cleanup command to execute
clean=make clean

# run is the program execution command
run=OMP_NUM_THREADS={0} N={1} ./{2}

# count is the number of runs used to check for workload stabilization
count=16
```

Consideraciones Generales del Reporte

- **Formato Portable**: documento interoperable.
- **Hipervínculos**: rápido acceso a la información en bruto.
- **Explicación**: introduce el propósito de cada sección.
- **Tendencia y Comportamiento Ideal**: comparación directa.
- **Referencias**: permite revisar la base teórica.

Consideraciones Particulares del Reporte

- **Resumen**: versión e información de la infraestructura
- **Contenido**: índice de secciones con hipervínculos
- **Programa**: versión, fecha del análisis y parámetros de entrada
- **Capacidad del Sistema**: *hardware*/*software*, pruebas de referencia
- **Carga de Trabajo**: tamaño en memoria y alineación de estructuras, estabilidad del caso de prueba, impacto de optimizaciones del compilador
- **Escalabilidad**: tendencia al escalar cómputo o tamaño del problema
- **Perfil de Ejecución**: funciones y líneas de código más utilizadas
- **Bajo Nivel**: ensamblador y reporte de vectorización de ciclos

Sistema de Prueba

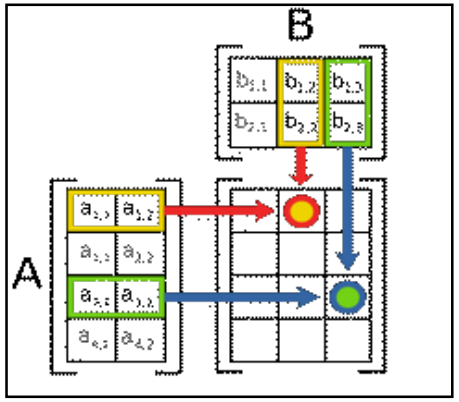
1. Host: `ubuntu`
2. Distribution: `Ubuntu, 14.04, trusty`.
   This codename provides LSB (Linux Standard Base) and distribution-specific information.
3. Compiler: `gcc (Ubuntu 4.8.2-19ubuntu1) 4.8.2`.
   Version number of the compiler program.
4. C Library: `GNU C Library (Ubuntu EGLIBC 2.19-0ubuntu6.6) stable release version 2.19`.
   Version number of the C library.

- **Sistema de Prueba**: *hardware* disponible refleja la máquina virtual utilizada para ejecutar las pruebas, el listado de *software* instalado es correcto, los datos de referencia fueron extraídos correctamente.
- **Pruebas de Rendimiento**: dimensionan la capacidad práctica del sistema para diferentes núcleos de cómputo.

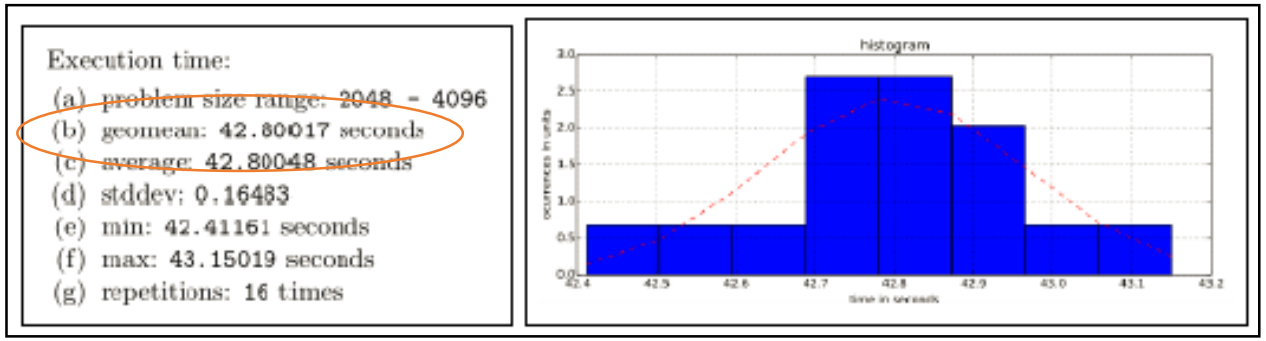
```
memory      7984MiB System memory
processor   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
bridge      440FX - 82441FX PMC [Natoma]
bridge      82371SB PIIX3 ISA [Natoma/Triton II]
storage     82371AB/EB/MB PIIX4 IDE
network     82540EM Gigabit Ethernet Controller
bridge      82371AB/EB/MB PIIX4 ACPI
storage     82801HM/HEM (ICH8M/ICH8M-E) SATA Controller [AHCI mode]
```

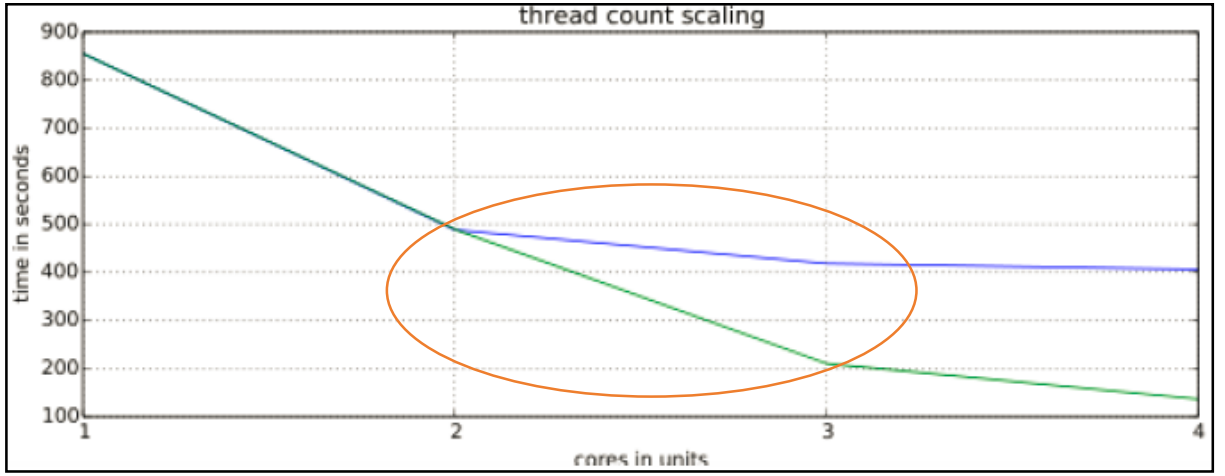
Benchmark	Value	Unit
hpl	0.00365811 TFlops	tflops
dgemm	1.72977 GFlops	mflops
ptrans	1.58619 GBs	MB/s
random	0.0544077 GUPs	MB/s
stream	7.00091 MBs	MB/s
fft	2.32577 GFlops	MB/s

Caso de Aplicación: Matrices

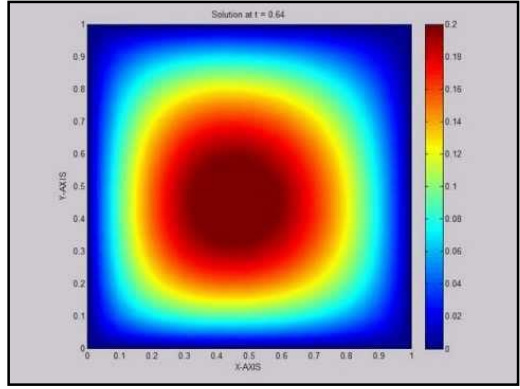


- **Multiplicación de Matrices**: operación fundamental en diversos campos de modelado y simulación.
- **Conclusiones**: caso de prueba estable, las optimizaciones del compilador tienen impacto, el escalamiento de unidades de cómputo muestra mejoras pero al agrandar el problema el tiempo no crece monotónicamente.





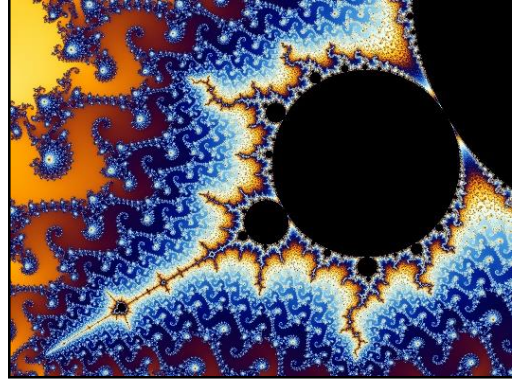
Caso de Aplicación: Calor 2D



- **Calor en 2 Dimensiones**: simulación de transferencia de calor en celdas dispuestas un plano bidimensional.
- **Conclusiones**: el caso de prueba es estable, optimizaciones del compilador tienen un impacto acotado, el tiempo de ejecución no crece monotónicamente, el paralelismo identificado muestra una mejora máxima de 3x al incrementar unidades de cómputo, hay dos cuellos de botella a resolver, el 35% del tiempo se invierte en mover datos en memoria, los ciclos no están vectorizados.

```
      :   /* set boundary values */
      :   for (i = 0; i < CRESN; i++)
      :     {
      :       if (i < 256 || i > 768)
      :         solution[cur_gen][i][0] = solution[cur_gen][i][1];
 6.77 :   400e3b:       movsd  0x8(rdx),xmm0
36.41 :   400e40:       movsd  xmm0,(rdx)
 2.92 :   400e44:       jmpq   400db7 <compute_one_iteration+0x37>
```

```
Analyzing loop at heat2d.c:46
heat2d.c:46: note: not vectorized: loop contains function calls or data references that cannot be analyzed
heat2d.c:46: note: bad data references.
heat2d.c:43: note: vectorized 0 loops in function.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
heat2d.c:46: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
```



Caso de Aplicación: Mandelbrot

- **Conjunto de Mandelbrot**: secuencia de operaciones sobre números complejos que no tienen a infinito, se grafican como fractales.
- **Conclusiones**: la estructura esta alineada y compactada, el caso de prueba es estable, la proporción de trabajo en paralelo es irreal, cerca del 50% del tiempo de ejecución se invierte en una sola línea, los ciclos ya están vectorizados utilizando `movss`/`addss`.

```
Flat profile:
Each sample counts as 0.01 seconds.
     cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 44.60     17.09    17.09                             main._omp_fn.0 (mandel.c:45 @ 400afd)
 16.52     23.42     6.33                             main._omp_fn.0 (mandel.c:48 @ 400b0f)
  9.76     27.16     3.74                             main._omp_fn.0 (mandel.c:43 @ 400aed)
```

```
      :          do
      :            {
      :              temp = z.real * z.real - z.imag * z.imag + c.real;
      :              z.imag = 2.0 * z.real * z.imag + c.imag;
 0.57 :   400999:       addsd  xmm6,xmm0
 6.56 :   40099d:       unpcklpd xmm0,xmm0
      :              z.real = temp;
      :              lensq = z.real * z.real + z.imag * z.imag;
 5.17 :   4009a5:       movaps xmm1,xmm0
 2.26 :   4009a8:       mulss  xmm1,xmm0
 3.94 :   4009ac:       movaps xmm2,xmm3
 0.13 :   4009af:       mulss  xmm2,xmm3
19.29 :   4009b3:       addss  xmm3,xmm0
```

Conclusiones

- La **optimización** es un trabajo no trivial; requiere disciplina, conocimiento y experiencia.
- Se puede **simplificar** el trabajo con una metodología de referencia basada en herramientas.
- Se propone un **procedimiento** gradual e iterativo con soporte automático.
- Se desarrollo una **infraestructura** para que un especialista en el dominio pueda obtener información cuantitativa fácilmente.
- Se revisaron casos de estudio como **aplicación** de la infraestructura.

Trabajo Futuro

- **Extensión**: nuevas o mejoradas secciones en el reporte, contadores de *hardware* específicos.
- **Aplicación**: utilización de la infraestructura por algún grupo de investigación y/o desarrollo.
- **Soporte MPI**: incorporar soporte para programas basados en MPI.
- **Reportes Dinámicos**: implementación en HTML5 o compatible con formato de planilla de cálculo utilizando gráficos configurables o tablas filtrables.