Recolección de Basura en D

Leandro Lucarella

Facultad de Ingeniería, UBA

Diciembre 2010

Motivación

- Recolección de basura
- Lenguaje de programación D
- Investigación + aplicación
- Software Libre

Recolección de Basura

¿Qué?

- Administración automática de memoria

¿Para qué?

- Simplificar interfaces
- Evitar errores de memoria
- Mejorar eficiencia (!)

¿Cómo?

- Análisis del grafo de conectividad del *heap*
- 50+ años de desarrollo
- 3000+ *papers*

Recolector Actual de D

- Marcado y barrido
  - Marcado iterativo
- Conservativo
  - Con una pizca de *precisión* (`NO_SCAN`)
- *Stop-the-world*
  - Durante el marcado (en teoría)
- *Lock* global
  - Muy propenso a extender el tiempo de *stop-the-world* en la práctica

Recolector Actual - Lo Bueno

- Anda :)
- Organización del *heap* (< fragmentación)
- Marcado iterativo (!*overflow*)
- *Bitset* para bits de marca (*cache friendly*)

(bueno != perfecto)

Recolector Actual - Lo Malo y lo Feo

Lo malo

- $\downarrow$ Configurabilidad (*no silver bullet*)
- $\downarrow$ Precisión (información de tipos) $\rightarrow$ Memoria inmortal
- $\downarrow$ Concurrencia $\rightarrow$ Grandes pausas
- $\downarrow$ Control sobre el factor de ocupación del *heap*
  $\rightarrow$ Casos patológicos

Lo feo

- El código (complejo, intrincado, duplicado, poco documentado)
  $\rightarrow$ Difícil de mantener, modificar y mejorar

Concurrencia

- Algoritmo basado en el trabajo de Gustavo Rodriguez-Rivera y Vince Russo (*Non-intrusive Cloning Garbage Collector with Stock Operating System Support*)
- Minimiza tiempo de pausa realizando fase de marcado concurrente vía `fork(2)`
- Proceso padre sigue corriendo el programa
- Proceso hijo realiza fase de marcado
- Se comunican resultados vía memoria compartida
- Sincronización mínima (`fork(2)` + `waitpid(2)`)

Concurrencia - Problemas

- Hilo que disparó la recolección bloqueado hasta fin de recolección completa (marcado concurrente inclusive)
- Otros hilos potencialmente bloqueados durante toda la recolección también (*lock* global)

$\rightarrow$ Tiempo de pausa en la práctica ~= tiempo total de recolección

Concurrencia - Eager Allocation

- Pide más memoria al OS antes de lanzar el marcado concurrente
- Devuelve memoria nueva al programa mientras termina el marcado concurrente
- Permite al programa (todos sus hilos) seguir trabajando mientras se realiza el marcado concurrente
- Compromiso
  ↑ Consumo de memoria
  ↓ Tiempo de pausa real

Concurrencia - Early Collection

- Dispara una recolección *preventiva* antes de que se agote la memoria
- Permite al programa (todos sus hilos) seguir trabajando mientras la recolección *preventiva* está en progreso
- Si se agota la memoria antes de que la recolección *preventiva* finalice, se vuelve a bloquear
- Combinable con *eager allocation* para evitar bloquear
- Pueden realizarse más recolecciones de las necesarias
- Compromiso
  ↑ Consumo de procesador (potencialmente)
  ↓ Tiempo de pausa real (no garantizado)

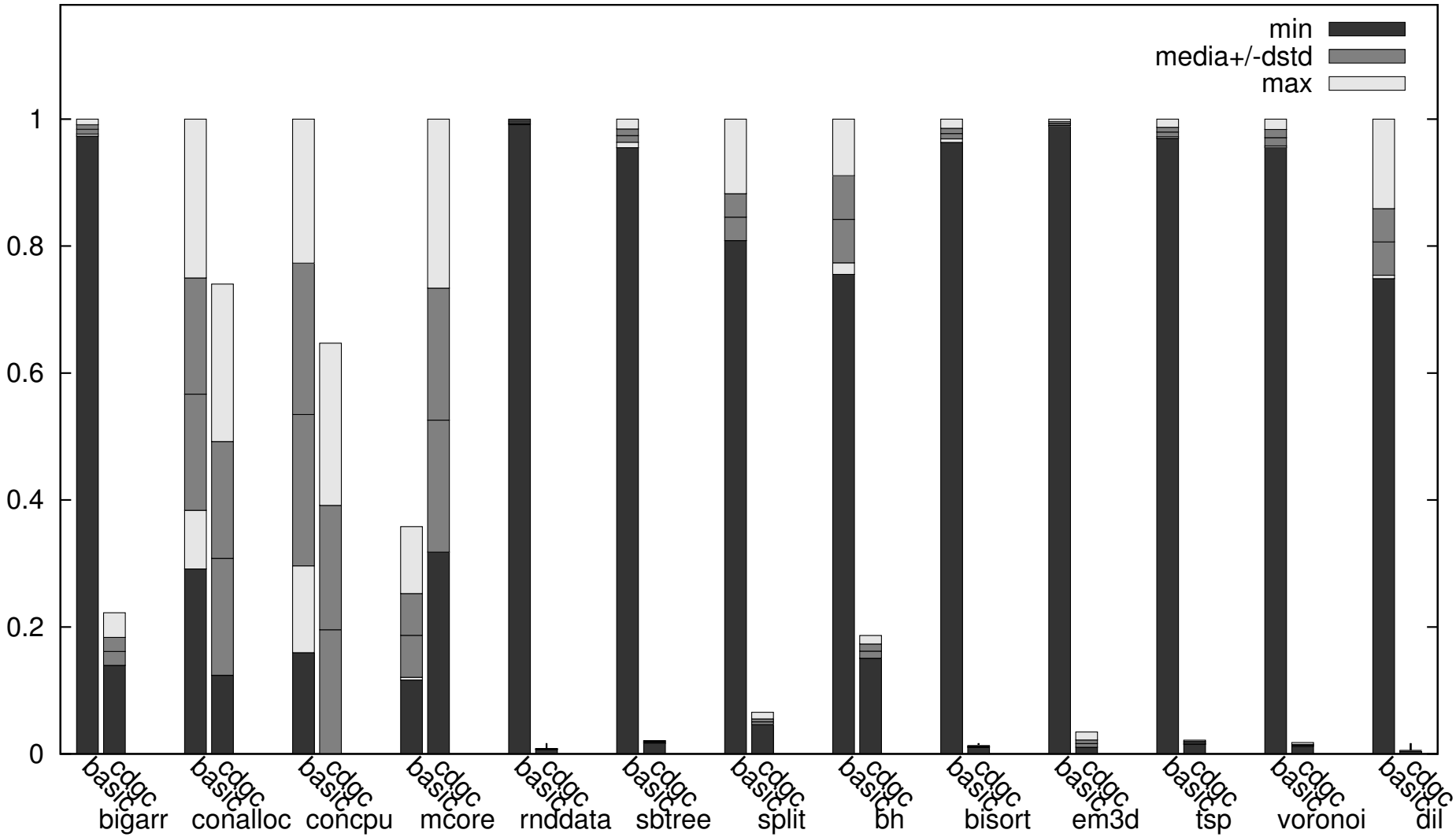
Otras Mejoras

- Marcado semi-preciso del *heap*
- Mejora del factor de ocupación del *heap*
- Caché de consultas críticas para acelerar cuellos de botella
- Reestructuración, modularización, simplificación y limpieza del código
- Pre-asignación de memoria
- Optimizaciones algorítmicas sobre búsquedas frecuentes
- Registro de pedidos de memoria y recolecciones realizadas
- Configurabilidad (en *tiempo de inicialización*)

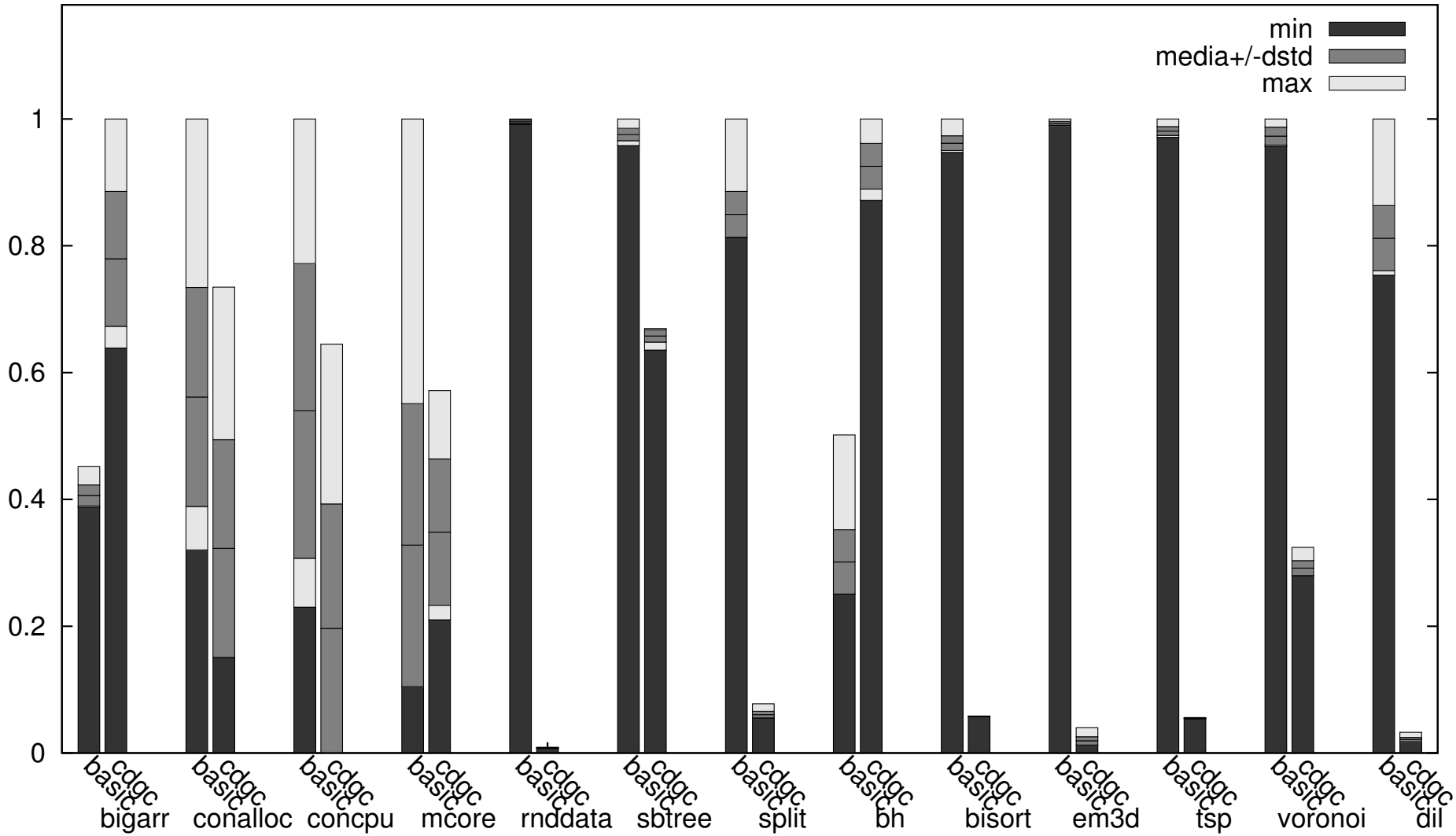
Banco de Pruebas

- Programas
  - 7 *Micro-benchmarks*
  - 5 *Olden Benchmark* (400-1000 *SLOC*)
  - Dil (32K+ *SLOC*, 86 módulos, 300+ *clases*)
- Métricas
  - Tiempo total de ejecución
  - Tiempo máximo de *stop-the-world*
  - Tiempo máximo de pausa real
  - Cantidad máxima de memoria utilizada

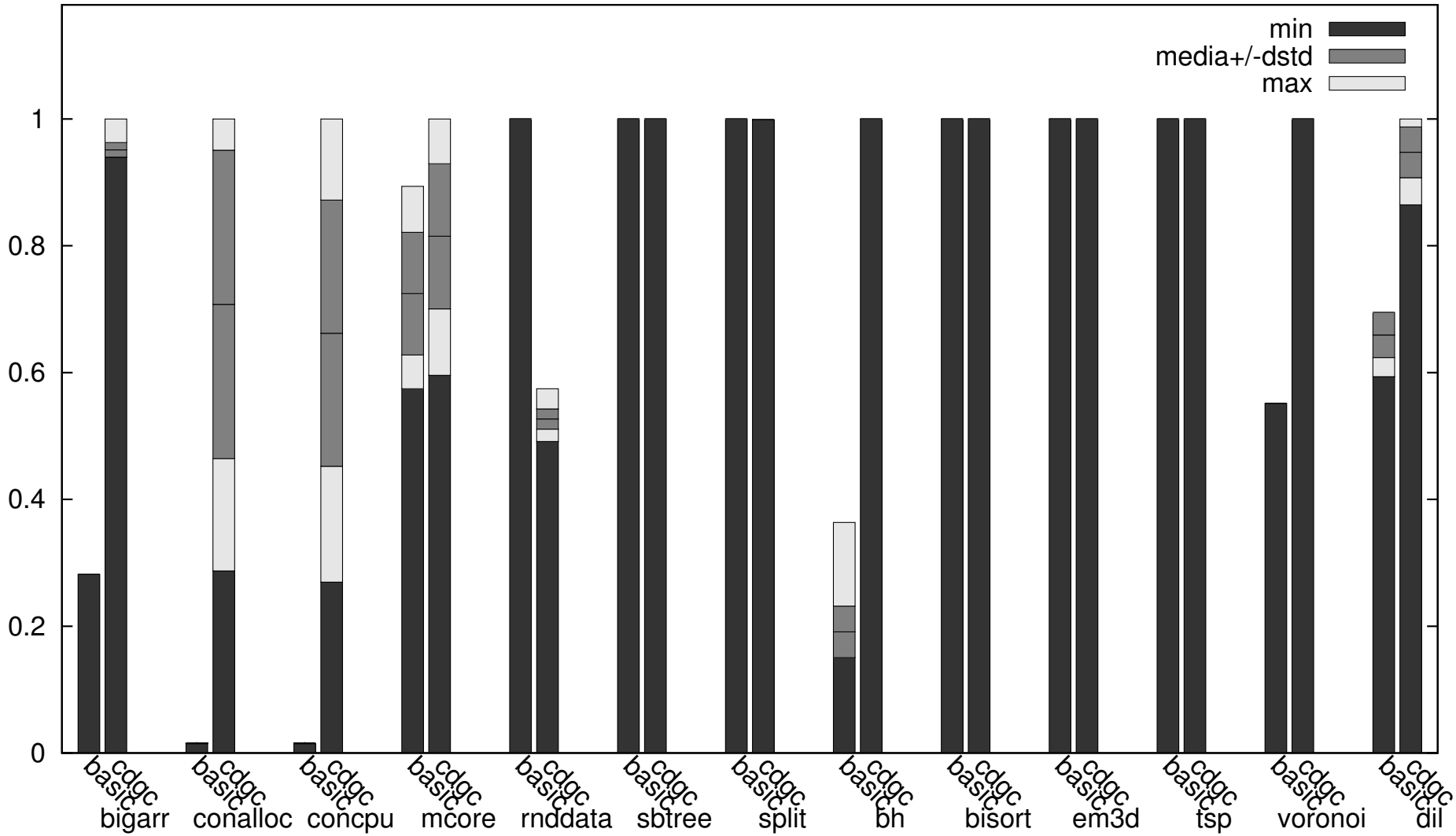
Tiempo Máximo de Stop-The-World



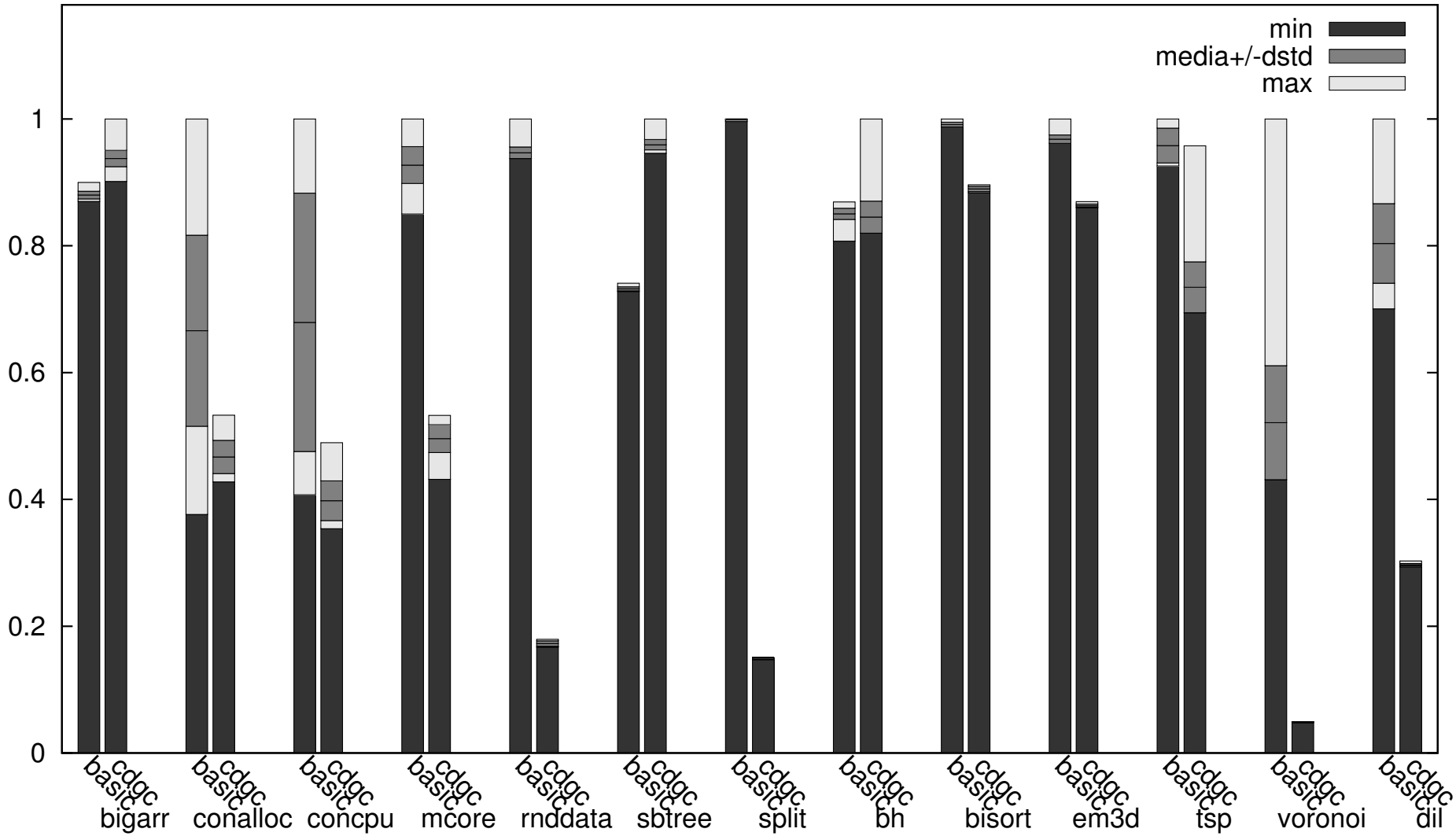
Tiempo Máximo de Pausa Real



Cantidad Máxima de Memoria Utilizada



Tiempo Total de Ejecución



Resumen

- Objetivo principal

  Minimizar tiempo de pausa para programas reales
  Tiempo de pausa de Dil:
  - *Stop-the-world* 160 veces menor (1.66s → 0.01s)
  - Pausa real 40 veces menor (1.7s → 0.045s)
- Objetivo secundario

  No empeorar mucho el recolector actual en ningún aspecto
  Utilización de memoria de Dil:
  50 % mayor (213MiB → 307MiB)
  (mucho *overhead* por marcado preciso)
- Yapa

  Tiempo total de ejecución de Dil:
  Casi 3 veces menor (55s → 20s)

Problemas, Limitaciones y Puntos Pendientes

- Explosión de uso de memoria con *eager allocation*
- Eficiencia del marcado preciso
- Mejorar predicción de *early collection*
- Experimentar con `clone(2)`

Trabajos Relacionados

- *Memory Management in the D Programming Language*
  Vladimir Panteleev. Proyecto de licenciatura, Universitatea Tehnică a Moldovei, 2009.
- *Integrate Precise Heap Scanning Into the GC*
  David Simcha (GC + diseño) y Vincent Lang (compilador). No formal, *bug report*, 2009-2010.
- *Non-intrusive Cloning Garbage Collection with Stock Operating System Support*
  Gustavo Rodriguez-Rivera y Vince Russo. Software Practiceand Experience Volumen 27, Número 8. Agosto 1997.

Trabajos Futuros

- Organización de memoria
- Barrido
- + Precisión
- Concurrencia → *Lock* global
- Movimiento

Preguntas

¿?

Fin

¡Gracias!