Análisis detallado de las diferencias clave en los ensambladores (compilados con -O0):

1. Operaciones en el bucle crítico
sin_opt.s (Código no optimizado):
Multiplicación y división explícitas:

asm
```
imull -212(%rbp), %eax   ; a = i * 8 (m3=8)
idivl -208(%rbp)         ; b += a / 32 (m5=32)
```
Corresponde a:

c
```
a = i * m3;
b += a / m5;
```
Costo alto: idivl es una operación lenta (~20-80 ciclos en CPUs modernas).

opt_old.s (Código "optimizado" manualmente):
Desplazamientos de bits:

asm
```
sarl $2, %eax            ; i >> 2 (equivalente a dividir por 4)
sall $3, %eax            ; (N-1) << 3 (equivalente a multiplicar por 8)
```
Corresponde a:

c
```
b += i >> 2;
a = (N[index] - 1) << 3;
```
Teóricamente más rápido: Los desplazamientos (sarl, sall) suelen ser de 1 ciclo.

Problema: En -O0, el compilador no elimina accesos redundantes a memoria, anulando la ventaja.

2. Accesos a memoria en -O0
En ambos códigos, todas las variables se almacenan en la pila, generando accesos innecesarios:

Ejemplo en opt_old.s:

asm
```
movl -224(%rbp), %eax    ; Carga i desde la pila
sarl $2, %eax            ; i >> 2
addl %eax, -216(%rbp)    ; b += resultado (almacena en pila)
```
3 accesos a memoria por iteración (carga i, operación, guarda b).

Ejemplo en sin_opt.s:

asm
```
movl -232(%rbp), %eax    ; Carga i
imull -212(%rbp), %eax   ; Multiplica por m3 (8)
cltd                     ; Prepara división
idivl -208(%rbp)         ; Divide por m5 (32)
addl %eax, -224(%rbp)    ; b += resultado
```
4 accesos a memoria + operación lenta (idivl).

Conclusión: El costo de los accesos a memoria en -O0 eclipsa cualquier ganancia teórica de los desplazamientos.

3. Dependencias de datos en opt_old.s
Bucle interno en opt_old.s:

```
asm
.L11:
  movl -224(%rbp), %eax  ; i
  sarl $2, %eax          ; i >> 2
  addl %eax, -216(%rbp)  ; b += ...
  addl $1, -224(%rbp)    ; i++
```

Dependencia estricta: Cada iteración depende del valor anterior de b (-216(%rbp)), lo que impide la ejecución paralela (pipeline stall).

En sin_opt.s, aunque hay división, no hay dependencia tan crítica:

```
asm
.L11:
  imull -212(%rbp), %eax  ; a = i * 8
  cltd
  idivl -208(%rbp)        ; a / 32
  addl %eax, -224(%rbp)   ; b += ...
```

El idivl es lento, pero el resultado de a no depende de b, permitiendo cierto solapamiento.

4. Overhead fuera del bucle crítico en opt_old.s
Línea adicional en opt_old.s:

```
asm
movl -212(%rbp), %eax    ; index
subl $1, %eax            ; N[index] - 1
sall $3, %eax            ; << 3 (multiplica por 8)
movl %eax, -204(%rbp)    ; Almacena en a
```

Fuera del bucle interno, pero en -O0, esto genera:

2 accesos a memoria (-212(%rbp) y -204(%rbp)).

Operaciones innecesarias si a no se usa luego.

5. Consecuencias en los tiempos de ejecución
sin_opt.s parece más rápido debido a:

Menos dependencias de datos en el bucle crítico.

Aunque usa idivl, el compilador no genera código adicional para operaciones fuera del bucle.

opt_old.s sufre por:

Accesos a memoria redundantes.

Dependencias estrictas en el bucle interno.

Instrucciones adicionales fuera del bucle.

Conclusión final
Las "optimizaciones manuales" en opt_old.s no mejoran el rendimiento con -O0 debido a:

Accesos constantes a memoria (no se usan registros).

Dependencias de datos que limitan el paralelismo.

Overhead de operaciones fuera del bucle crítico.

El costo de gettimeofday enmascara diferencias mínimas.

Recomendación: Para ver mejoras, compilar con -O1/-O2, donde el compilador:

- Usará registros.
- Eliminará accesos redundantes.
- Reordenará instrucciones para mitigar dependencias.