Multiplicación de matrices

Tonny Gonzales Villafuerte

March 2017

1 The simple three-nested-loop version of the matrix product

Normalmente, un algoritmo que se refiere a elementos individuales es reemplazado por uno que opera en sub-arrays de datos, que se llaman bloques en el campo de cálculo de matriz. Las operaciones sobre sub-arrays se pueden expresar de la forma habitual. La ventaja de este enfoque es que los pequeños bloques se pueden mover a la memoria local rápida y sus elementos pueden ser utilizados repetidamente.

```
for(i=0; i<m; i++)
    for(j=0; j<n; j++)
    {
           z[i][j] = 0;
           for(k=0; k<c; k++)
            z[i][j] += x[i][k] * y[k][j];
    }
```

1.1 Evaluación

- Con matrices 'x[3][6] y y[6][17]' el algoritmo se demora : 0.107000 ms.
- Con matrices 'x[33][36] y y[36][27]' el algoritmo se demora : 1.461000 ms.
- Con matrices 'x[83][96] y y[96][77]' el algoritmo se demora : 8.600000 ms.

1.2 Valgrind

Al ejecutar 'valgrind –leak-check=full -v ./a.out ' o 'valgrind –tool=callgrind ./foobar', con matrices 'x[83][96] y y[96][77]' nos da el siguiente resultado.

```
--4895-- REDIR: 0x40c8c60 (libc.so.6:free) redirected to 0x402b370 (free)
==4895==
==4895== HEAP SUMMARY:
==4895==     in use at exit: 0 bytes in 0 blocks
==4895==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4895==
==4895== All heap blocks were freed -- no leaks are possible
==4895==
==4895== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4895== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

1.3 Kcachegrind

Al ejecutar 'kcachegrind callgrind.out.XYZ', con matrices 'x[83][96] y y[96][77]' nos da el siguiente resultado. Figura 1.

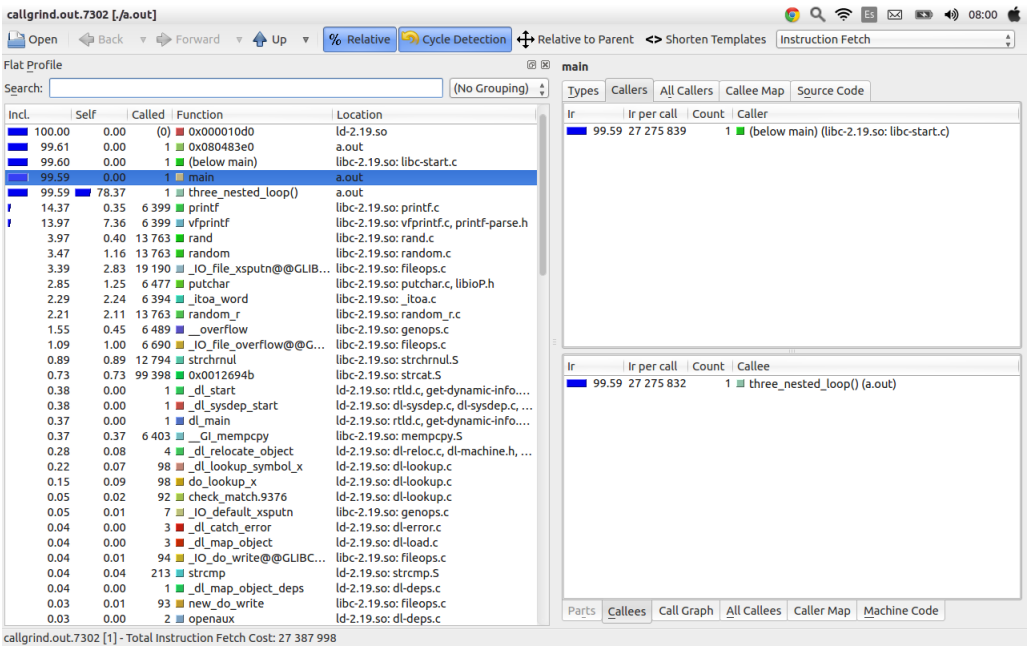

Figura 1. Kcachegrind

1.4 Conclusión

El cálculo completo $2n^3$ operaciones aritméticas (contando adiciones y multiplicaciones por separado), pero produce y consume sólo $3n^2$ valores de datos. En conjunto, el cómputo exhibe admirable reutilización de datos. En general, sin embargo, una matriz entera no cabrá en una memoria local pequeña. El

trabajo debe por lo tanto ser dividido en pequeños trozos de cálculo, cada uno de los cuales utiliza una pieza suficientemente pequeña de los datos. Obsérvese que para cada iteración del bucle exterior (es decir, para un valor dado de i) se realizan operaciones n^2 y se hace referencia a los datos n^2, sin reutilización. Para los valores fijos de i y j, n de cálculo y n datos referidos también - de nuevo, no hay reutilización.

2 The blocked version with six nested loops

```
for( i1=0;i1<(nn/BlockSize);++i1)
{
    for(j1=0;j1<(nn/BlockSize);++j1)
    {
        for(k1=0;k1<(nn/BlockSize);++k1)
        {
            for(i=i1;i<min(i1+BlockSize-1,nn);++i)
            {
                for(j=j1;j<min(j1+BlockSize-1,nn);++j)
                {
                    for(k=k1;k<min(k1+BlockSize-1,nn);++k)
                    {
                     cc[i][j] = cc[i][j] + a[i][k] * b[k][j];

                    }
                }
            }
        }
    }
}
```

2.1 Evaluación

- Con matrices 'x[27][27] y y[27][27]' el algoritmo se demora : 3.569000 ms.
- Con matrices 'x[77][77] y y[77][77]' el algoritmo se demora : 7.282000 ms.
- Con matrices 'x[177][177] y y[177][177]' el algoritmo se demora : 72.445999 ms.

2.2 Valgrind

Al ejecutar 'valgrind –leak-check=full -v ./a.out ' o 'valgrind –tool=callgrind ./foobar', con matrices 'x[83][96] y y[96][77]' nos da el siguiente resultado.

```
==7765== Callgrind, a call-graph generating cache profiler
==7765== Copyright (C) 2002-2013, and GNU GPL'd, by Josef Weidendorfer et al.
==7765== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==7765== Command: ./a.out
==7765==
==7765== For interactive control, run 'callgrind_control -h'.
33 36 27

15.421000
==7765==
==7765== Events    : Ir
==7765== Collected : 1105761
==7765==
==7765== I   refs:      1,105,761
```

2.3 Kcachegrind

Al ejecutar 'kcachegrind callgrind.out.XYZ', con matrices 'x[83][96] y y[96][77]' nos da el siguiente resultado. Figura 2.

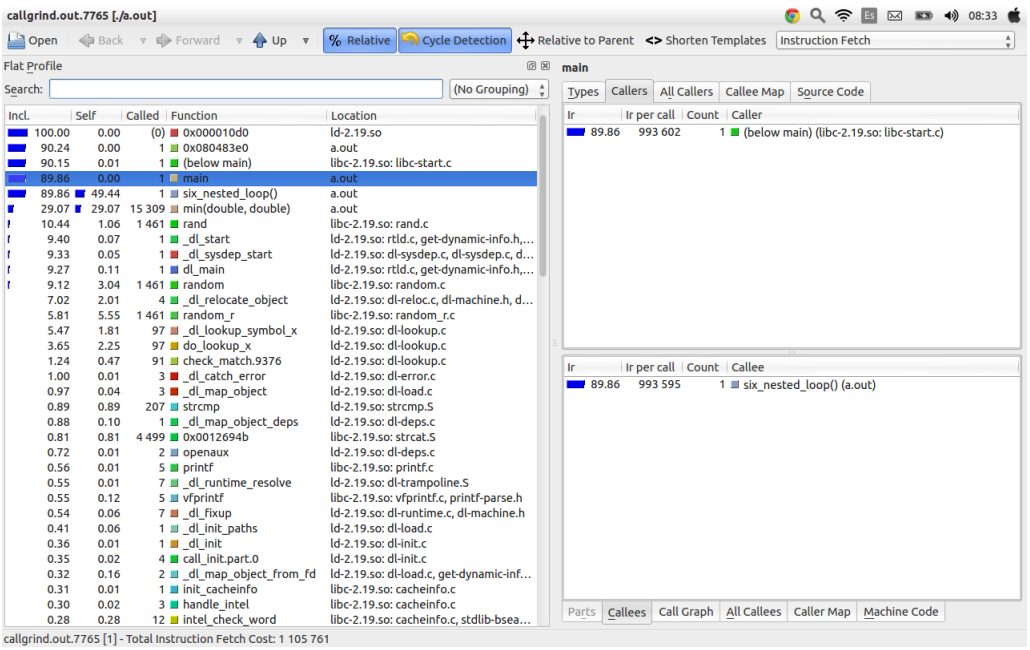

Figura 2. Kcachegrind

2.4 Conclusión

En primer lugar, tenga en cuenta que en este programa se realizan exactamente las mismas operaciones en los mismos datos; Incluso el error de redondeo es idéntico. Sólo la secuencia en la que se realizan operaciones distintas es diferente del programa desbloqueado. Todavía hay reutilización en todo el programa de orden n. Pero si consideramos una iteración con i0 fijo, j0 y k0, vemos que se realizan operaciones de $2b^3$ (por los tres bucles internos) y se informan los datos de $3b^2$. Ahora podemos elegir b lo suficientemente pequeño para que estos datos $3b^2$ encajen en la memoria local y así lograr la reutilización b-fold. (Si esto no es suficiente - si b ¡B en otras palabras - entonces la máquina está mal diseñada y necesita más memoria local.) Dicho de otra manera, si necesitamos la reutilización B-fold, elegimos el tamaño del bloque b = B.