Metodi diretti per matrici sparse

Antonio Vivace, Febbraio 2019

Ambienti

Windows
Microsoft Windows 8.1 Professional, 64 Bit

Linux
Debian GNU/Linux buster/sid, 64 Bit
Kernel 4.19.0-2

Versioni dei software
MATLAB R2018b
Python 3.7 64 Bit, stesse versioni di numpy, scipy (requirements.txt lock, python virtualenv)

Caratteristiche macchina
8GB RAM DDR3
i5-4690K 3.50GHz
Swap su SSD (300 MB/s W/R)

Metriche rilevate

Tempo

fullTime: esecuzione completa, comprende l'avvio del programma
loadTime: tempo di caricamento della matrice
solveTime: tempo di soluzione della matrice

Rilevamento:
Python: time.time() deltas
MATLAB: tock()

Memoria

peakMem: picco massimo di memoria raggiunto
baseMem: memoria fino al caricamento della matrice
solveMem: max - peak

Rilevamento:
Python: `psutils.memory_info` (RSS, peak working set).
MATLAB: `memory()` su Windows, `/proc/<pid>/statm` su Linux. Profiler di MATLAB (per solve). [...]

Panoramica

- Esecuzione in batch tramite script bash
- Script Python per l'integrazione dei dati sulle matrici sui risultati finali
    - Size = rows * columns
    - patternEntries =explicit entries on the matrix, including the explicit zero entries
- GNU tool "time" per peakMem su Linux (più affidabile)
- Script R per generare i grafici (ggplot, trasformazione log2 per le ordinate, export con ggsave in SVG, PNG)
- Err è in scala log2 nel plot err
- Difficile profilare la memoria di MATLAB su Linux (memory non c'è, il profiler restituisce dei valori che sono 10% del reale osservato)
- Riavviare le istanze di MATLAB cambia i risultati di 2-3% (pulizia workspace?).
- MATLAB ha overhead diversi su Linux e Windows (1GB vs 600MB), JVM?

ex19	273K
graham1	343K
kim2	11330K
PR02R	8187K
raefsky3	1500K
water_tank	2039K
ex15	109K
shallow_water1	338K
parabolic_fem	3684K
cfd1	1834K
cfd2	3101K

MATLAB

Lanciato con

```
/usr/bin/time -f '%M,%e' matlab -nojvm -nodesktop
-r "inputMatrix='$x'; solve" | tail -n 2
```

Per avere un'idea dell'overhead temporale e di memoria di MATLAB.

CLI, senza JVM (-100 MB di RAM, circa).

1200 MB di consumo BASE dell'ambiente MATLAB, non incluse dal Profiler.

Metriche registrate con `tic()`, Profiler di MATLAB per rilevare `peakMemory` del comando `solve()`

Su Linux, il tool GNU time registra il picco (PeakMemory) del Resident Set Size del processo.

```
profile -memory on

tic();
load(inputMatrix);
loadTime=toc();

[filepath,name,ext] = fileparts(inputMatrix);

A = Problem.A;
n=size(A,1);
xe=ones(n,1);

tic();
b=A*xe;
x = A\b;
[user,sys] = memory();
mem = user.MemUsedMATLAB;

solveTime=toc();
err = norm(x-xe)/norm(xe);


profile off
a = profile('info');
sep = ',';
results = [name sep num2str(err) sep num2str(solveTime) sep
num2str(a.FunctionTable(3).PeakMem)];
disp(results)
```

Python - SciPy

Lanciato con

```
for x in $DIR/*.mat; do
      /usr/bin/time -f '%M,%e' python3 solve.py $x
done
```

Su Windows, l'aumento di memoria provocato da solve è misurato rilevando baseMem e solveMem all'interno dello script. `process.memory_info().peak_wset`,

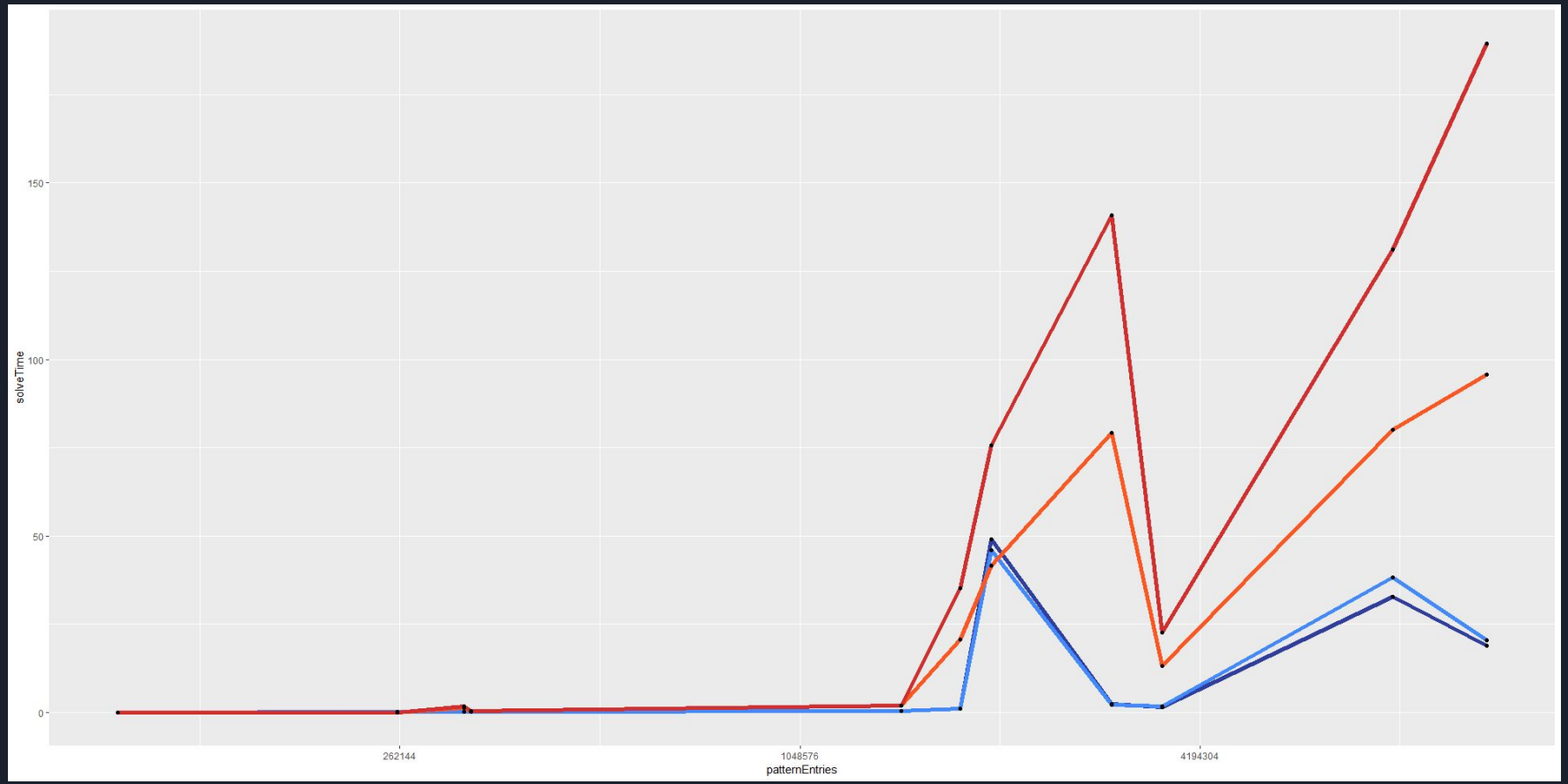
Su Linux, peakMem è rilevato con il tool GNU time (%M), mentre baseMem sempre con `process.memory_info().rss` (~ `/proc/<pid>/statm`). Peak working set non è disponibile su Linux.

deltaMem = peakMem - baseMem

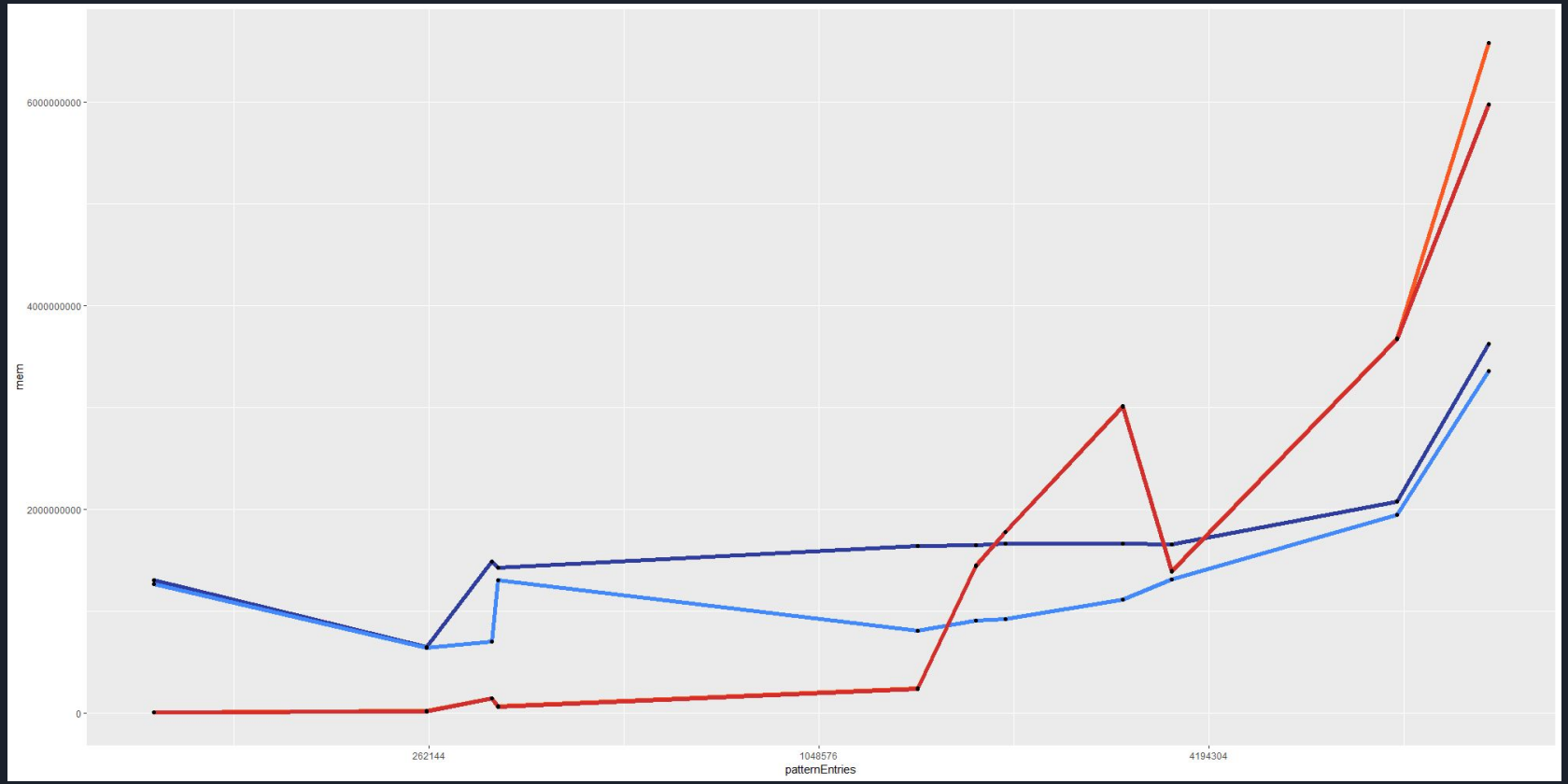
baseMem è acquisito subito dopo aver caricato la matrice.

```
[...]

process = psutil.Process(os.getpid())

x = 0
A = []
b = []


mat_file = sys.argv[1]
mat = loadmat(mat_file)
basemem= process.memory_info().rss

A = mat['Problem']['A'][0][0]
N = A.shape[0]
b = A * ones(N)
tick = time.time()
x = spsolve(A, b, use_umfpack=True)
tock = time.time()
error = norm(x - ones(N))/norm(ones(N))


#peakmem= process.memory_info().peak_wset WINDOWS ONLY
print(sys.argv[1].split("/")[-1][:-4], ",", error, ",", tock -
tick, ",",basemem)

[...]
```

Tempi di esecuzione

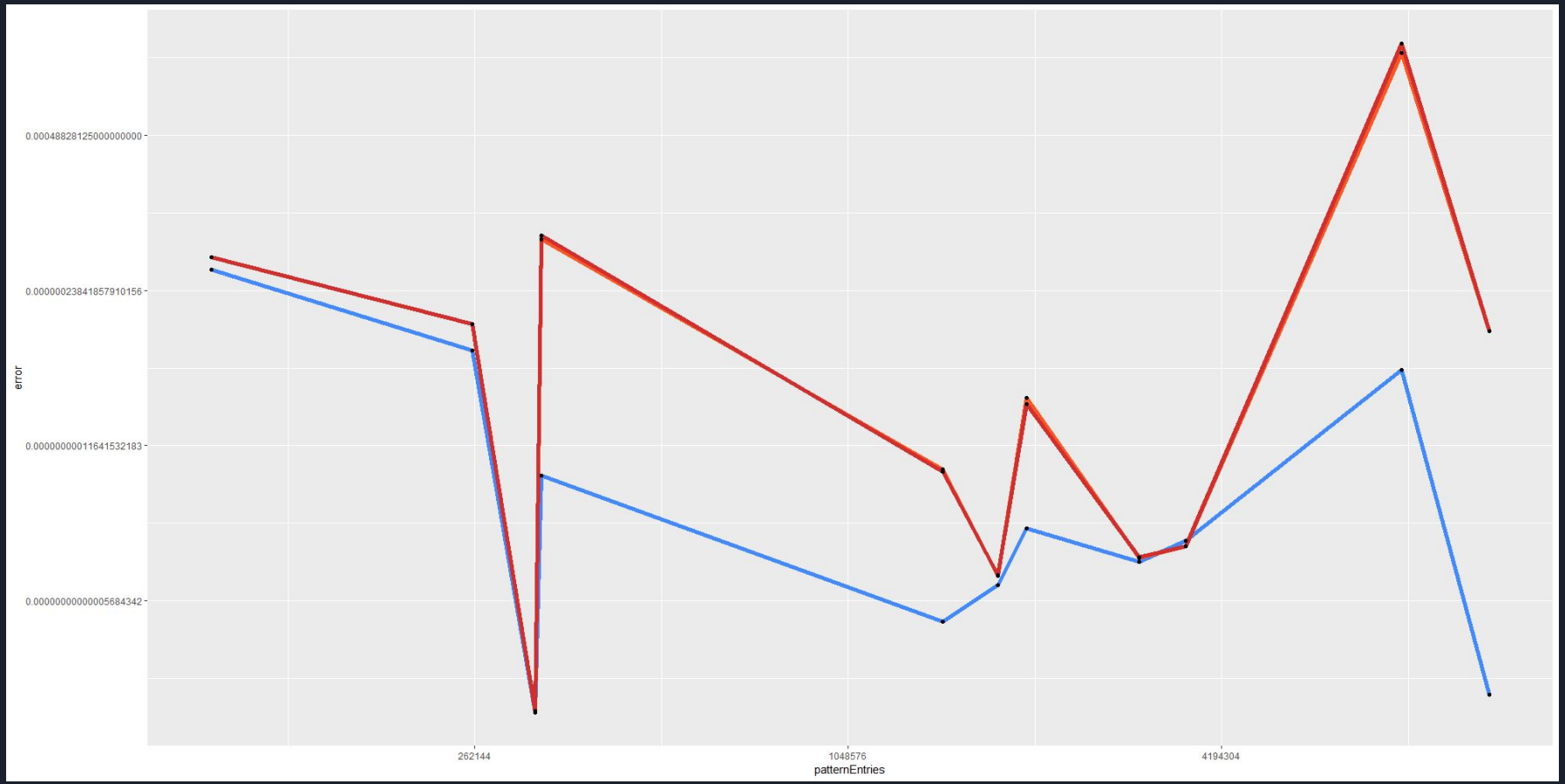


Memoria



valori maggiori di 4/6 GB indicano swap

Errori


Python on Linux
Python on Windows
MATLAB on Linux
MATLAB on Windows
0.0004882812500000000
0.0000023841857910156
0.0000000011641532183
0.0000000000005684342
error
262144
1048576
4194304
patternEntries

Conclusioni

Windows:

- A volte più stabile, gestione dello Swap a volte migliore
- Limitazione multithread
- Computazioni pesanti non intaccano la stabilità del sistema

Linux:

- Più facile la gestione dei risultati, bash pipes, ecc ecc. Scripting e manipolazione risultati.
- Python utilizza correttamente multithreading, spawnando 4 thread per un utilizzo massimo della CPU
- Su matrici che eccedono la disponibilità di RAM per la computazione non viene fatto correttamente SWAP (matrici apache2, torso3)
- L'utilizzo completo della CPU comporta anche il freeze/crash dei processi per il desktop, ecc,ecc. Ovviabile utilizzando un tty
- GPU Computing!
- Mancano molti QoL improvements che ci sono su Windows. (Ubuntu stable? altre distro)

MATLAB:

- memory() non funziona su Linux
- Molto più performante sulle matrici definite positive
- Overhead di 1200-1300 MB di memoria, costante (solo per aprire l'ambiente)
- Gestisce meglio lo swap

Python SciPy:

- cannot expand memtype, Issue noto di SciPy su linux, memory fragmentation (SuperLU?)
- Open Source, mantenuto
- Esistono alternative che girano su CUDA/parallellizzazioni molto più performanti

SciPy code frequency