Profiling and Code Optimizations

Dimitris Dellis

GRNET

Athens, 20 - 22 Nov. 2018

Outline

- Profilers
- From Theory (i.e. equations) to serial code, to efficient parallel code.
- Examples of profiling in real applications
- Hands On on supplied codes or your own code
- Discussion

- Profiler is software that gets metrics on source execution, without addition of timers in source code.
- Serial Profilers
  - One can find detailed time spent in code procedures, i.e. How many times a procedure was called, average time per call, total time spent in procedure, from which point in source was called etc.
  - Standard Unix profiler gprof and its variants, for example sprof.
  - Graphical Interface to gdb ddd.
  - Compiler specific profilers, like vtune for Intel compilers or pgprof for PGI.

- MPI
  - It is possible to use gdb for parallel applications ?
    - Yes
    - mpirun -np N (or mpiexec.hydra -n N) xterm -c gdb exe
    - You may need to press many run commands
  - mpiP : Traces MPI calls and gives performance indicators, possible bottlenecks etc. OpenSource, Works with any compiler and MPI implementation.
    - In most cases you can find in few runs your application MPI problems.
      More details during Hands-On.
  - MPI implementations profilers, for example OpeMPI VampirTrace.

- Hybrid MPI/OpenMP/Threads Profilers
  - scalasca : Traces MPI calls, as well as OpenMP calls, provides detailed information timing information per thread, task, node, code line. Graphical Interface to explore profile information.
  - Other mainly commercial profilers/debuggers, for example DDT

In Practice

- Serial Applications : gprof
  - At Compile time use the flags : -pg
  - It is suggested to use -O0 for optimization to avoid any inlining that may result to missing functions timing.
  - Example : `00_profiling1.f` : Matrix Matrix Maltiplication.

```
module load binutils
gcc 00_profiling1.f -pg -O0 -o 00_profiling1.x
./00_profiling1.x
gprof 00_profiling1.x
```

  - You'll se something like

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
100.30	10.82	10.82	1	10.82	10.82	mymm_
0.09	10.83	0.01	1	0.01	0.01	initializearrays_
0.00	10.83	0.00	1	0.00	10.83	MAIN__

- In Brief :
  - mymm is executed 1 times, need 10.82 seconds for each call, it is the main time consuming procedure.
  - initializearrays is executed 1 times, need 0.01 secs per call.
  - Main is executed 1 times, it needs less than 0.005 seconds to complete.
  - We have a good estimation where the execution time is spent. In real serial applications output is more interesting.

In Practice

- Pure MPI Applications : mpiP
- If you compile your application using : mpif90 mycode.f -o mycode.x
  do

```
module load mpiP
mpif90 mycode.f -g -L$MPIPROOT/lib -lmpiP -lbfd -lunwind -o mycode.x
```

- -g (debug) flag is needed to include source code information in executable.
- If (that is the case) you have a makefile to compile, use in the linking stage mpiP, example :

```
$LD $(OBJECTFILES) -g -L$MPIPROOT/lib -lmpiP -lbfd -lunwind -o mycode.x
```

- Run it : srun mycode.x in slurm

- or mpiexec.hydra -n 8 mycode.x (interactively on login node with 8 procs)
- After completion you'll find a report file called mycode.x.NPROCS.PID.mpiP
- Have a look in the provided information.
- You'll se something like

```
@ mpiP
@ Command : ./06.x
@ Version                  : 3.4.1
@ MPIP Build date          : Sep  7 2015, 16:33:51
@ Start time               : 2017 11 29 21:45:28
@ Stop time                : 2017 11 29 21:45:31
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 29284
@ Final Output Dir         : .
@ Report generation        : Collective
@ MPI Task Assignment      : 0 login01
```

```
......
@--- MPI Time (seconds) --------------------------------------------------
-----------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0       2.72        0.7    25.69
   1       2.72       1.16    42.52
   2       2.72       1.07    39.11
   3       2.72       1.06    38.98
   4       2.72       1.04    38.24
   5       2.72       1.32    48.29
.....
 31       2.72       1.13    41.51
  *       87.1        34.9    40.05
......
@--- Callsites: 11 -------------------------------------------------------
-----------------------------------------------------------------------------
 ID Lev File/Address                    Line Parent_Funct             MPI_Call
  1   0 06_md_inhomegeneous_reduce.f     115 md                       Bcast
  2   0 06_md_inhomegeneous_reduce.f     137 md                       Bcast
  3   0 06_md_inhomegeneous_reduce.f     202 md                       Reduce
.....
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
-----------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      COV
```

```
Reduce                    4   1.27e+04   14.57   36.37    0.94
Barrier                   8   1.03e+04   11.78   29.41    1.09
Bcast                     6    8.8e+03   10.10   25.22    0.18
......
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
--------------------------------------------------------------------------
Call                   Site      Count      Total       Avrg  Sent%
Reduce                    3         32    3.2e+07      1e+06  11.11
Reduce                    4         32    3.2e+07      1e+06  11.11
Reduce                    9         32    3.2e+07      1e+06  11.11
Bcast                    11         32    3.2e+07      1e+06  11.11
......
@--- Callsite Time statistics (all, milliseconds): 352 --------------------
--------------------------------------------------------------------------
Name                 Site Rank  Count      Max     Mean      Min   App%   MPI%
Barrier                 8    0      1    0.048    0.048    0.048   0.00   0.01
Barrier                 8    1      1      774      774      774  28.42  66.83
Barrier                 8    2      1      769      769      769  28.23  72.17
```

....

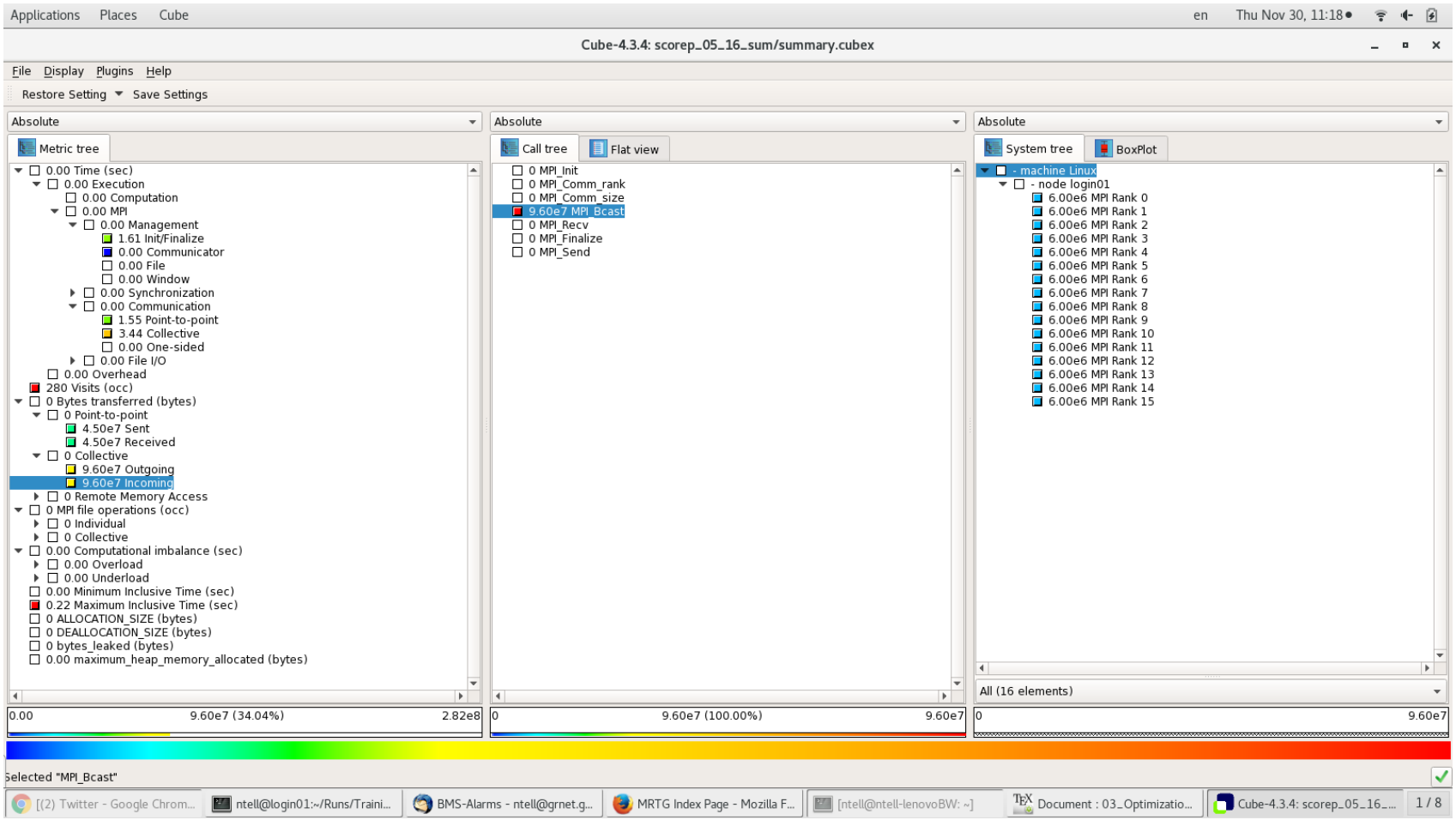
and more interactively.

In Practice

- Hybrid Applications : Scalasca
- If you compile your application using : mpif90 mycode.f -o mycode.x
  do

```
module load binutils qt/5.6.0 cuda/7.5.18
scalasca -instrument mpif90 mycode.f -o mycode.x
scalasca -analyze mpiexec.hydra -n 8 ./mycode.x
scalasca -examine scorep_mycode.x_8_sum
```

- You'll see something like (You need X11 at your Desktop)
- https://sourceforge.net/projects/xming/
- If not X11 available, instead of scalasca -examine use : square -s `scorep_mycode.x_8_sum`. A report will be in `scorep_mycode.x_8_sum/scorep.score` text file.


Applications Places Cube
en Thu Nov 30, 11:18
Cube-4.3.4: scorep_05_16_sum/summary.cubex
File Display Plugins Help
Restore Setting Save Settings
Absolute
Metric tree
0.00 Time (sec)
0.00 Execution
0.00 Computation
0.00 MPI
0.00 Management
1.61 Init/Finalize
0.00 Communicator
0.00 File
0.00 Window
0.00 Synchronization
0.00 Communication
1.55 Point-to-point
3.44 Collective
0.00 One-sided
0.00 File I/O
0.00 Overhead
280 Visits (occ)
0 Bytes transferred (bytes)
0 Point-to-point
4.50e7 Sent
4.50e7 Received
0 Collective
9.60e7 Outgoing
9.60e7 Incoming
0 Remote Memory Access
0 MPI file operations (occ)
0 Individual
0 Collective
0.00 Computational imbalance (sec)
0.00 Overload
0.00 Underload
0.00 Minimum Inclusive Time (sec)
0.22 Maximum Inclusive Time (sec)
0 ALLOCATION_SIZE (bytes)
0 DEALLOCATION_SIZE (bytes)
0 bytes_leaked (bytes)
0.00 maximum_heap_memory_allocated (bytes)
0.00 9.60e7 (34.04%) 2.82e8
Absolute
Call tree
Flat view
0 MPI_Init
0 MPI_Comm_rank
0 MPI_Comm_size
9.60e7 MPI_Bcast
0 MPI_Recv
0 MPI_Finalize
0 MPI_Send
0 9.60e7 (100.00%) 9.60e7
Absolute
System tree
BoxPlot
- machine Linux
- node login01
6.00e6 MPI Rank 0
6.00e6 MPI Rank 1
6.00e6 MPI Rank 2
6.00e6 MPI Rank 3
6.00e6 MPI Rank 4
6.00e6 MPI Rank 5
6.00e6 MPI Rank 6
6.00e6 MPI Rank 7
6.00e6 MPI Rank 8
6.00e6 MPI Rank 9
6.00e6 MPI Rank 10
6.00e6 MPI Rank 11
6.00e6 MPI Rank 12
6.00e6 MPI Rank 13
6.00e6 MPI Rank 14
6.00e6 MPI Rank 15
All (16 elements)
0 9.60e7
Selected "MPI_Bcast"
[(2) Twitter - Google Chrom...
ntell@login01:~/Runs/Traini...
BMS-Alarms - ntell@grnet.g...
MRTG Index Page - Mozilla F...
[ntell@ntell-lenovoBW: ~]
Document : 03_Optimizatio...
Cube-4.3.4: scorep_05_16_...
1 / 8

From a Book containing equations to parallel program

- One of physics problems subject to parallelization/optimization is the N-Body problem.

- Many Body problem. Conservative interactions.
- Interaction Energy between two particles is function of distance $U(r)$.
- In a system with N particles
- Energy :

$$E_{pot} = \frac{1}{2} \sum_{i=1}^{N} \sum_{j \neq i}^{N} U(r_{ij})$$

- Total Force on particles

$$\vec{F}_i = - \sum_{j=1 \; j \neq i}^{N} \vec{\nabla} U(r_{ij})$$

- Force between particles *i* and *j*.

$$\vec{F}_{ij} = -\vec{F}_{ji} = -\frac{\partial U\left(r_{ij}\right)}{\partial \vec{r}_{ij}} = -\frac{dU(r_{ij})}{dr_{ij}}\frac{\vec{r}_{ij}}{r_{ij}}$$

- Cartesian Components of Force

$$F_{ij}^{x} = -F_{ji}^{x} = -\frac{dU(r_{ij})}{dr_{ij}}\frac{x_{ij}}{r_{ij}}$$

$$F_{ij}^{y} = -F_{ji}^{y} = -\frac{dU(r_{ij})}{dr_{ij}}\frac{y_{ij}}{r_{ij}}$$

$$F_{ij}^{z} = -F_{ji}^{z} = -\frac{dU(r_{ij})}{dr_{ij}}\frac{z_{ij}}{r_{ij}}$$

- Some Interaction functions :
- Gravity

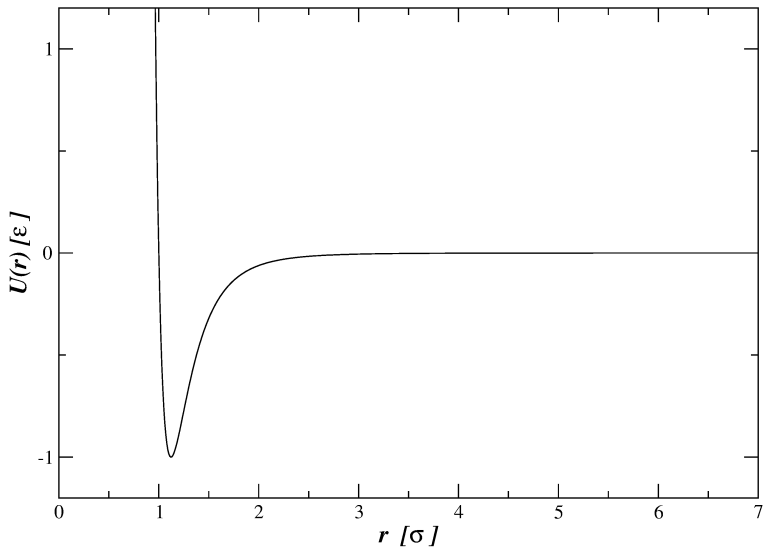
$$U(r) = -K_g \frac{m_i m_j}{r}$$

- Atomic van der Walls, Lennard-Jones type

$$U(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$$

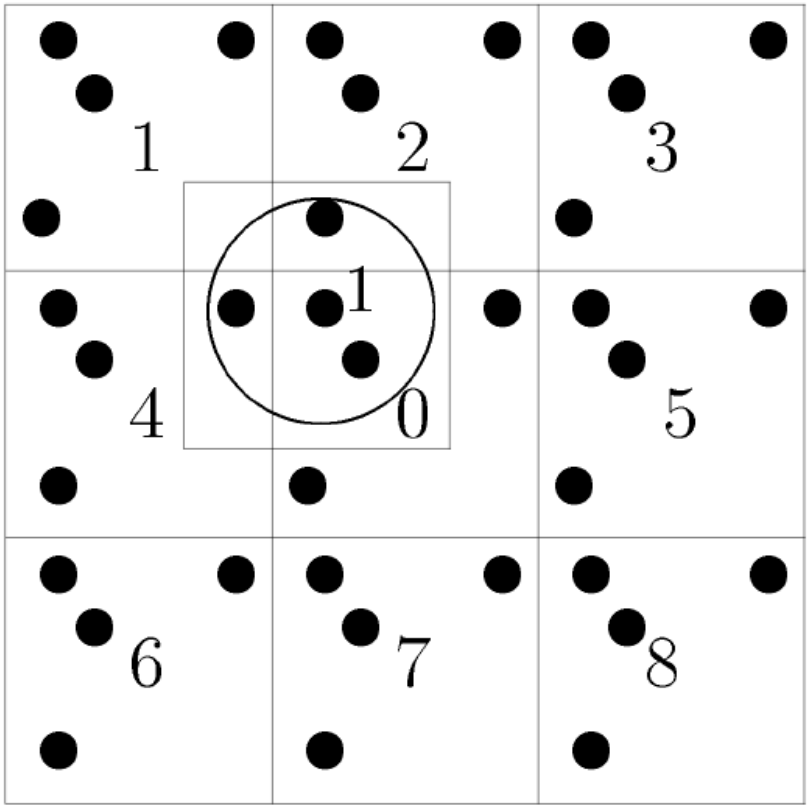
- Force

$$F_{ij}^{x} = -24\epsilon \frac{x_{ij}}{r_{ij}} \left[2 \left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$$



- LJ potential characteristics
- Minimum at : $\sqrt[6]{2}\sigma$
- Depth of Minimum : ϵ
- At distance 6σ, $U(r) = -8.5 \cdot 10^{-5} \times \epsilon$, $F(r) = 8.5 \cdot 10^{-5} \times \frac{\epsilon}{\sigma}$
- We can ignore interactions at distances $> 6\sigma$ = Cut-off distance.

- In order to simulate 18 grams of water one needs $6.0225 \cdot 10^{23} = N_{Avogadro}$ water molecules : Extremely high.
- Technique of Periodic System : One selects a small number of particles (order of few 100s or 1000s, supposes that this sample is representative, assumes that the bulk system (for example 1 Kg of water) is a periodic replica of the small simulation cell.
- This is not true in general, but the bulk properties depend mainly on distances in the reange of interaction potential depth, i.e. few Å.
- In practice, in most cases for simple molecules (i.e. no proteins etc) typically 1000 particles are more than enough.
- Implication : Periodic boundary conditions.



- If the distance component (x, y ,z) between two particles in the cubic simulation cell is more than half the cell edge, we should consider its periodic image in this dimension.

- Start with the code
- One first implementation according to equations

```
c  Timer
         tzero = mytimer()
         do i = 1, N
             do j =  1, N
              if ( i .ne. j ) then
                dx = x(i) - x(j)
                dy = y(i) - y(j)
                dz = z(i) - z(j)
c  Boundary conditions
                if ( dx .lt. -BoxL2 ) then
                     dx = dx + BoxL
                endif
                if ( dy .lt. -BoxL2 ) then
                     dy = dy + BoxL
                endif
                if ( dz .lt. -BoxL2 ) then
                     dz = dz + BoxL
                endif
                if ( dx .gt. BoxL2 ) then
                     dx = dx - BoxL
                endif
                if ( dy .gt. BoxL2 ) then
                     dy = dy - BoxL
                endif
                if ( dz .gt. BoxL2 ) then
                     dz = dz - BoxL
                endif
                dr2 = dx*dx + dy*dy + dz*dz
                dr  = dsqrt(dr2)
```

```
c... Cutoff
                if ( dr.le.cutoff ) then
                  epot = epot + 4.0d0 * epsilon*((sigma/dr)**12 -
     $                                              (sigma/dr)**6)
                  fx(i) = fx(i) - 24.0*epsilon*(dx/dr)*
     $                    (2.0*(sigma/dr)**12 - (sigma/dr)**6)
                  fy(i) = fy(i) - 24.0*epsilon*(dy/dr)*
     $                    (2.0*(sigma/dr)**12 - (sigma/dr)**6)
                  fz(i) = fz(i) - 24.0*epsilon*(dz/dr)*
     $                    (2.0*(sigma/dr)**12 - (sigma/dr)**6)
                endif
              endif
            enddo
          enddo
          epot = epot / 2.0000d0
          tnow = mytimer() - tzero
```

- What improvements can be done ?
  - Replace operations that are repeated with one time calculation
  - `dsqrt` as well as other math functions are expensive. Calculate `dsqrt` ONLY when it is really needed.
  - Question : If cutoff is half of the cube edge, and system is homegeneous, how many not needed `dsqrt` calculations we expect to bypass ?

- Particles In Cubic Box :

$$\propto L^3$$

- Particles inside cut-off sphere :

$$\propto \frac{4}{3}\pi\left(\frac{L}{2}\right)^3$$

- Ratio :

$$Ratio = \frac{\pi}{6} \sim 0.52$$

- Code might look like :

```
Out of loops
        cutsq = cutoff * cutoff
        e4 = 4.0d0 * epsilon
        e24 = 24.0d0 * epsilon
......
               if ( dr2.le.cutsq ) then
                 dr  = dsqrt(dr2)
                 sigr6 = (sigma/dr)**6
                 sigr12 = sigr6 * sigr6
                 sigr126=2.0*sigr12 - sigr6
                 dxdr = dx/dr
                 dydr = dy/dr
                 dzdr = dz/dr
                 epot = epot + e4* (sigr12 - sigr6 )
                 e24sigr126 = e24 * sigr126
                 fx(i) = fx(i) - e24sigr126*dxdr
                 fy(i) = fy(i) - e24sigr126*dydr
                 fz(i) = fz(i) - e24sigr126*dzdr
               endif
```

- We are lucky, compilers can do some of these improvements for us at high optimization levels and probably specifying compilers flags. But not to avoid the not needed `dsqrt`. We have to care about this.
- Any other possible improvement ?

- Accoding to equations in the begining

$$\vec{F}_{ij} = -\vec{F}_{ji} = -\frac{\partial U\left(r_{ij}\right)}{\partial \vec{r}_{ij}} = -\frac{dU(r_{ij})}{dr_{ij}}\frac{\vec{r}_{ij}}{r_{ij}}$$

- i.e. Force on particle j is the opposite of force on particle *i* due to their pair interaction.
- In the code up to now, we calculate two times the distance between particles *i* and *j*, check against cutoff, do the kernel calculations. These can be performed ONLY once :
- Find distances, check cutoff, calculate force on *i* and force on *j* is $-$*Force* on *i*.

- In terms of code :

```
do i = 1, N
   do j = 1, N
      if (i.ne.j ) then
      .....
      f(i)= f(i) + ...
      endif
   enddo
enddo
```

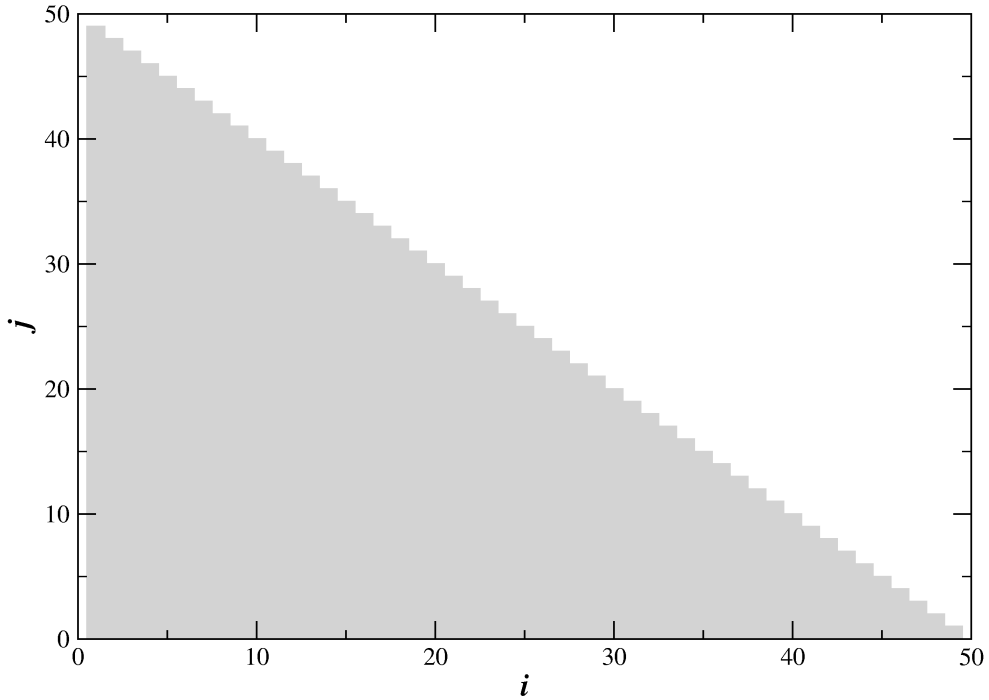
  can be replaced with

```
do i = 1, N - 1
   do j = i + 1 , N
      f(i) = f(i) + ...
      f(j) = f(j) - ...
   enddo
enddo
```

- Loop Count : $\frac{N^2-N}{2}$ vs $(N-1)^2$, ~ 0.5.
- Up to now we have for large enough N
  $\sim 0.5 \times 0.52 = \sim 0.26$ less operations to perform.

- Other optimizations ?

- Increasing *i*, the *j* loop count decreases : Load imbalance.



i

- One can do (if N is even, in case of odd N, few more lines are required, like in yesterday's chunk size calculation.)

```
do ii = 1 + me, N/2, np
   do k = 1, 2
       if (k.eq.1) then
           i = ii
       else
           i = n - ii
       endif
     ....
   enddo

enddo
```

- This type of parellelization (i.e. atom decomposition) is not used anymore or could be applied at samples of $N < 2000$ or similar. Instead, domain decomposition is used, that is another (also complicated) story.

Questions ?

Hands On

- Profile Serial, MPI, Hybrid MPI/OpenMP applications with gprof, mpiP, scalasca.
- For those who have their own Code, try to profile your own code.
- Those who are familiar with vtune, try also vtune, especially with OpenMP only codes.
- Discuss Findings, Suggestions to improve performance.