Local Validation to the reference

+ *OpenMP and optimization*

Filip Váňa

filip.vana@chmi.cz

ONPP / ČHMÚ - LACE

Outline

- Local validation of source code
- OpenMP
- Optimization

How to regard this presentation

- The aim is to install locally a model cycle.
- The local code should be made available for the others (means that some parts are supposed to be modified and recompiled) and perhaps also to store it under some versioning system.
- No specific platform, compilation tool or versioning system is a prior assumed here.
- Still the typical decisioning follows situation at CHMI (no interest to run global model, limited amount of configurations to be validated, usually all validated configurations aims to run operationally)

Before starting...

Make sure the cycle is worth to be installed in your service.

- Do we need the particular cycle?
- Is there anything interesting, we would like to have for our operational?
- Do we plan to do some new research possibly changing significant part of the code?
- If there's only a particular code which attracts us, couldn't it be just back phased or applied as a patch to my actual cycle? (Example: simple bugfix, new computation of diagnostic cloudiness, new T2m or RH2m diagnostics...)

Before starting... II.

- Validation is a long process (sometimes full of surprises)
- Not necessary new cycle = better cycle
- Usually new cycle implies higher cost and/or memory requirements
- Frequent local validation helps to keep the track with the changes

⇒ Some good compromise for the validation frequency should be found.

(at CHMI since 2003 locally installed 10 full cycles: AL25T1 , AL25T2, AL28T1 , AL28T3 , AL29T2 , AL31T0, AL32T1 , AL34, AL35T1 , AL36T1)

Before starting... III.

Now we are sure (our boss is sure) we want the new cycle.

- Take the export version (if available): cougar marp001/pub/export/
  Sometimes it needs to be taken directly from the ClearCase.
- Collect information: (read phasing reports, ask phasers, ask contributors, ...) $\rightarrow$ be sure about all norm changes, new cpp directives, new tricks in ODB,...
- Make sure about the content of reference cycle (your previous one).

Getting started...

- Unpack tarball
- Clean the content (*txt, *doc, *sh, sym links, Makefiles, ...)
- Remove projects of no specific interest (aeo, obt, scr,...)
- Change names (if needed) :
  - *.mnh/*.f90 -> *.F90
  - capital letters -> small letters (su_so4_A1B2000.F90 -> su_so4_a1b2000.F90,...)
  - writesurf_flake_confn.f90 -> writesurf_flake_conf_n.F90
- Remove duplicates (if needed)
- Move routines to their original place to maintain some inter-cycle continuity (if you care)

Local modifications

Incorporate local modification to the code not promoted to the common source

- monitor routines: wrmlppa, cnt4, sufpc, elsirf + ishell
- local fixes: (poor-man norms diagnostics: wrgp2fa)
- if necessary local version of system routines (mpif.h, ...)

Preparing executable

- Prepare compilation options: xrd, mse, mpa,...
- Scan for dependencies (*.d)
- Compile
- Fix compilation errors:
  - some notorious recidivists causing problem to a local compiler (NEC: sat/rtlimb/rtlimb_traceray_2d(_tl/_ad).F90, mpa/chem/internals/ch_jac.f90)
  - new issues (always new issues) $\Rightarrow$ fix and report back to TLS
- make executable (problem can appear also here: too many entry points, not enough memory on stack,...)

Namelist preparation

- Prepare namelist: Usually the best method is to start from previous/reference cycle namelist and to update it according to a new namelist example.
- Try to run it. If problem, go to the previous item.

Real validation starts

- See the output listing files (NODE.*)
- Make sure all tunables are equally set to the reference cycle. (tkdiff, gvimdiff,...)
- Don’t expect to find the same norms already after the first execution. (We are living in real world.)
- Target should be to see:
  - norms identity for e(e927)
  - norms identity for dynamics (adiabatic model)
  - norms identity for odb, screening,...
  - 5-6 digits identity after 5-10 time-steps for physics, e701

Real validation continues

Try to deactivate the known changes responsible for the norms difference:

- By namelist
- Directly in the code

Example *(adiabatic run of CY36T1 compared to CY35T1):*

Known changes causing norms difference:

- reordering of loops in spectral computation (espchor.F90)
- scale (in)dependent horizontal deformation computation in SLHD becomes aware of map factor (sudyn.F90)

$\Rightarrow$ By deactivating those two, the two cycles gives identical norms.

Real validation continues II.

Indeed not every time the situation is that favorable to allow a direct comparison. In such a case:

- Check less aggressive optimization, different memory organization (static, stack, OpenMP)
- Look at the results: compare fields (precipitations, vertical velocity,...)
- Compare scores against observations.
- Check some other characteristics (number of observations,...)

In general there's no universal guidance. At CHMI we keep rule to understand all differences in the norms. Any effort during this step pays back at the end!

Parallelization validation

- MPI (1,2, more CPUs) - usually OK as it is validated in Toulouse
- OpenMP more frequently problem (overseen during phasing, race conflicts invisible when OpenMP is not used)
  Beware also of:
  - Platform dependent synchronization of OpenMP in adjoint of SL code (with LVECADIN=.F.)
  - There's a bug for NPROMA optimization in LAM with more than two OpenMP threads. Always use negative value in such case.
- Mixed MPI and OpenMP solution - usually OK when MPI and OpenMP are validated.

Parallelization validation II.

OpenMP debugging crash course:

-P openmp doesn't run even for 1 thread - worst case, indicates problem in memory overwriting

-P openmp code run fine but only for 1 thread - typical problem of OpenMP

- multithread job crashes or deadlock - some part is not initialized or problem of synchronization $\longrightarrow$ fix not trivial (locate the place and ask OpenMP expert to help)
- it runs smoothly just giving wrong results - race conflict = incorrect use of private/global arrays

Problem location is simple: deactivate OpenMP for suspicious regions (at the code level) and see the impact...

Local code optimization

- profiling charts should look similar to previous cycle
- no routine with really poor performance within the TOP 20

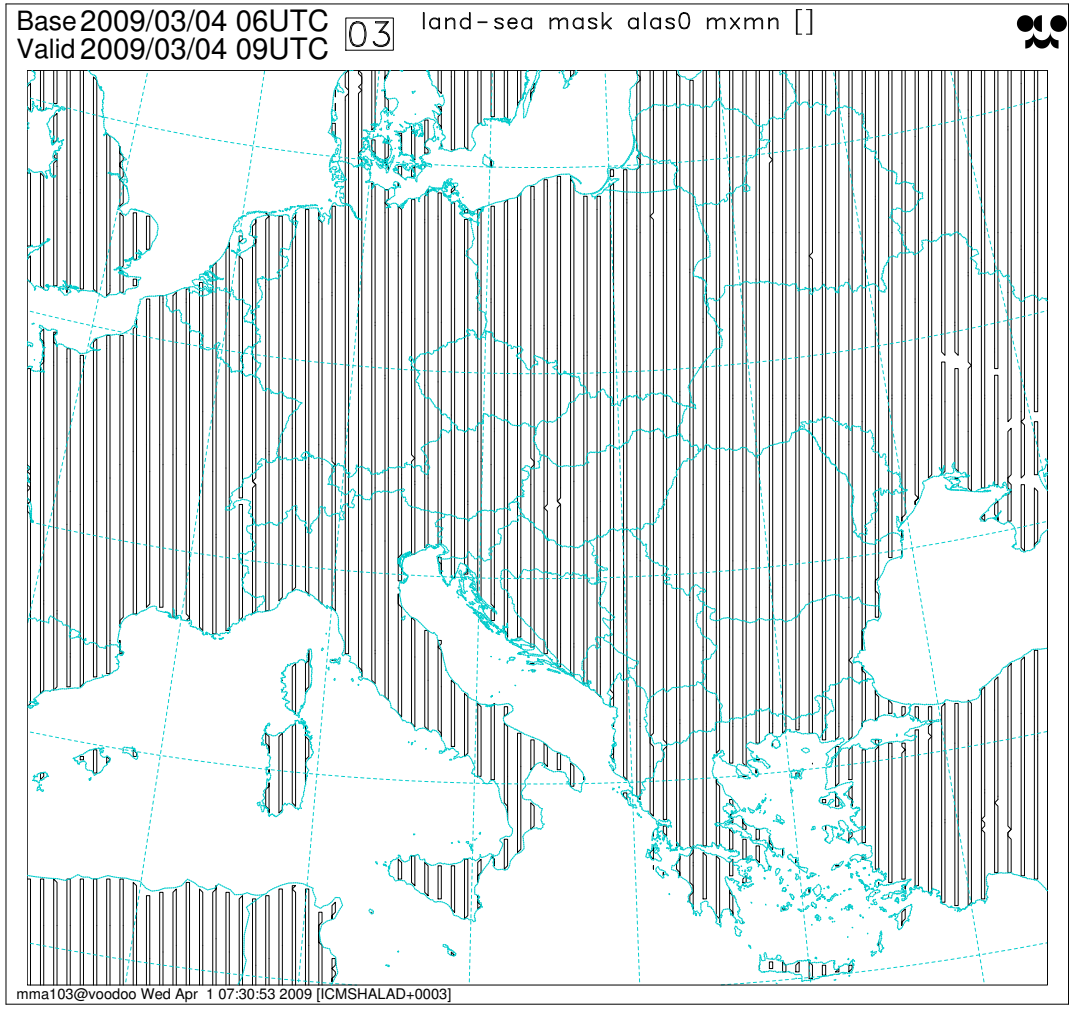
```
Total CPU Time : 0:12'02"741 (722.741 sec.)

PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.
                      TIME[sec](  % )    [msec]                  RATIO V.LEN

ismax       35578558   440.630( 61.0)     0.012    435.0     47.2 72.73 145.1
caspia        175714   118.431( 16.4)     0.674    790.2     14.6 79.86  32.9
casgqa        260113    59.605(  8.2)     0.229    490.7    131.2 28.91 144.4
minv.geco     171186    27.854(  3.9)     0.163    393.8     62.9 58.81  27.4
minv.gedi     171186    11.471(  1.6)     0.067    381.3     62.9 46.41  13.7


Total CPU Time : 0:04'08"323 (248.323 sec.)

PROC.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.
                      TIME[sec](  % )    [msec]                  RATIO V.LEN

caspia        175714   116.897( 47.1)     0.665    800.6     14.8 79.86  32.9
casgqa        260113    32.594( 13.1)     0.125   3919.2    768.7 91.59 179.5
minv.geco     171186    27.668( 11.1)     0.162    396.4     63.4 58.81  27.4
minv.gedi     171186    11.305(  4.6)     0.066    386.5     63.8 46.46  13.7
```

Output files validation

- Very important step.
- Must be done at the end.
- We are using simple tool (off the model) comparing two FA files for selected (sub)area and all records. The output is either identity or RMSE and BIAS of differences.

Output files validation

Land-sea mask difference of two model outputs



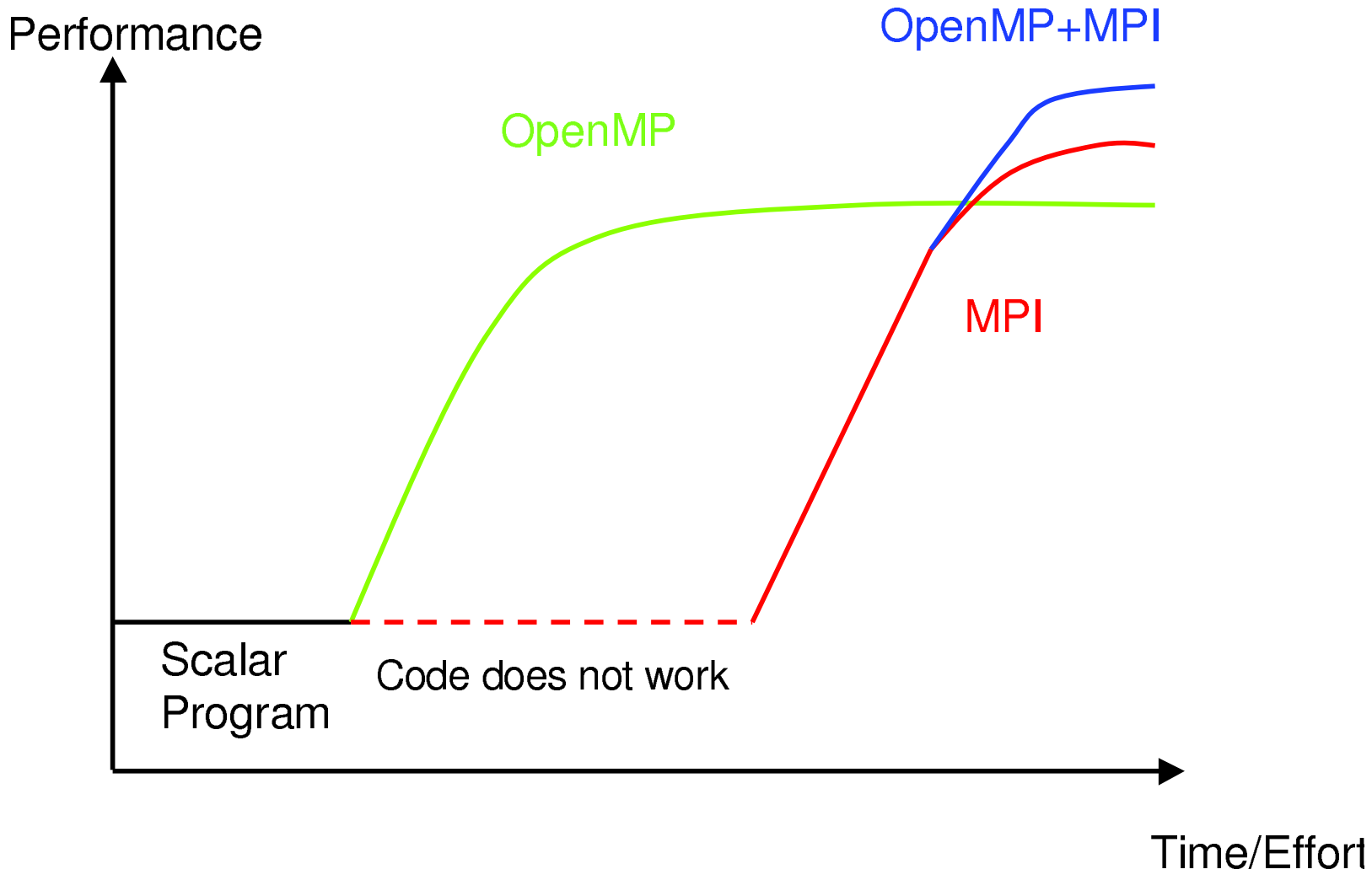
Validated...

- new cycle seems to be validated and ready for local use (operational and R&D)
- record all the code changes or tricks (for next time and for the others)
- if there's something of a common interest, don't forget to announce it (LACE forum,...) and to commit it back to ClearCase
- don't forget to continuously monitor the other's experience...

OpenMP

- model for parallel programming
- shared memory parallelization
- portable across shared-memory architectures
- scalable (usually memory access creates limitation)
- incremental parallelization
- compiler based
- extension to existing programming languages
  - mainly by directives
  - a few library routines
- supports data parallelism

Why OpenMP?


Performance
OpenMP+MPI
OpenMP
MPI
Scalar
Program
Code does not work
Time/Effort

OpenMP Programming Model

- shared memory model

OpenMP Programming Model

- shared memory model
- workload is distributed between threads

OpenMP Programming Model

- shared memory model
- workload is distributed between threads
- variable can be:
  - shared among all threads
  - duplicated for each thread

OpenMP Programming Model

- shared memory model
- workload is distributed between threads
- variable can be:
  - shared among all threads
  - duplicated for each thread
- threads communicate by sharing variables

OpenMP Programming Model

- shared memory model
- workload is distributed between threads
- variable can be:
  - shared among all threads
  - duplicated for each thread
- threads communicate by sharing variables
- unintended sharing of data can lead to race condition, i.e. when the program outcome changes as the threads are scheduled differently

OpenMP Programming Model II.

- synchronization available to control race condition
  - usually slows down the performance
  - careless use of synchronization can lead to the dead-locks

OpenMP Programming Model II.

- synchronization available to control race condition
  - usually slows down the performance
  - careless use of synchronization can lead to the dead-locks
- fork-join model of parallel execution
  - begin execution is a single process (master thread)
  - start of parallel construct: MT creates team of threads
  - completion of a parallel construct: implicit barrier
  - only master thread continues execution

OpenMP directive format (Fortran)

- treated as Fortran comments
- !$OMP directive_name[clause[[,]clause]...]
- directive can be split to several lines by &
- not case sensitive
- Conditional compilation code after: !$

OpenMP data scope clauses

- PRIVATE = variables private to each thread in a team
  - uninitialized values
  - private copy is not storage associated with the original

```
      JLAST = -999
!$OMP PARALLEL DO PRIVATE(JLAST)
      DO J=1,1000
         ...
        JLAST = J
      ENDDO
!$OMP END PARALLEL
      print *, JLAST   --> writes -999 !!!
```

OpenMP data scope clauses II.

- SHARED = variables shared among all threads in a team
  - Sharing a variable might slow execution
  - Always careful when modifying SHARED variable

Default is SHARED, but:

- local variables in called sub-programs are PRIVATE
- control variable of parallel DO loops are PRIVATE
- (automatic variables within a block are PRIVATE)

Race condition

Typical OpenMP error

```
      JSUM = 0.
!$OMP PARALLEL DO
      DO J=1,1000
         ...
        JSUM = J + JSUM
      ENDDO
!$OMP END PARALLEL
      print *, JSUM
```

$\Rightarrow$ Result varies unpredictably based on specific order of execution of the parallel section.

Program is wrong, but there will be no warning!

Example of OpenMP from Aladin

Extract from CALL_SL_AD:

```
!$OMP PARALLEL DO SCHEDULE(DYNAMIC,1) PRIVATE(JSTGLO,IST,IEND,IBL,IOFF)

DO JSTGLO=1,KGPTOT,NPROMA
  IST=1
  IEND=MIN(NPROMA,KGPTOT-JSTGLO+1)
  IBL=(JSTGLO-1)/NPROMA+1
  IOFF=JSTGLO

  CALL LAPINEA5(IST,IEND,.TRUE.,&
   & PCOLON(IOFF),PSILON(IOFF),PGM(IOFF),PGEMU(IOFF),&
   & PGSQM2(IOFF),PGECLO(IOFF),PGESLO(IOFF),PGNORDL(IOFF),&
   & PGNORDM(IOFF),PRINDX(IOFF),PRINDY(IOFF),
   & PB15(1,1),PB2(1,1,IBL),&
   & ZLON5(1,1,1,IBL),ZLAT5(1,1,1,IBL),&
   & ZQX5(1,1,1,IBL),ZQY5(1,1,1,IBL),&
   & ZUF5(1,1,1,IBL),ZVF5(1,1,1,IBL),&
   & IL0(1,1,0,1,IBL),ILH0(1,1,0,IBL),&
   & ZDLO5(1,1,0,1,IBL),ZCLO5(1,1,1,1,IBL),ZCLOSLD5(1,1,1,1,IBL),&
   & ZDLAT5(1,1,1,IBL),ZCLA5(1,1,1,IBL),ZCLASLD5(1,1,1,IBL),&
   & ZDVER5(1,1,1,IBL),ZVINTW5(1,1,1,IBL),ZVINTWSLD5(1,1,1,IBL),&
   & ZCOSCO5(1,1,1,IBL),ZSINCO5(1,1,1,IBL),&
   & ZCOPHI5(1,1,1,IBL),ZSINLA5(1,1,1,IBL),&
   & ZLEV5(1,1,1,IBL),ZOUT5(1,1,1,IBL),IDEP(1,1,IBL))
ENDDO
!$OMP END PARALLEL DO
```

OpenMP summary

- increasing importance for multi-core systems
- capable to improve load balancing
- reduces number of MPI processes - helps if scalability is an issue
- depends on size of the problem (STATIC versus DYNAMIC scheduling) - decreasingly important in Aladin as our domains gets sufficiently big (still an issue for vector computers)
- platform dependent (IBM doesn't support to call function from parallel region; to replace such function by subroutine would break vectorization on NEC...)
- easy to implement (mostly directives and not much to change in the code - the bit-wise reproducibility might be an issue)
- optimization: As much parallel code as possible, without frequent synchronizations

Optimization

Typical optimization activities

- Start with profiling - ftrace, Dr.Hook, xprofiler, hpm
- Check compilation logs (*.L files)
  - problems with vectorization
  - some feature prevents optimization
  - more appropriate compilation options & directives
  - problem can be caused by bug in compiler
- More directives (NODEP, UNROLL,...)
- Incorporate intrinsic functions or idioms:
  - sgemmx (NEC - MATMUL, VPP - dvmggm)
  - laitri (IBM - fsel)
- Add more OpenMP parallel regions
- Remove copies and zeroing of arrays

Optimization II.

The code is multi-platform...

- Always keep in mind the code is used by other centers with different computers and for different domains. Would be your optimization optimal for everyone?
- The best solution is the same solution for everybody.
- Duplication of code is possible (if unavoidable) but it increases amount of maintenance (validation) and usually makes the source less readable.
- Unnecessary code duplication is justified for routines:
  - belonging to TOP 10 exclusive time performers
  - if there's a chance to speed up the whole code performance by at least 3-5%
  - when there's no other chance to run code for a given platform

Optimization III

- Optimization is a continuous process
- Always the best way of optimization is to write efficient code
- Make sure the "optimized code" will outperform the previous one also for different problem size
- Make sure the "optimized code" will outperform the previous one also for different number of processors/different parallelization technique
- Make sure the source of problem is well understood prior to any optimization

Profiling interpretation

Pure OpenMP job (9 threads) profiling for the CPG on SX9:

PROC.NAME	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	VECTOR TIME	BANK CONFLICT CPU	BANK CONFLICT PORT	BANK CONFLICT NETWORK
cpg$1	729	13.935(4.0)	19.116	2331.2	0.1	98.70	249.2	8.242	0.326		6.503
-micro1	81	0.995(0.3)	12.289	3665.4	0.1	99.14	249.5	0.945	0.037		0.749
-micro2	81	1.292(0.4)	15.954	2829.9	0.1	98.91	249.5	0.928	0.037		0.732
-micro3	81	1.233(0.4)	15.228	2963.7	0.1	98.95	249.5	0.926	0.037		0.730
-micro4	81	1.303(0.4)	16.085	2807.1	0.1	98.90	249.5	0.928	0.037		0.732
-micro5	81	1.276(0.4)	15.755	2865.4	0.1	98.92	249.5	0.925	0.037		0.729
-micro6	81	1.270(0.4)	15.677	2879.4	0.1	98.92	249.5	0.927	0.037		0.731
-micro7	81	1.278(0.4)	15.775	2861.7	0.1	98.92	249.5	0.927	0.037		0.731
-micro8	81	1.265(0.4)	15.615	2891.0	0.1	98.93	249.5	0.925	0.037		0.729
-micro9	81	4.023(1.1)	49.662	805.7	0.0	96.43	246.3	0.812	0.032		0.641

What is wrong with CPG? And where the imbalance of the last thread comes from?