MPII - parallelization

C. Kleinwort

DESY/HH tracker alignment chat 05.10.10

Introduction

- Single most cpu time intensive part of MPII has been parallelized with OpenMP™
- Next steps
  - Quantify parallelization
  - Improve parallelization
- Test case: bows&kinks with CRAFT10 at NAF
  - 194k parameters, 1080k tracks, X5550 2.67 GHz

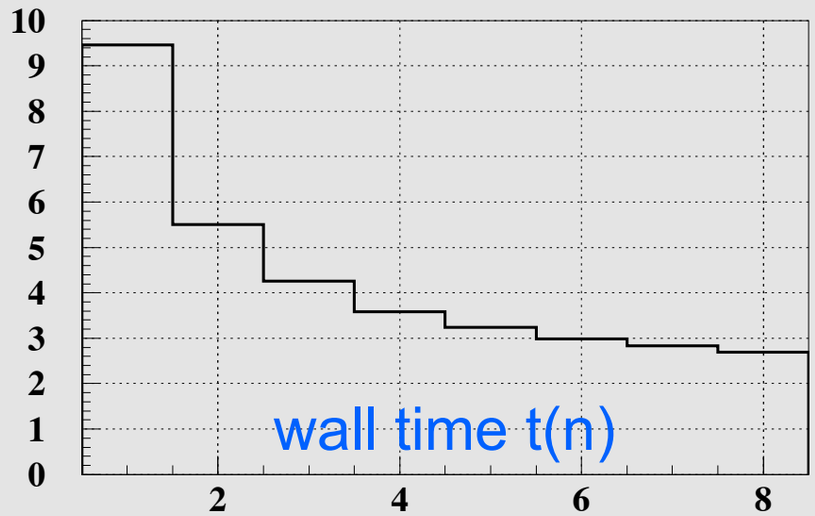
MPII work flow

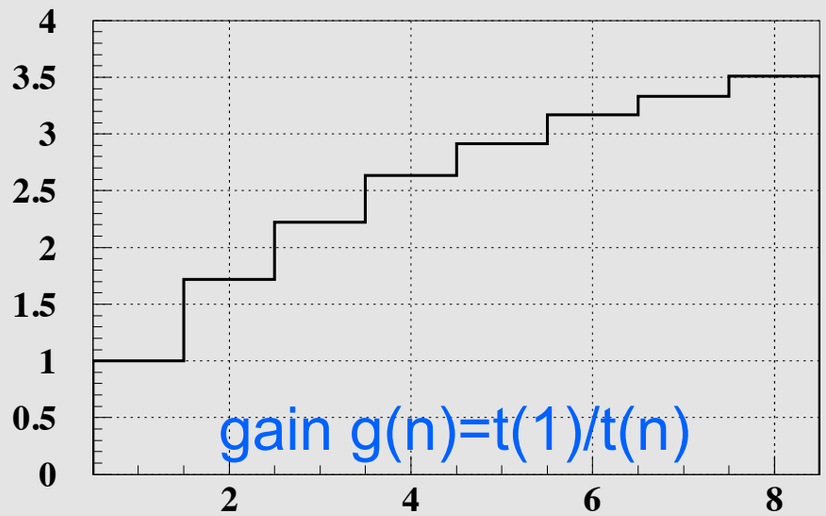
- LOOP1
  - Detect (variable) global parameters
- LOOP2
  - Detect global parameter pairs, build sparsity info
- LOOPN, iterated
  - For each record (track)
    - Local (track) fit
    - Update global vector **b**
    - Update global matrix ***A***, only first iteration (usually)
  - Global fit $\boldsymbol{A}\cdot\boldsymbol{x}=\boldsymbol{b}$, the only parallelized part (MINRES)

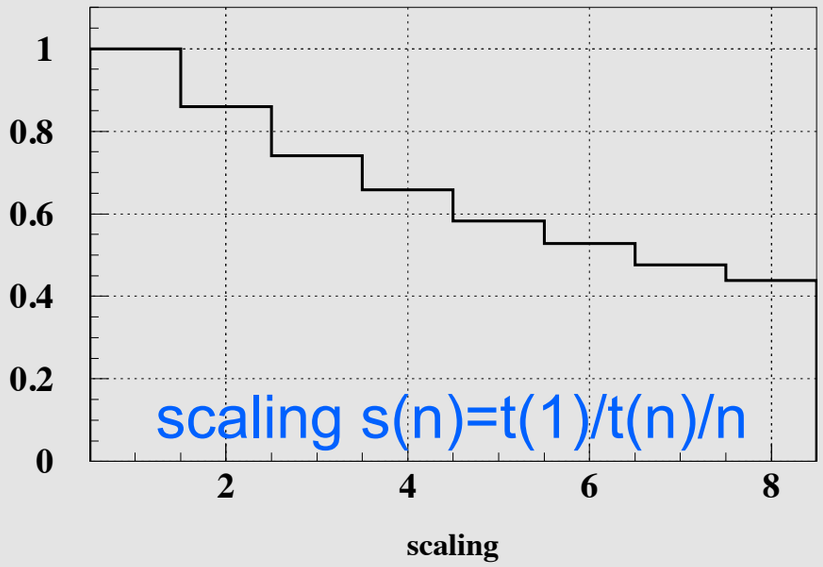
Quantify parallelization

- Use OpenMP profiler **ompP** (http://www.ompp-tool.com)
  - GPL, by Karl Fuerlinger, University Tennessee
  - Source code instrumented by prepending script (`kinst-ompp`) to compiler and linker (+ `libompp.a`)
  - OpenMP constructs instrumented automatically
  - Profiling information as text or csv file
  - Measure integrated wall time for serial (t_s) and parallel (t_p) code regions and OpenMP overhead
  - Total time with n threads: $t(n) = t_s(n) + t_p(n)/n$

Profiling Rev64







n_{thread}

$t_s \approx$ 5900s, ~ n_{track}

$t_p \approx$ 29000s, ~ n_{par}^2

Improve parallelization

- Caching
  - Input/Output difficult to parallelize
  - Read (serially) many records into read cache, analyze them in parallel
- Automatic (FORTRAN) variables
  - Parallelized code needs to call external subroutines
    - Linear algebra (track fitting), update of sparse matrix, ..
  - Local variables must be on the stack (local to thread)
    - Compilation with `-fautomatic` needed
    - General `SAVE` statements not allowed

LOOP1/2 parallelization

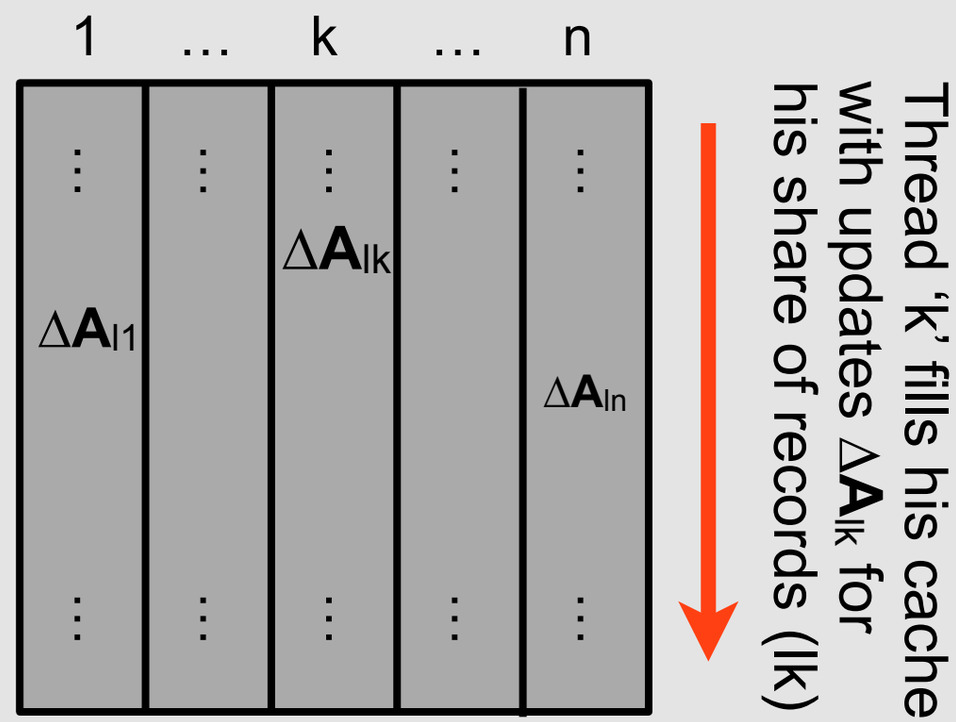
- LOOP1
  - Stays serial
- LOOP2
  - Building of sparsity structure parallelized
    - Counting of parameter pairs (i,j) (INBITS) ~ n_{track}
    - Analysis of pair counters (NDBITS) ~ n_{par}^2
    - Creation of sparsity structure (SPBITS) ~ n_{par}^2
  - Parallelization by row number (=max(i,j))

LOOPN parallelization (I)

- Local fit, update of **b** (FITLOC) ~ n_{track}
  - Parallelization by record (track)
  - Monitoring, histogramming only with one thread
- Global fit
  - Solution with MINRES, parallelization by row number (AVPROD) ~ n_{par}^2

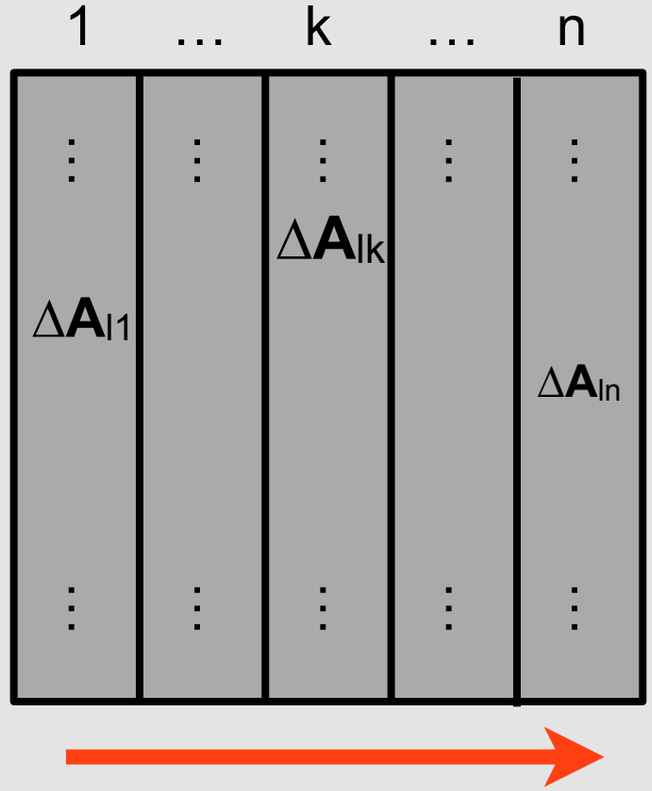
LOOPN parallelization (II)

- ★ Update of ***A*** (MUPDAT) ~ n_{track}
  - ▸ Update ΔA
    - ✦ Matrix of size of number of global parameters in record
  - ▸ Caching, parallelization by record
    - ✦ Different records (threads) could update same part of ***A***!
    - ✦ To avoid access conflicts put updates ΔA into write cache (one per thread)



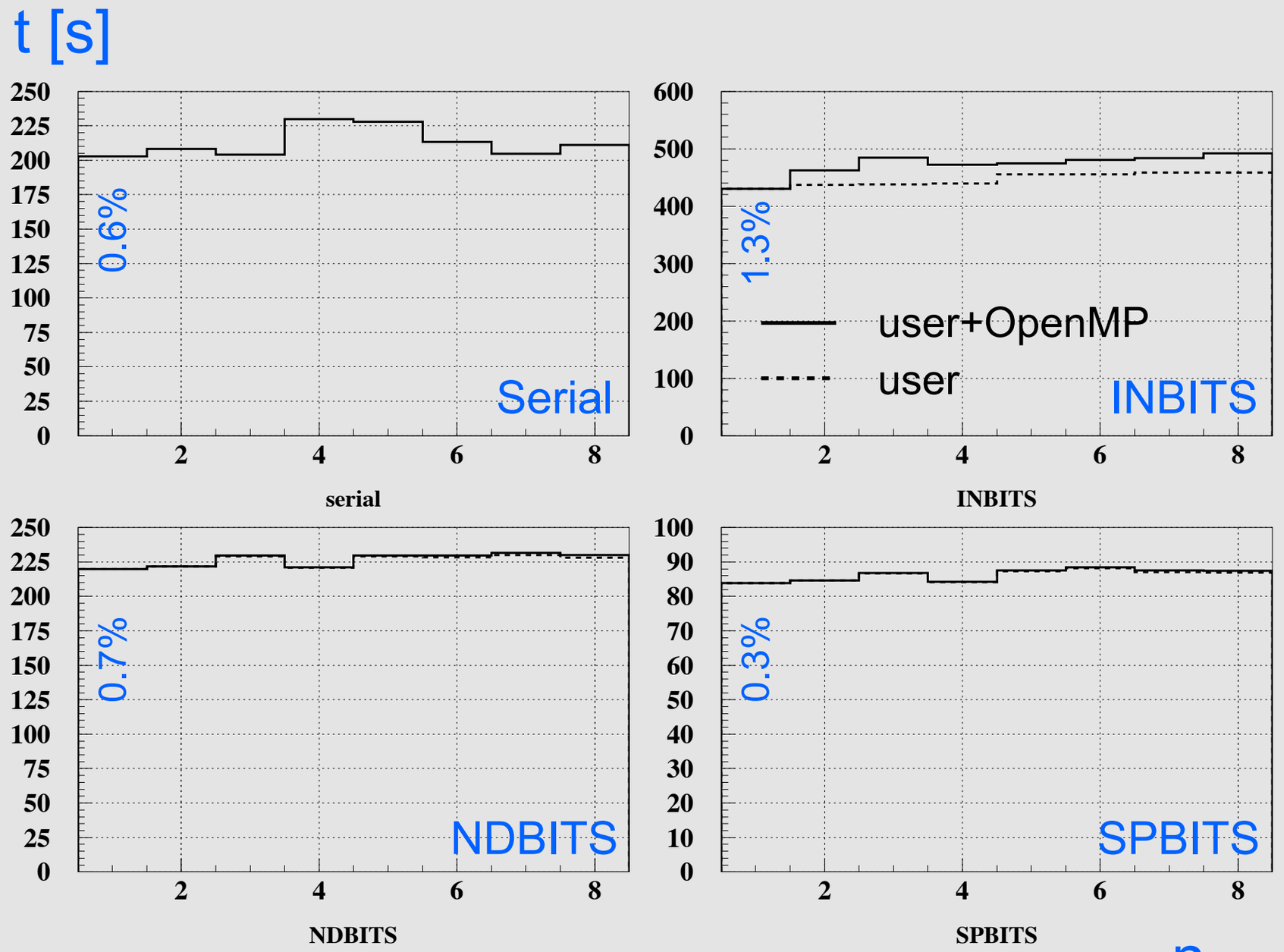
LOOPN parallelization (III)

- Flushing, parallelization by row number
  - Read cache empty before write caches full: flush parallel
- Write cache overrun: flush in serial mode
  - Thread empties write cache for all rows, all other threads have to wait (CRITICAL region)
- Automatic adjustment of read/write cache sizes to avoid overruns

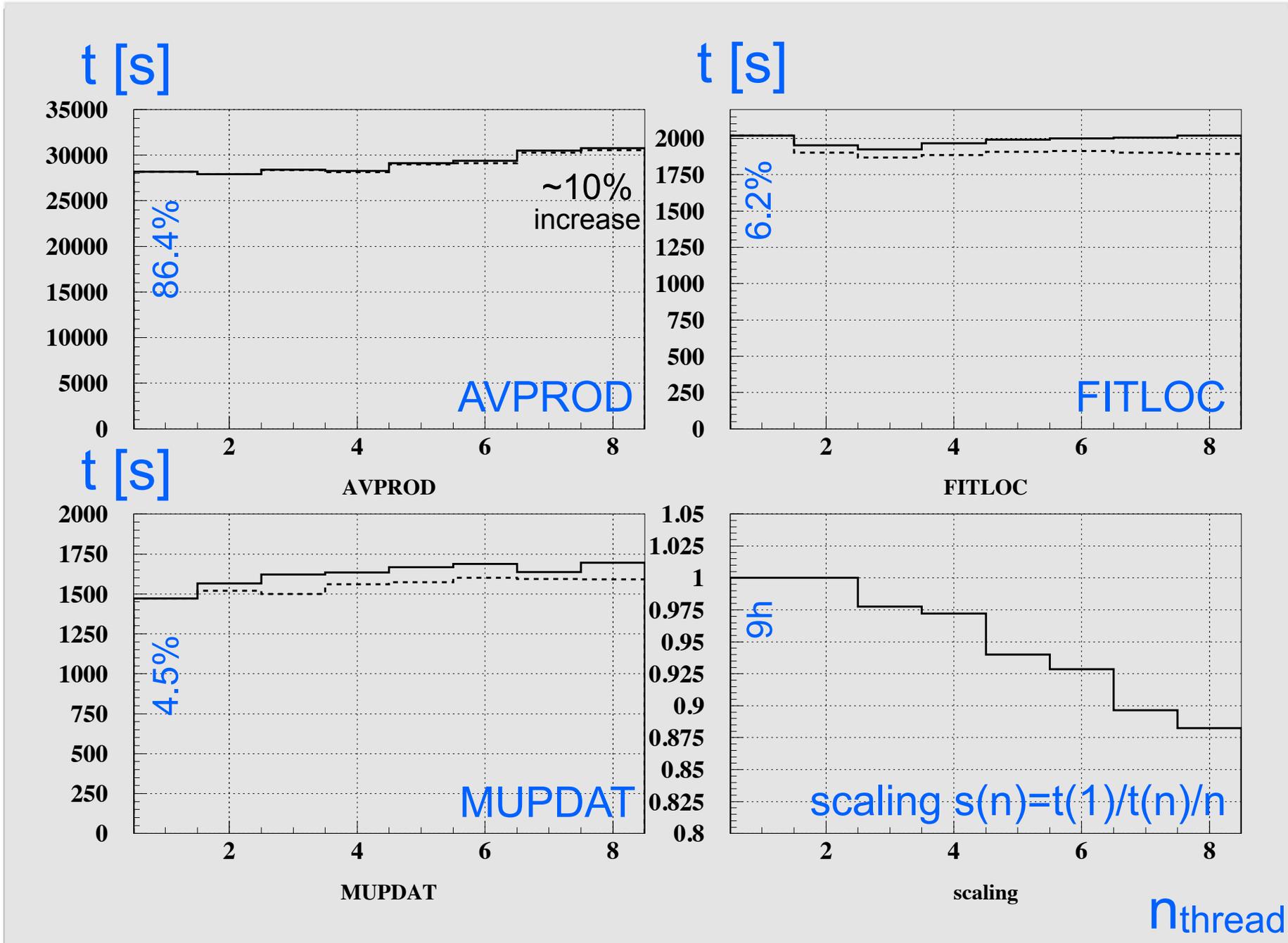


Thread ‘k’ flushes for all caches his share of rows ‘i’ (mod(i,n)=k-1)

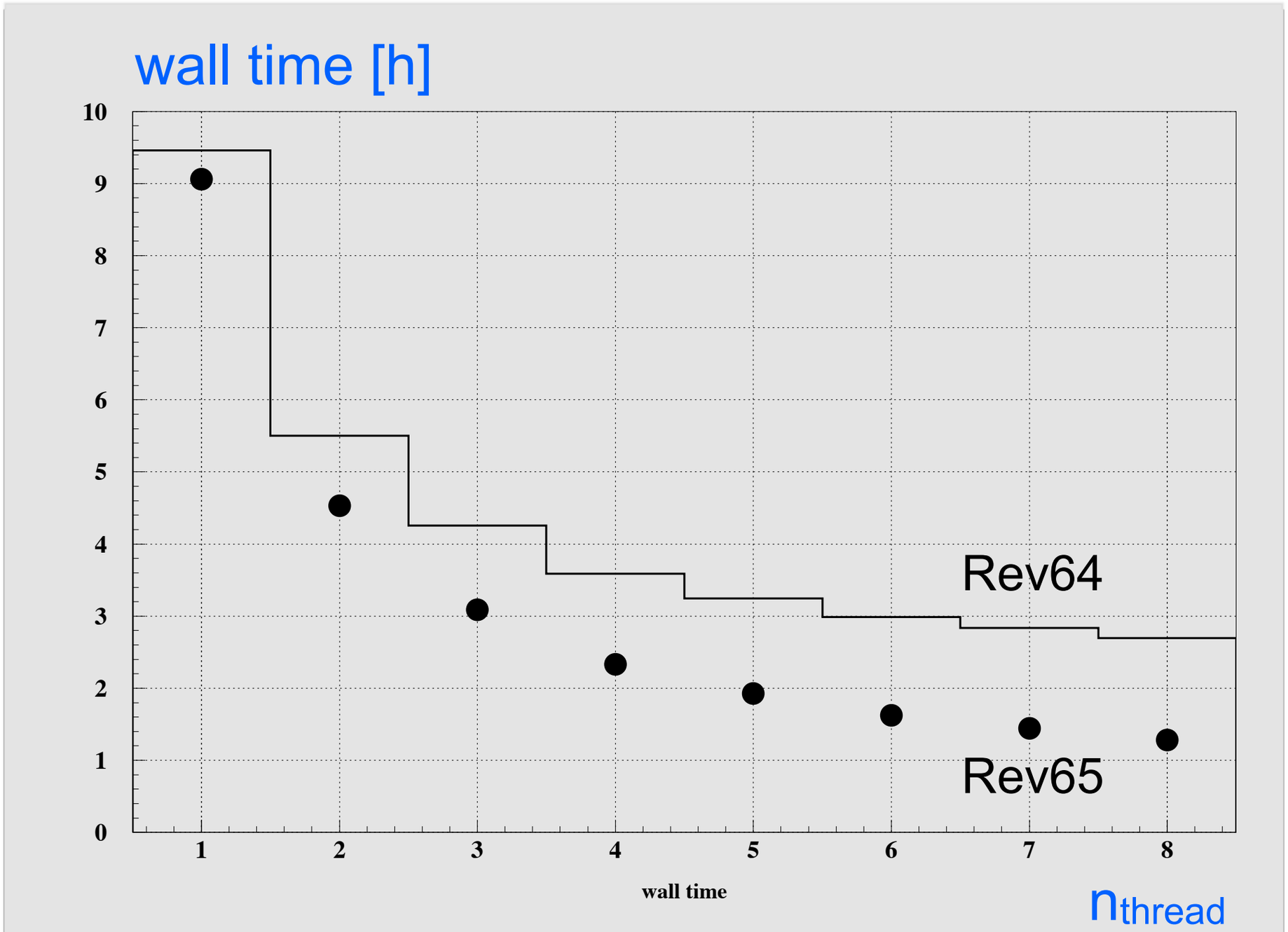
Profiling Rev65 (I)



Profiling Rev65 (II)



Rev64 vs Rev65



Summary

- Parallelization in MillePede-II revision 65
  - All major loops ~n_{par}^2 or ~n_{track}
    - Except reading binary files (~70% of remaining serial time)
- Performance (from ompP, 1-8 threads)
  - OpenMP overhead ≤1%, mainly implicit barrier (wait for last thread at end of a parallel region)
  - Parallel coverage ($t_p/(t_s+t_p)$) ≥95%
  - Good scaling (still 90% with 7 threads)
    - Time for global fit (AVPROD) increases by 10%, (distributed) memory access ?