Hybrid parallelization of maximum likelihood fitting with MPI and OpenMP

Alfio Lazzaro
alfio.lazzaro@cern.ch
CERN openlab

CERN openlab Minor review meeting

24 April 2012

Maximum Likelihood Fits

- It allows to estimate free parameters over a data sample, by minimizing the Negative Log-Likelihood (NLL) function

$$NLL = \sum_{j=1}^{s} n_j - \sum_{i=1}^{N} \left[\ln \sum_{j=1}^{s} \left(n_j \prod_{v=1}^{n} \mathcal{P}_j^v(x_i^v | \hat{\theta}_j) \right) \right]$$

N number of events
$\hat{x}_i$ set of observables for the event i
$\hat{\theta}$ set of parameters

n observables
$\mathcal{P}$ probability density function
s species, i.e. signals and backgrounds
n_j number of events belonging to the species j

- The procedure of minimization can require several evaluation of the NLL
  - Depending on the complexity of the function, the number of observables, the number of free parameters, and the number of events, the entire procedure can require long execution time
  - Mandatory to speed-up the evaluation of the NLL

Algorithm Description

- Recalling the NLL definition

$$NLL = \sum_{j=1}^{s} n_j - \overset{\text{④}}{\boxed{\sum_{i=1}^{N}\left[\ln \overset{\text{③}}{\boxed{\sum_{j=1}^{s}\left(n_j \overset{\text{②}}{\boxed{\prod_{v=1}^{n} \overset{\text{①}}{\boxed{\mathcal{P}_j^v(x_i^v|\hat{\theta}_j)}}}}\right)}}\right]}}$$

① Each $\mathcal{P}$ (Gaussian, Polynomial,…) is implemented with a corresponding class (basic PDF)
  - Virtual protected method to evaluate the function
  - Virtual public method to return the normalized value

② Product over all observables (composite PDF)

③ Sum over all species (composite PDF)

④ Reduction of all values

Implementation description

- The code is implemented in a library used for different users analyses
  - ROOT/RooFit in C++ code (official code used in HEP analyses)
- CERN openlab activity to improve RooFit based on a prototype (~5K lines of code)
  - Optimization, vectorization, parallelization
- Summary of changes for optimization and vectorization:
  - Input data are organized in memory as vectors
    - A vector for each observable
    - Improve access to memory by overlapping computation and memory accesses
  - A loop executed inside each PDF over the values of the observables
    - A loop iteration per each input event
    - Use Intel compiler for the auto-vectorization of the loops (using svml library by Intel)

Optimization and vectorization: Performance results

- Testing on dual-socket Sandy Bridge-EP server, CPU E5-2680 @ 2.7GHz (Turbo OFF), dual socket, 8*2 cores, 20*2MB L3 cache
- Intel C++ compiler version 12.1.0
- Input data is composed by 1,000,000 events per 3 observables, for a total of about 12MB; results are stored in 29 vectors of 1,000,000 values, i.e. about 230MB
- ~90% of the execution time of the sequential code is spent in floating-point operations
- Results:
  - Original RooFit algorithm: 5726s
  - New algorithm (vectorization off): 2097s
  - New algorithm (AVX vectorization): 1054s

Vectorization gives a 2.0x speed-up (AVX)

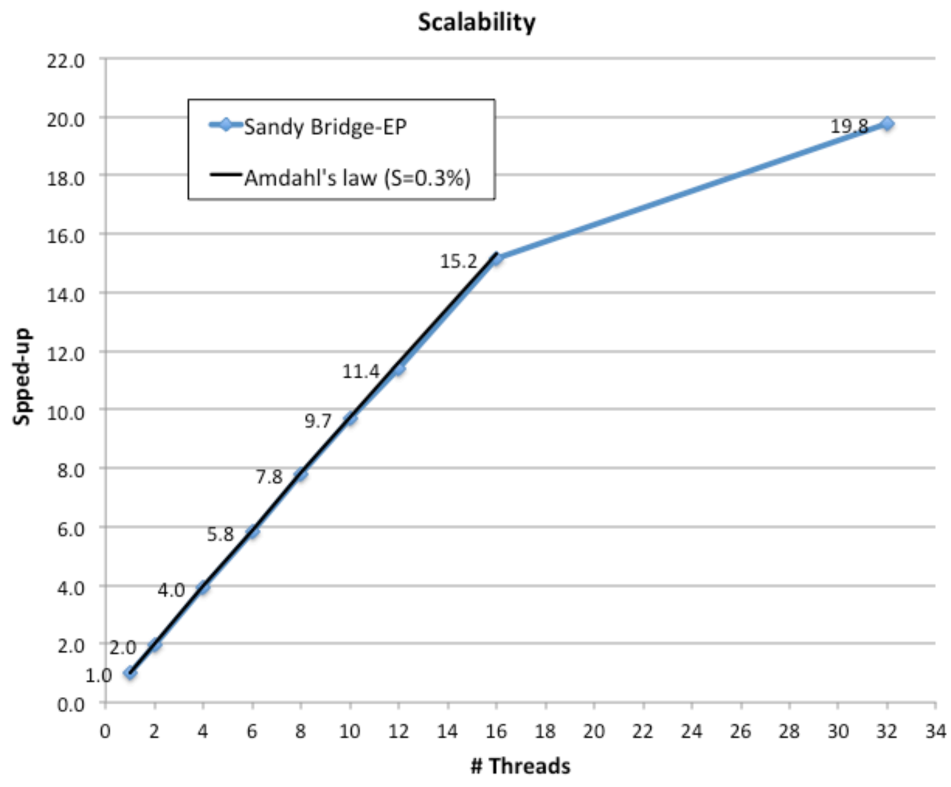
- **Total speed-up: 5.4x**

Parallelization

- Split of the loop iterations (independent)
  - Decomposition of the input events in chunks to be executed in parallel
  - Easy to balance: each chunk is composed by the same number of events
- Final parallel reduction to evaluate the NLL value executed in parallel
  - Predictable, takes in account floating point rounding problem
- Very easy parallelization with OpenMP
  - Input data are shared in memory
  - Start only a single OpenMP parallel region for each NLL evaluation: minimum OpenMP overhead, take in account possible race conditions

OpenMP scalability results

- Sequential portion 0.3%
- Satisfactory result
  - Close the Amdahl's law prediction
  - 19.8x with 32 SMT threads! (times 5.4x with respect to original single-threaded RooFit)



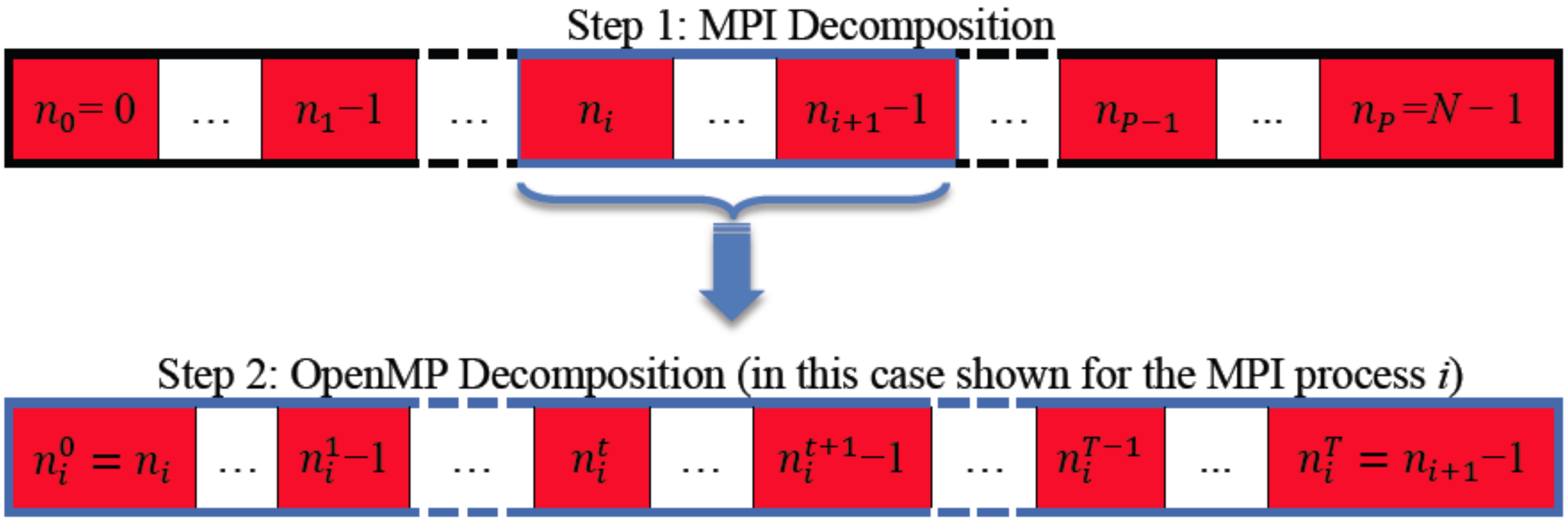
MPI parallelization

- MPI parallelization allows going beyond the constraint of the parallel execution on a single host
  - MPI standard *de facto* for massive HPC parallelization on distributed hosts connected by network links
- The standard does not make any basic distinction whether the MPI processes are running on single multicore host or they are distributed on independent hosts
  - In the case of multicore systems it is possible to consider the hybrid parallelization where each MPI process can run several OpenMP parallel threads
    - It becomes possible to exploit both shared memory parallelism enforced by OpenMP and message passing parallelism between processes enforced by MPI

Adding MPI support

- Modification of the OpenMP implementation to exploit also MPI in the computation
- Each MPI process holds a copy of the whole input dataset
  - Increase memory footprint
- Same algorithm of the decomposition of the data elements used in OpenMP-only implementation applied twice:
  - Step 1 for the MPI processes
  - Step 2 for the OpenMP threads belonging to each MPI process

MPI+OpenMP data decomposition



- P is the number of MPI processes involved, T is the number of OpenMP threads.
- OpenMP thread $t = 0,1,\ldots(T-1)$ of the MPI process $i = 0,1,\ldots(P-1)$ runs on the elements of the input data arrays with indices in the range $[n_i^t, n_i^{t+1}-1]$.

MPI+OpenMP *NLL* evaluation

1. Performing the loops on the elements inside each OpenMP thread of each MPI process
2. Reduction for the OpenMP threads of each MPI process
   - Each MPI process holds a partial result of the reduction
3. Broadcast all partial results to all MPI processes, so that each MPI process will have all partial results
   - Based on MPI function **`Allgather`**
   - This is the only communication function (one per *NLL* evaluation) ➔ Very limited MPI communication overhead
4. A second reduction is executed on the MPI partial results to get the final results on all MPI processes
5. All MPI processes will proceed to execute the same part of code (e.g. the minimization in a maximum likelihood fit)
   - Reduce MPI communications in the remaining part of the application
   - At the very end of the application each MPI process will have the same final results

MPI+OpenMP implementation

- Initial activity started last year with a CERN openlab summer student (R. Caravita, see report on the openlab webpage)
- Possibility to compile without MPI support without losing functionality, i.e. switch to the OpenMP-only parallelization
  - Based on preprocessor macros

```
// the purpose of this macro is shutting down MPI
#ifdef ENABLE_MPI
    #define MPISafeCall(p)      p
    #define MPIElse(p)
#else
    #define MPISafeCall(p)
    #define MPIElse(p)          p
#endif
```

- MPI calls encapsulated in a singleton class
  - Decoupling the MLfit code from the direct MPI calls
- Handling the printing to standard output
  - Only MPI process with rank 0 can print

Single-host performance results

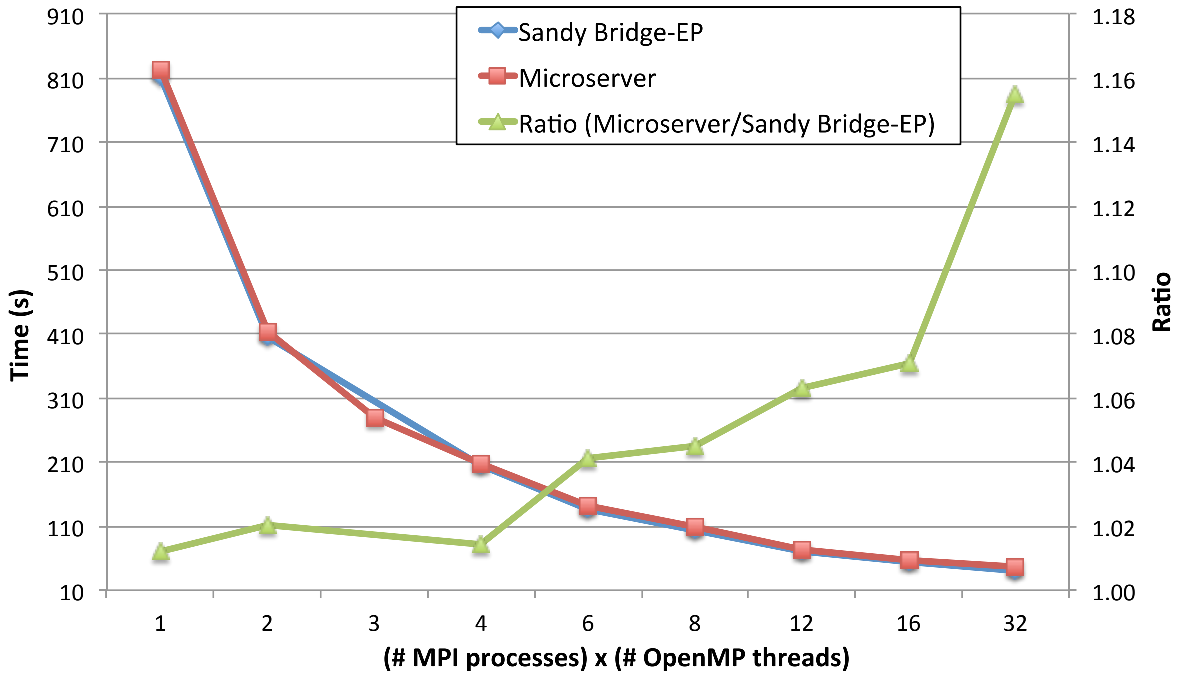
- OpenMP threads of the MPI processes are bound to cores of CPUs on different sockets before filling the cores of a given CPU
  - Maximize available cache per thread
  - Take in account NUMA effect in a multi-socket system
  - E.g.: 4 MPI processes with 2 OpenMP threads each on a dual-socket system: 2 MPI processes per socket, i.e. 4 OpenMP threads per socket
- Configuration with (# MPI)x(1 OpenMP) gives about 2% better performance with respect to (1 MPI)x(# OpenMP)
  - Better access to the input data (replicated per each MPI process)
  - Side effect: increase memory footprint
- Good tradeoff when considering a MPI process per socket, so that corresponding OpenMP threads run on that socket
  - +1% for a dual-socket, +2% for a quad-socket (Westmere-EX system)

Multi-host performance results

- Testing on DELL C5220 Microserver, 4 hosts based on single-socket Sandy Bridge desktop, CPU E3-1280 @ 3.50GHz (Turbo OFF), 4 cores, 8MB L3 cache
  - One Ethernet link per host @ 1Gb
  - Report in preparation at openlab on the evaluation of the system
- Process topology to maximize the number of hosts, with a single MPI process per each host
- Comparison of the performance with the Sandy Bridge-EP system (frequency scaled)
  - Same number of total cores
  - Smaller L3 cache size on the CPU desktop version

Comparison

- Main limitation comes from the smaller L3 cache
  - Small penalty (1%) already with a single process
  - Higher penalty (16%) when the desktop CPU are fully loaded (32 threads in total)
- Analysis of the MPI communication time shows no penalty to the scalability
- Overall good scalability
  - Microserver can be a suitable solution with respect to a standard server (dual- or even quad-socket)



Conclusion

- Code will be released in the next ROOT release (June 2012)
  - At least the optimization&vectorization and the OpenMP parallelization
  - Validation ongoing
  - Benefit for several LHC analyses
- Presented in several conferences (IPDPS, ParCO, CHEP, ACAT)
- Used as benchmark for CERN openlab and Intel activities
- From the hardware perspective, a system based on microserver can be used instead of a conventional server
  - Note that up to 12 hosts can be embedded
- A report is in preparation describing the implementation and results