2nd PhD Meeting

Konstantinos Iliakis

September 8, 2017

Table of Contents

BLonD Main Data-structures

Table 1: Overview of Main Data-structures

Name	**Type**	**Length**	**Typical Size**	**Description**
dt	1D float64 array	#particles	1M-100M	Time coordinates of particles
dE	1D float64 array	#particles	1M-100M	Energy coordinates of particles
profile	1D float64 array	#slices	1K-100K	Beam profile (histogram)
bin_centers	1D float64 array	#slices	1K-100K	Center of each bin of the beam profile
voltage (phi_rf)	1D float64 array	#slices	1K-100K	RF voltage program
induced_voltage	1D float64 array	#slices	1K-100K	Induced voltage from the sum of the wake sources in [V]
total_impedance	1D float64 array	#n_fft	100K-1M	Total impedance of all sources in [Ω] as seen by each particle

BLonD Main Functions

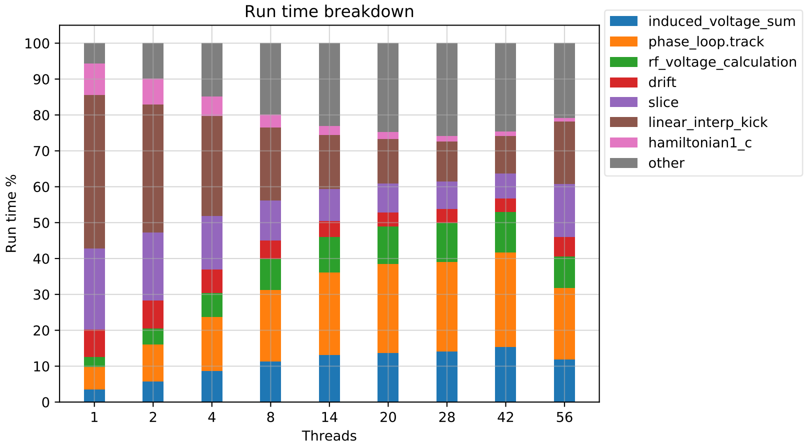
Table 2: Inputs/Outputs of BLonD main functions

Name	In	Out
kick()	dt, dE	dE
drift()	dt, dE	dt
lin_interp_kick()	dt, dE, voltage	dE
histogram()	dt	profile
induced_voltage_sum()	profile	induced_voltage
phase_loop.track()	profile	omega_rf, phi_rf
cavity_feedback()	profile, voltage, phi_rf	voltage, phi_rf

Table 3: Description of BLonD main functions

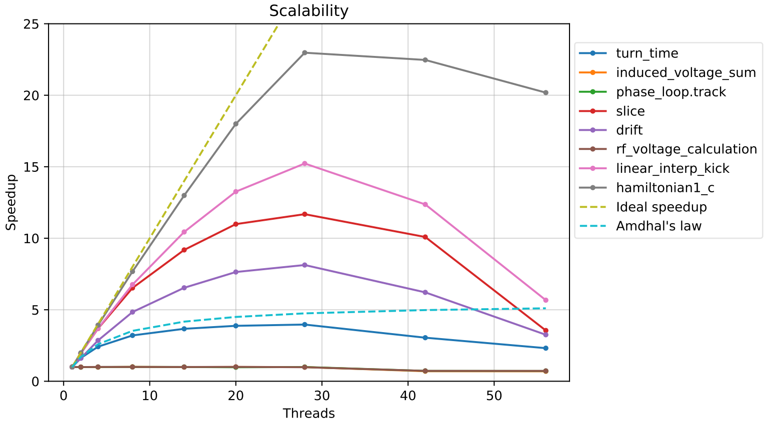
Name	Operations	Complexity	Remarks
kick()	2 nested loops, mul/ add/ sin	$\mathcal{O}(n)$	vectorization, OpenMP
drift()	1 loop, mul/ add	$\mathcal{O}(n)$	vectorization, OpenMP
lin_interp_kick()	1 loop, mul/ add, random mem	$\mathcal{O}(n)$	vectorization, OpenMP
histogram()	1 loop, mul/ add, random mem	$\mathcal{O}(n)$	vectorization, OpenMP
induced_voltage_sum()	rfft, irfft	$\mathcal{O}(n \log n)$	regular numbers
phase_loop.track()	convolution	$\mathcal{O}(n^2)$ or $\mathcal{O}(n \log n)$	–
cavity_feedback()	ffts/ convolution	$\mathcal{O}(n \log n)$	–

LHC Execution Time Breakdown



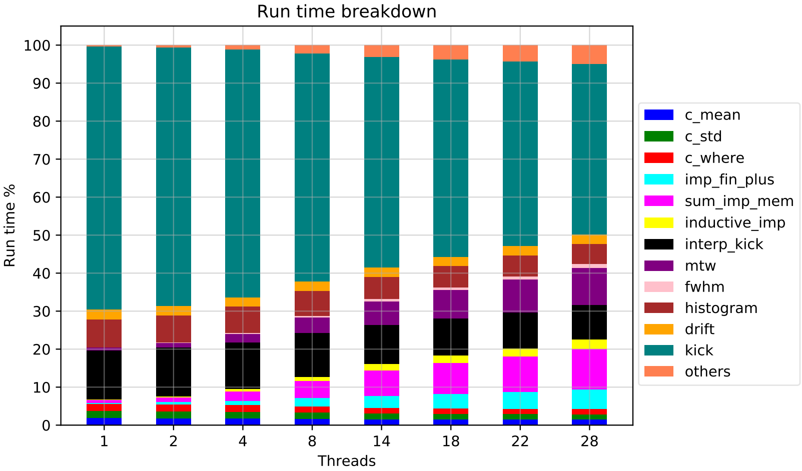
- Parallelized part: 82%
- Serial part when using 28 threads: 77%

LHC Scalability



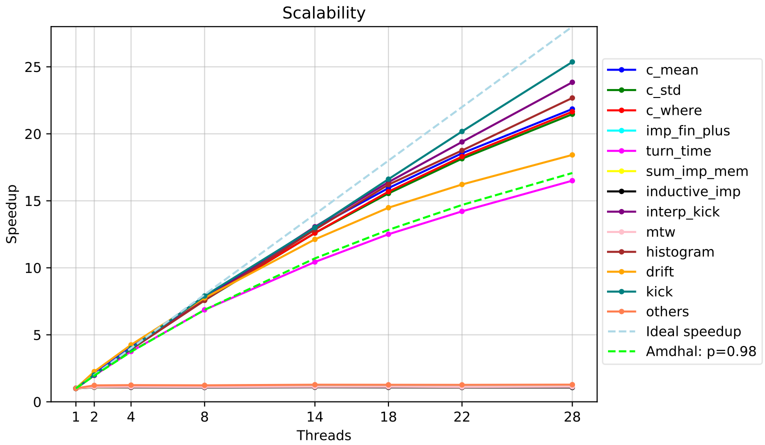
- Theoretical peak speedup: 5.5x
- Bottlenecks: `induced_voltage_sum()`, `phase_loop.track()` and `rf_voltage_calculation()`

PSB Execution Time Breakdown



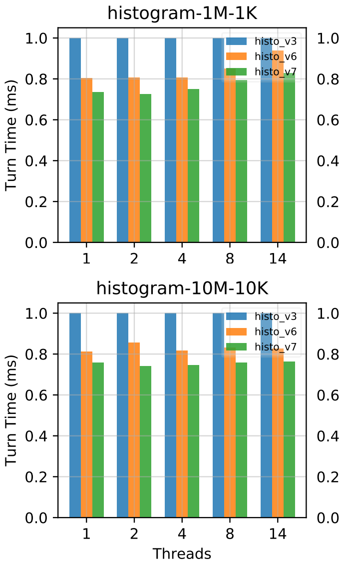
- Parallelized part: 98%
- Serial part when using 28 threads: 34%

PSB Scalability



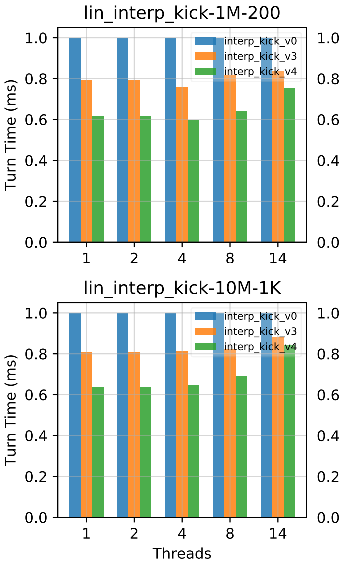
- Theoretical peak speedup: 42x
- Bottlenecks: `imp_fin_plus()`, `sum_imp_mem()`, `inductive_imp()`, `mtw()`, `fwhm()`

Histogram



- histo_v3: Original implementation
- histo_v6: New version compiled with gcc5.1
- histo_v7: New version compiled with icc17
- Optimization: Loop tiling to enforce auto-vectorization in part of the loop
- 20-25% average speedup

Linear Interpolation



- interp_kick_v0: Original implementation
- interp_kick_v3: Pre-calculate part of the loop independent of the particle coordinates
- interp_kick_v4: + loop tiling to enforce auto-vectorization in part of the loop
- all versions compiled with gcc5.1
- 35-40% average speedup

Python PAPI Library

Motivation

- Need a way to extract info about processor counters in python.
- Counters are meaningful only when combined in metrics.

Implementation

1. Build a C library (backend) that uses the PAPI interface to extract native and preset events.
2. Expose the C library to Python with the ctypes library.
3. Build a Python module with preset metrics that will communicate with the backend to read the necessary counters and compute the requested metrics.
4. Metrics found in Intel64 and IA-32 Architecture Optimization Manual

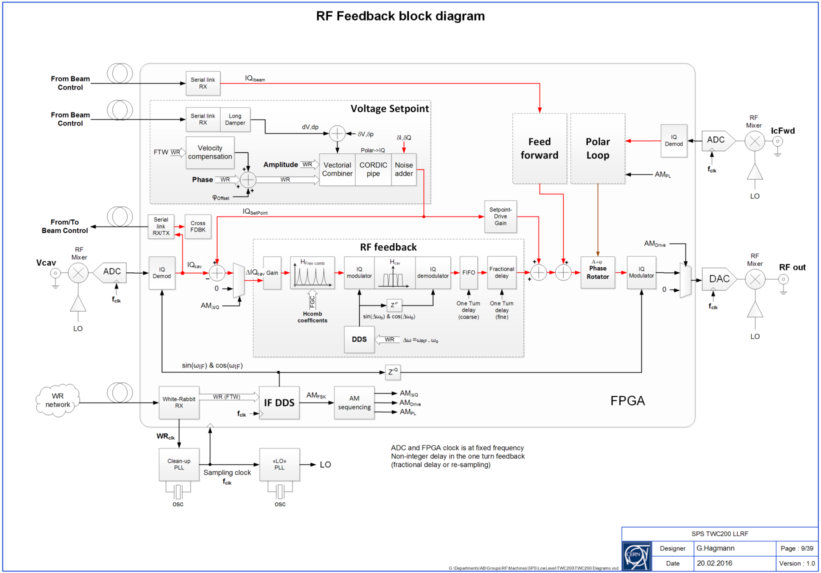
```
@papiprof(['CPI', 'MEM_BOUND', 'CORE_BOUND'])
def foo():
    compute()

foo()
papiprof_report_metrics()
```

Simple Use-Case

ISCAS

- Submission deadline: 16 Oct
- Notification of acceptance: 15 Jan
- Size: 4 pages
- Contents:
  - Short introduction of BLonD + how BLonD utilizes signal theory and DSP.
  - HPC for BLonD
    - hotspots, scalability analysis (other metrics?)
    - Optimizations: Compiled C kernels, multi-threading, vectorization, other code optimizations
  - GPU analysis (Deadline too soon?)
  - Or Maybe focus only on a signal processing module?
    - Work-in-progress

Diagram Cavity Loop SPS TWC200



Thank you for your attention