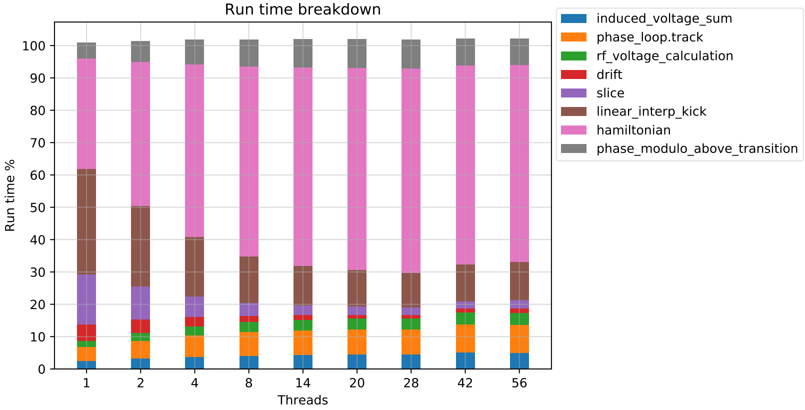
BLonD Meeting

Konstantinos Iliakis

August 18, 2017

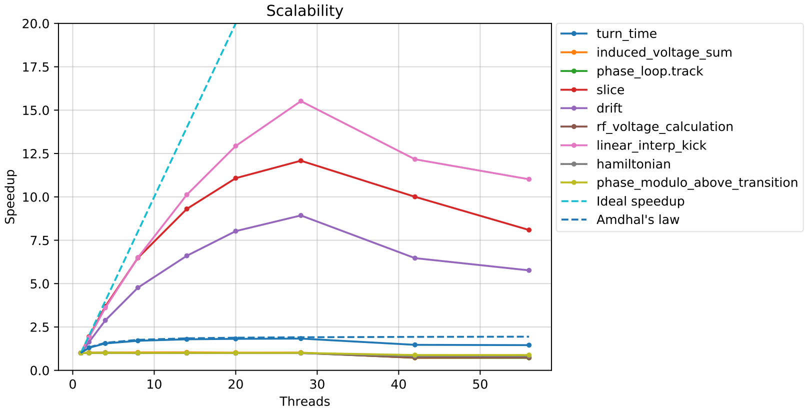
Table of Contents

Runtime Breakdown



- Only 49% parallel part
- `hamiltonian()` dominates the runtime (called 1/10 turns)

Scalability

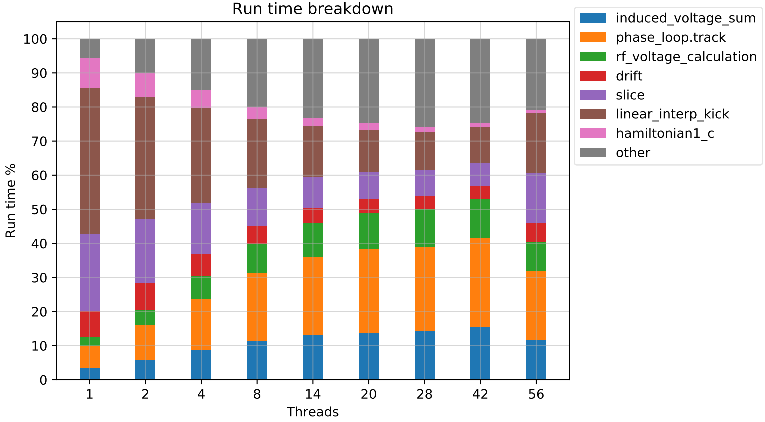


- Adequate scalability up-to 8/14 threads for `kick()`, `histo()`
- `drift()` again seems problematic
- 1.96x theoretical peak speedup

Improved Hamiltonian

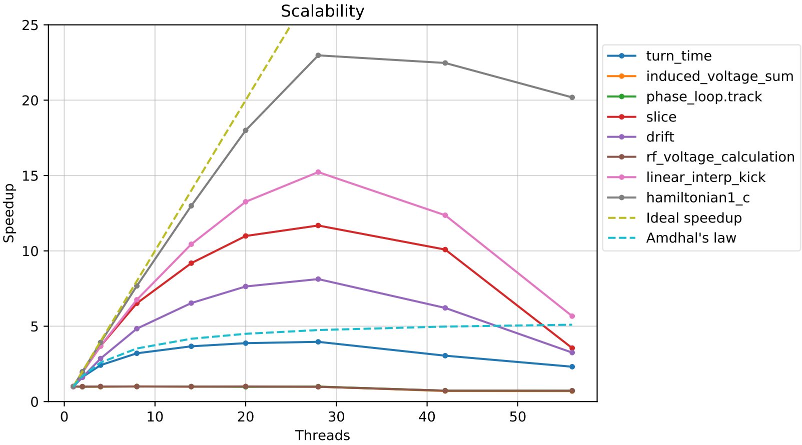
- Translated in C `phase_modulo_below/above_transition()` Go to listing
- and the `hamiltonian()` return expression Go to listing
- `hamiltonian()` now runs 5.6x (1 thread) – 23.8x (28 threads) faster

Runtime Breakdown



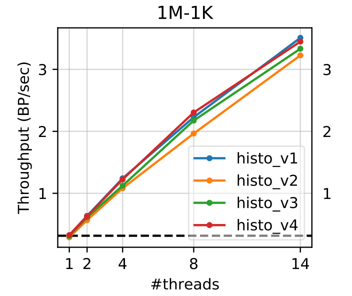
- 82% parallel part
- The problem has been moved to `induced_voltage_sum()`, `phase_loop.track()` and `rf_voltage_calculation()`

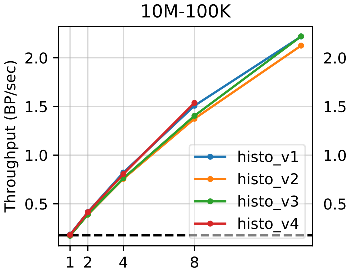
Scalability



- `hamiltonian1_c()` scales well up-to 28 threads
- 5.5x theoretical peak speedup

Parallel Histogram





- histo_v1: current version, static mem, single allocation, upper limit for #slices
- histo_v2: dynamic mem, 1D array, allocate/free on every call
- **histo_v3**: dynamic mem, 2D array, allocate/free on every call
- histo_v4: static mem, 2D array, allocate on every call, no upper limit for #slices, seg fault when too many slices
- histo_v0: dashed line, serial histogram

On-going work

- The bottlenecks in both LHC and SPS testcases are
  - `rf_voltage_calculation()`
  - `induced_voltage_sum()`
  - `phase_loop.track() (LHC only)`
- Profile the PSB test-case (Danilo's main file)
- `linear_interp_kick()` cuda implementation is ready but not benchmarked yet

Thank you for your attention

Phase modulo listing

```
extern "C" void
phase_modulo_above_transition(double *phi,
                              const int size)
{
    const double two_pi = 2.0 * M_PI;

    #pragma omp parallel for
    for (int i = 0; i < size; i++)
        phi[i] = phi[i] - two_pi *
                          floor(phi[i]/two_pi);
}
```

Phase modulo below/above transition function

Back to Back to presentation

Hamiltonian listing

```
extern "C" void
hamiltonian1(const double *dE, const double *phi_b,
             double *result, const double c1,
             const double c2, const double phi_s,
             const int size)
{
    const double sin_phi_s = fast_sin(phi_s);
    const double cos_phi_s = fast_cos(phi_s);

    #pragma omp parallel for
    for (int i = 0; i < size; i++)
        result[i] = c1 * dE[i] * dE[i]
                      + c2 * (fast_cos(phi_b[i]) - sin_phi_s)
                      + (phi_b[i] - phi_s) * sin_phi_s;
}
```

hamiltonian return expression

Back to Back to presentation