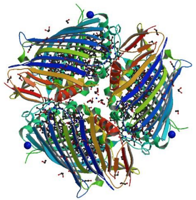
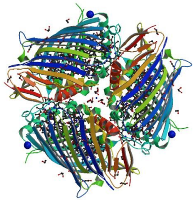
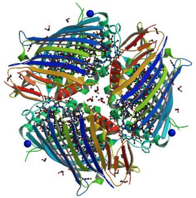






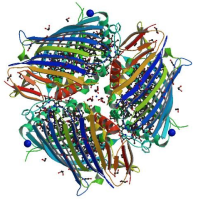
Implementation and Parallelization of Redfield Equations

CS 205 Final Project

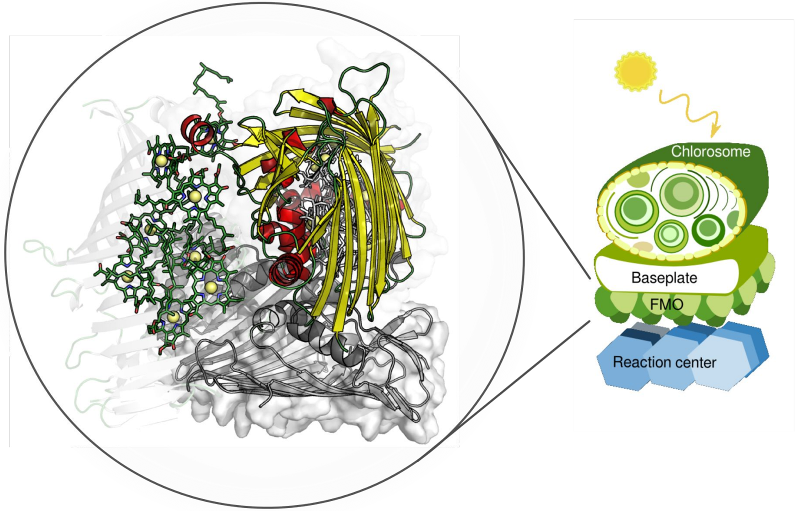
Florian Häse
Sukin Sim
Teresa Tamayo



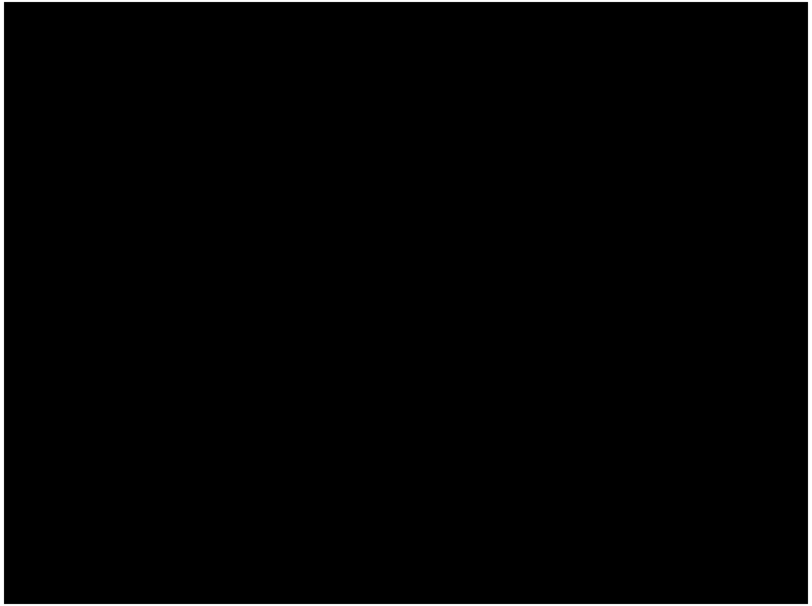



Background: Exciton Transfer

- Important for improving efficiency of solar cells
- Current methods limited by:
    - small number of excitonic sites (<30) → Scaling: $O(N^6)$, N = number of sites
    - short time scales
- Redfield Equation



David Glowacki



Redfield Equation

Bottleneck: matrix-matrix multiplications

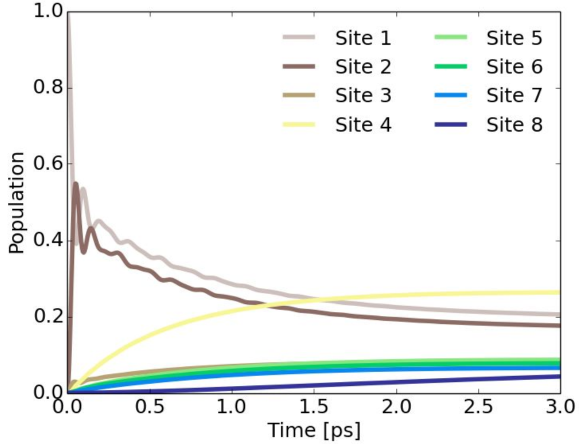
$$\frac{d\rho(t)}{dt} = \boxed{-\frac{i}{\hbar}[H,\rho(t)]} + \boxed{\sum_{k,m,n} \gamma(\omega_{mn})\left(V_k(\omega_{mn})\rho(t)V_k^\dagger(\omega_{mn}) - \frac{1}{2}V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\rho(t) - \frac{1}{2}\rho(t)V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\right)}$$

System contribution

Contribution from coupling between system and environment

Dimension: N sites + loss state + target state = N+2 total states

- Dynamics of excited states described by time evolution of density matrix
- Current methods do not scale well beyond systems with > 30 excitonic states.
- Can we do better using different parallelization models? (Long term goal: run a full-scale simulation of large systems)

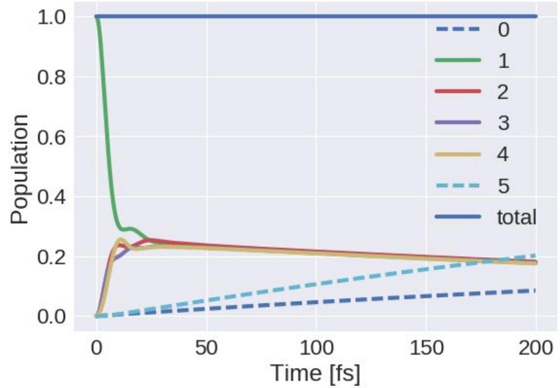


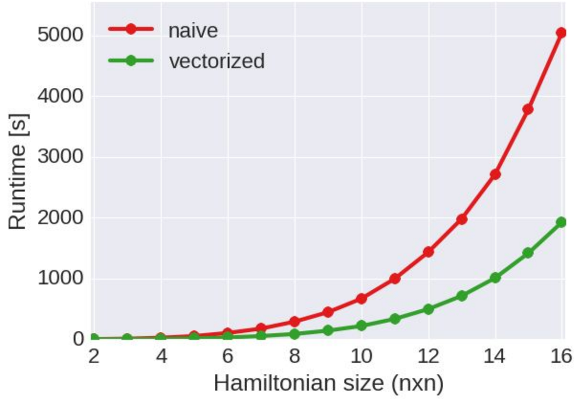
Approaching the problem: Python implementation

1. Identified bottleneck (Lindblad term)

2. Preprocessing the problem ...

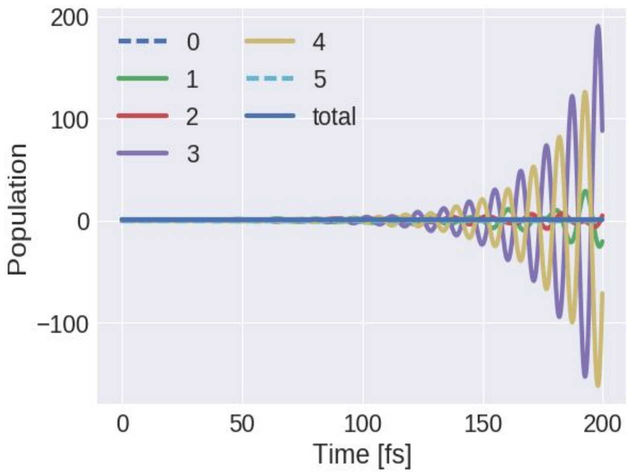
- Naive:
  - H, V, ρ complex-valued
- Vectorized:
  - H, V real-valued (can ignore complex phase)
  - H diagonalized to further reduce computational cost
  - V and γ independent of current time step so precomputed and stored in memory for later usage



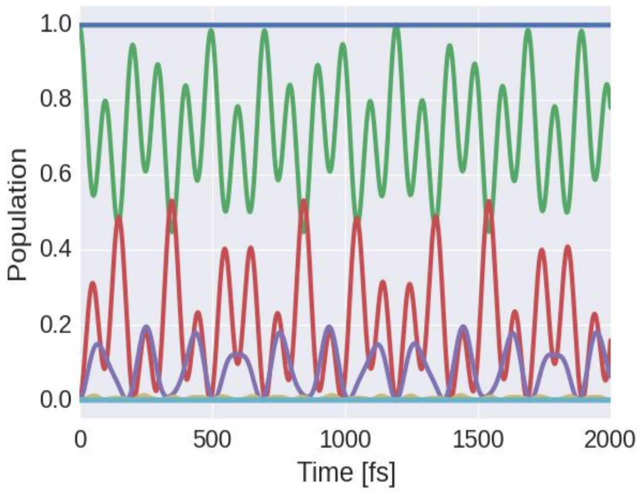


Increasing Efficiency and Accuracy

- Efficiency: translate code to C to further improve runtimes
- Accuracy: Runge-Kutta 4 (vs. Euler)



Euler integration (Python implementation)



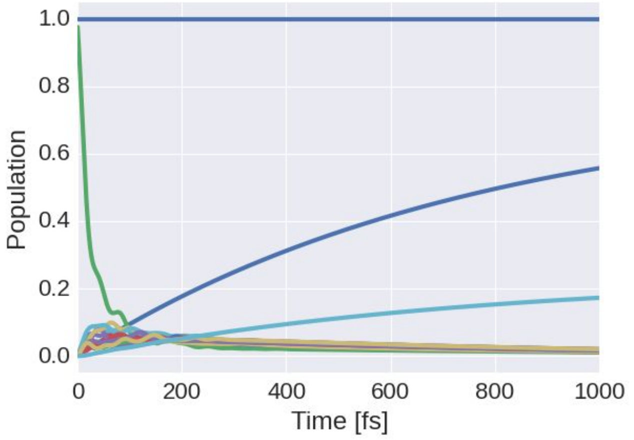
4th order Runge Kutta integration (C implementation)

Computing Redfield in SIMT Model: OpenACC on Tesla K80

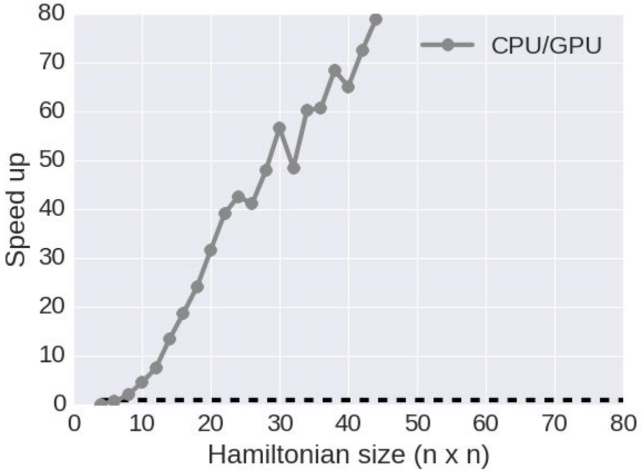
Features:

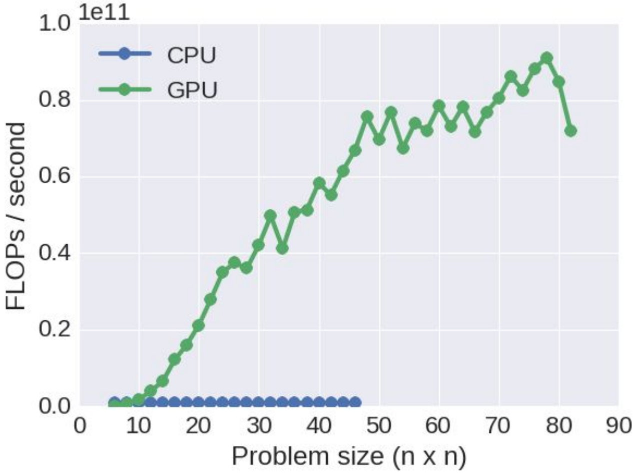
- Blocked matrix matrix multiplications
- Reducing the algorithm to the bare essentials

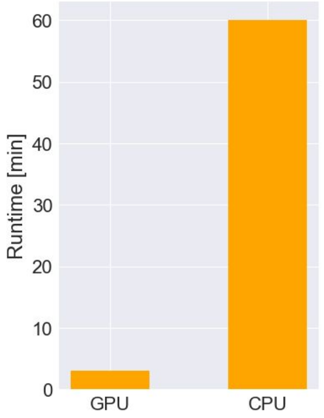
$$\frac{d\rho(t)}{dt} = \boxed{-\frac{i}{\hbar}[H,\rho(t)]} + \boxed{\sum_{k,m,n} \gamma(\omega_{mn})\Big(V_k(\omega_{mn})\rho(t)V_k^\dagger(\omega_{mn}) - \frac{1}{2}V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\rho(t) - \frac{1}{2}\rho(t)V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\Big)}$$



16 site population dynamics:





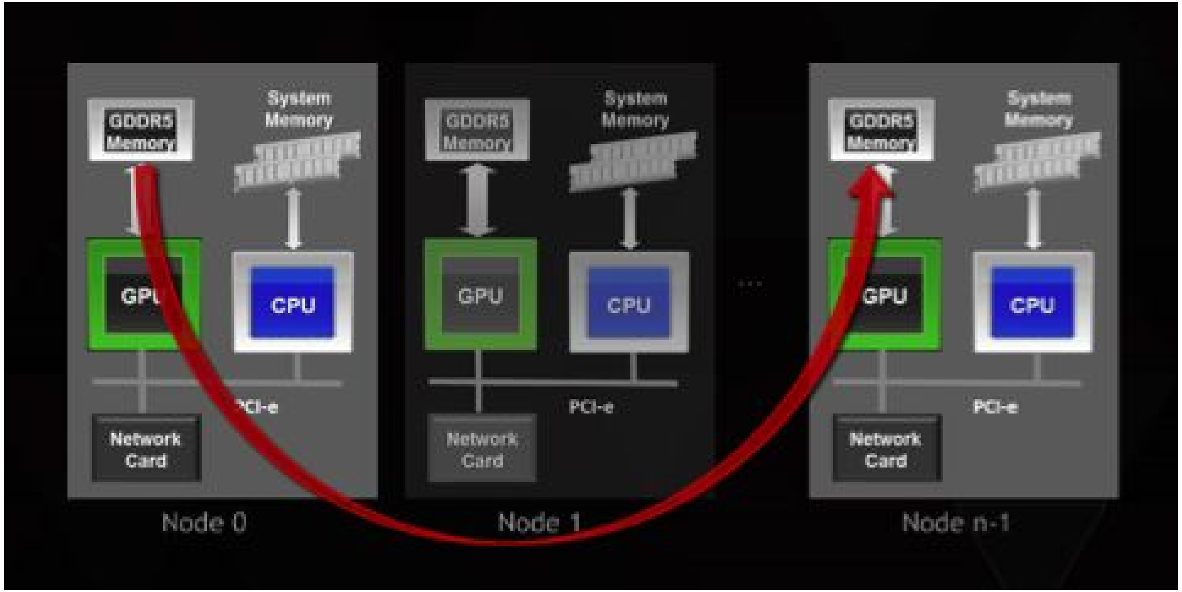


Parallel Implementation: MPI + OpenAcc

$$\frac{d\rho(t)}{dt} = -\frac{i}{\hbar}[H,\rho(t)] + \boxed{\sum_{k,m,n}\gamma(\omega_{mn})\Big(V_k(\omega_{mn})\rho(t)V_k^\dagger(\omega_{mn}) - \frac{1}{2}V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\rho(t) - \frac{1}{2}\rho(t)V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\Big)}$$

Contribution from coupling between system and environment

- Current status:
  - Memory Overheads
  - Corruptions in memory



Advanced Feature: Stronger Scaling via OpenMP

$$\frac{d\rho(t)}{dt} = \boxed{-\frac{i}{\hbar}[H,\rho(t)]} + \boxed{\sum_{k,m,n} \gamma(\omega_{mn})\left(V_k(\omega_{mn})\rho(t)V_k^\dagger(\omega_{mn}) - \frac{1}{2}V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\rho(t) - \frac{1}{2}\rho(t)V_k^\dagger(\omega_{mn})V_k(\omega_{mn})\right)}$$

- Bottleneck: matrix-matrix multiplication operations in Lindblad term
  - $15 * N^3$ operations but Runge-Kutta implementation so 4 of these sets per time step
- Approaches: blocking + multithreading
- Better speedups with more threads
- Better scaling as system size increases