Faculty of Mathematics and Natural Sciences
University of Bergen

Master Thesis in Physics

Modelling Laser Hydrogen Interactions with a GPU Accelerated Propagator

Konstantin Rygol

May 20, 2020

Under the supervision of
Prof. Morten Førre
at the Department of Physics and Technology
University of Bergen

Abstract

This thesis describes the development of the GPU accelerated propagator (GAP). It allows to simulate laser atom interactions. The design goal of GAP is to minimize communication across the cluster. GAP is designed for large clusters and runs efficiently on CPU and on GPU accelerated nodes. At run time different parallel libraries can be selected to ensure an optimal performance on the used hardware. The computations were conducted in the Amazon cloud. The highest speed up of the GPU version over a CPU version achieved was 7.4 for a single GPU problem and 4.4 for a multi GPU problem. GAP was then used in a study of the excitation and ionization of a circular hydrogen Rydberg state, the 5g (m=4) state, by a 800 nm pulse. The rule of beyond dipole corrections for the laser matter interactions was investigated and the concept of atomic stabilization was revisited.

Contents

List of Figures

List of Tables

Acknowledgements

I want to thank my supervisor Morten Førre for offering me a place in his research group and having an open ear for my questions. Special thanks goes to the atomic physics group for inspiring discussions. I would like to thank my friend Carsten for great outdoor trips and great tea breaks. Furthermore I would like to thank my parents for their support and for stimulating my scientific curiosity throughout my childhood.
I would also like to thank the AWS Cloud Credits for Research program for proving the funding for the calculations in the cloud.

1 Introduction

1.1 The fundamentals of atomic physics

Atomic physics has been a research field since ancient times. The Greek philosophers were arguing on the continuity of matter. They thought that there would have to be a indivisible particle that would be the smallest unit. They called it atomos meaning "uncutable" [1]. Atoms are the smallest unit of ordinary matter that constitute a chemical element. Every solid, gas and liquid consists of atoms. Atoms consist of electrons with negative charge and the same number of protons with positive charge. All atoms besides hydrogen have neutrons which are charge less. The charge of a proton and an electron is called an elementary charge and its existence has been proven by Robert A. Millikan in 1909 [2]. All electronic charges are expressed in increments of the elementary charge. Atoms without external interaction are stable systems that can be influenced by external forces and fields. The simplest atom is the hydrogen atom. It consists of a proton and an electron. Its simplicity makes it the best candidate to study the fundamental questions of quantum mechanics.

1.2 The concept of light

In 1665 Robert Hooke developed a pulse theory to describe the origin of colors [3]. He suggested that light vibrates perpendicular to the propagation direction. Christian Huygens then formalized this in a mathematical wave theory of light in 1678. This theory was well suited to explain interference and diffraction. A medium called ether was assumed but later disproven by the Michelson Morley experiment. In 1845 Michael Faraday discovered that the polarization of light can be changed due to electromagnetic effects [4]. In 1873 James Maxwell then published a full mathematical description of the behavior of electric and magnetic fields known as the Maxwell's equations [5]. In quantum theory photons are then

seen as wave packets which allow the description of effects such as spectral lines.

1.3 Time dependent Schrödinger equation

When an atom interacts with light the nucleus is often assumed to be stationary. The electron's movement is then described by the wave function ψ. The Schrödinger equation is a linear partial differential equation that describes the ψ of a quantum mechanical system. The Schrödinger equation for a free particle is,

$$i\hbar\frac{d}{dt}\psi = -\frac{\hbar^2}{2m}\nabla^2\psi \tag{1}$$

The Hamiltonian of a particle is defined by the sum of kinetic energies and the potential energies,

$$H = T + V = -\frac{\hbar^2}{2m}\nabla^2 + V \tag{2}$$

With the Hamiltonian the Time Dependent Schrödinger Equation (TDSE) can be formulated,

$$i\hbar\frac{d}{dt}\psi = H\psi \tag{3}$$

This equation describes the time dependent movement of a particle in a potential. In general the TDSE has to be solved numerically. As research progresses the problems to be studied become more demanding on the computing hardware. It is therefore important to use modern computers efficiently to obtain results in a reasonable time.

1.4 Micro chip development

Parallel computing is a field in computational science. It allows us to use modern electronic chips efficiently. To see why it became necessary to program in parallel, a closer look at the development of micro chips over the last decades is taken.

Moore's law

Moore's law states that the number of transistors on a chip doubles every two years. It was formulated in 1965 by Gordon Moore. Since then the growth has slowed down but the overall exponential character is still valid. In figure 1 we can see the development of CPUs until 2017. While the number of transistors still grows exponentially, the single thread performance has reached a limit. Around the year

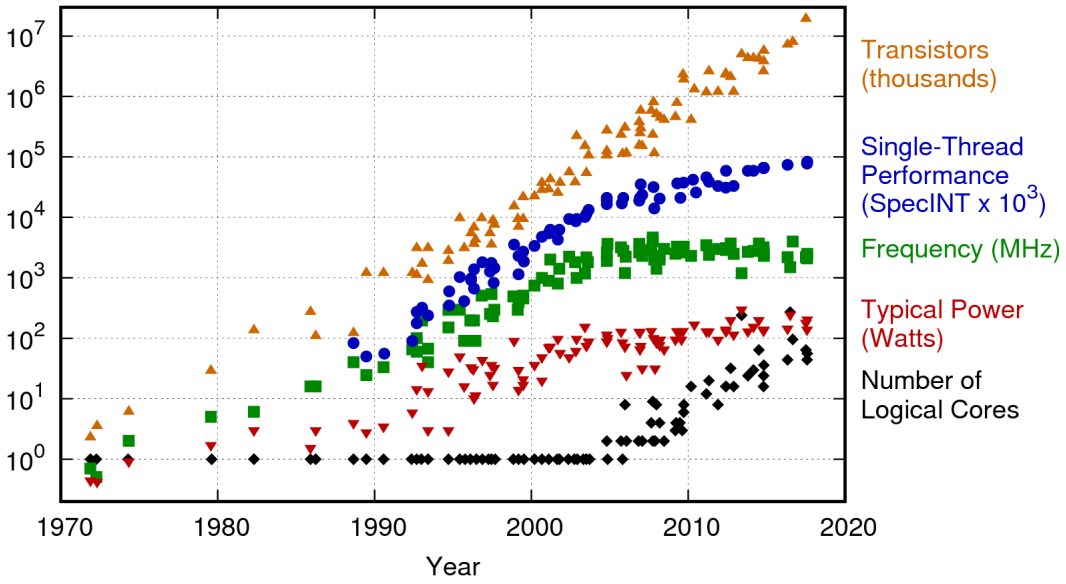


Figure 1: Moore's laws shown for CPUs over a span of 42 years. Figure obtained from [6].

2005 the growth of the clock speed has slowed down. At the same time multi core processors emerged. Instead of using the growing number of transistors on one core they get distributed over multiple cores. They started of as dual and quad cores growing to up to a 100 logical cores in recent years. To use the chip efficiently programs that run efficiently on multiple cores have to be developed. Depending on the framework the programmer has to manage data and task distribution.

Power consumption and clock speed

As a scientist it is easy to forget about economic and environmental impact of our calculations. However it is important to look at the energy consumption of our hardware as it is the main environmental impact and a large cost driver of our calculations. The power consumption of an electronic chip is split up in dynamic and static power consumption [7]. The dynamic power consumption follows the equation,

$$P_{dyn} \simeq V^2 f \tag{4}$$

with f the clock speed and V the voltage of the chip. The highest frequency at which a chip can operate is roughly proportional to the voltage. Then the power

dependency becomes,

$$\mathcal{O}(P_{dyn}) = f^3 \tag{5}$$

This relation explains that although modern electronic chips can run at clock speeds above 5 GHz it is rarely seen in scientific computing as the power consumption per instruction is too high. Therefore programs that can make use of multiple cores at a lower frequency are the preferred way of programming.

1.5 Scope of the thesis

This thesis shows the development of a cross platform high performance computing (HPC) simulation suite. The simulation suite is then used for a study of multi photon ionization of the circular 5g (m=4) state of hydrogen by a 800 nm Ti-sapphire laser pulse.
The developed GPU Accelerated Propagator (GAP) is a scalable multi node application. It can run on CPU or GPU on optimized platforms and can adapt to hardware constraints. GAP uses a multi GPU accelerated Lanczos propagator. Single GPU Lanczos propagators have been implemented before [8], but multi GPU Lanczos propagators have not been published. The development of GAP was therefore pioneering work made possible by high level GPU programming and the advancements in GPU-GPU communication protocols. The scalability of GAP is of high interest and a performance study is conducted in chapter 3. In the fourth chapter the results of the 800 nm study are described. Atomic stabilization was found. Beyond dipole effects played a minor role.

2 Background

2.1 Notation and Conventions

Atomic units

The study of atomic physics like many other fields of physics uses their own set of units the atomic units (a.u.). The reason behind using atomic units is to simplify calculations contrary to calculations in the SI units. The conventions used most commonly are listed in table 1. The constants listed in table 1 with value *one* will now be neglected in future equations.

Dirac notation

In quantum mechanics the Dirac notation or bra-ket notation is an often used notation for quantum states. [9] A ket, $|\rangle$ denotes a vector of a vector space V and a bra, $\langle|$ denotes a linear functional on V. In this thesis we assume that V is a Hilbert space. For the vector space $\mathbb{C}^n$, kets can be identified as column vectors and bras as row vectors. Operators can be identified as linear operators through the approximation property [10]. Combinations of kets, bras and operators are evaluated using matrix and vector multiplications.

Symbol	Name	Value in SI	Value in a.u.
$\hbar$	reduced Planck constant	$1.054571 \times 10^{-34} Js$	1
m_e	electron rest mass	$9.109383 \times 10^{-31} kg$	1
k_e	coulomb's constant	$8.987551 \times 10^{9} \frac{Nm^2}{C^2}$	1
c	speed of light	$2.99792 \times 10^{8} \frac{m}{s}$	137.036
e	elementary charge	$1.602176 \times 10^{-19} C$	1
a_0	Bohr radius	$5.291772 \times 10^{-11} m$	1

Table 1: Table of atomic units.

Linear operators acting on kets Linear operators can act on a ket and return another ket,

$$|\phi\rangle = A\,|\psi\rangle \tag{6}$$

For a linear operator A and $|\psi\rangle\,, |\phi\rangle \in V$. Using the description of $|\psi\rangle$ as a $1 \times N$ column vector in a N-dimensional Hilbert space and A as a matrix with the dimension $N \times N$. Then A acting on the ket is computed by the matrix vector product $A\,|\psi\rangle$.

Linear operators acting on bras Operators can also work on bras from the right side. Analogous to kets if a linear operator A works on a bra the result is a bra,

$$(\langle\phi|\,A)\,|\psi\rangle = \langle\phi|\,(A\,|\psi\rangle) \equiv \langle\phi|A|\psi\rangle \tag{7}$$

For calculations the matrix vector representations are used and in the case of equation (7) a scalar element is obtained.

Physical Observable	Symbol of operator	Hermitian operator
Momentum	$\hat{p}$	$-i\nabla$
Position	$\hat{r}$	r
Kinetic energy	$\hat{T}$	$-\frac{1}{2m}\nabla^2$
Potential energy	$\hat{V}$	$V(r)$
Hamiltonian	$\hat{H} = \hat{T} + \hat{V}$	$-\frac{1}{2m}\nabla^2 + V(r)$
Angular momentum	$\hat{L}_j$	$(\hat{r} \times \hat{p})_j = -i(r \times \nabla)_j$

Table 2: Physical observables and their Hermitian operators.

2.2 Quantum Mechanics

2.2.1 Postulates of Quantum Mechanics

The postulates of quantum mechanics give a deep understanding on the fundamentals of quantum mechanics. These principles can be formulated in various equivalent ways. This thesis uses a similar formulation to [11].

First Postulate

At each instant the state of a physical system is described by the wave function, a ket $|\psi\rangle$ in the space of states. This implies that the superposition of two states is again a state of the system,

$$|\psi\rangle = a_1 |\psi_1\rangle + a_2 |\psi_2\rangle \tag{8}$$

The vector space has an inner product,

$$\langle\psi|\phi\rangle = \int_V dV \psi^*(V)\phi(V) \tag{9}$$

Second Postulate

To each observable attribute of a physical system a corresponding Hermitian operator exists that acts on the wave function describing the system. Table 2 shows commonly used observables and their operators.

Third Postulate

The only possible result of a measurement of an observable A is one of the eigenvalues of its corresponding operator $\hat{A}$,

$$\hat{A}\psi = \lambda\psi \tag{10}$$

As values from measurements are real numbers, the eigenvalues corresponding to an observable are real. This implies that the operators are Hermitian. Eigenstates of Hermitian operators are orthogonal if the corresponding eigenvalues are distinct,

$$\langle a_j | a_k \rangle = \beta\delta_{jk} \tag{11}$$

The set of eigenstates span the space of possible states of the wave function. As they are orthogonal, they form an orthogonal basis.

Fourth Postulate

When a measurement of an observable A is made on the wave function, the probability of obtaining an eigenvalue α_n is given by the square of the inner product of the wave function with eigenstate $|a_n\rangle$,

$$p_{a_n} = \langle a_n | \psi \rangle^2 \tag{12}$$

The states are assumed to be normalized to one. With $\beta = 1$ in equation (11) the basis becomes an orthonormal basis.
If a measurement of an observable A is made on the wave function the average value of A is given by,

$$< A >= \int_V dV \psi^\dagger \hat{A} \psi \tag{13}$$

Fifth Postulate

Immediately after the measurement of an observable A has yielded α_n, the state of the system is the normalized eigenstate $|a_n\rangle$. This is also known as the collapse of the wave function. It is motivated through experiments. The fourth postulate states that multiple measurements for systems with the same set up follow a statistical distribution. It has been shown that if a system has yielded the result α_n and is immediately remeasured it yields the same result. Thereby the two immediate measurements of the system are not statistically distributed but coupled.

Sixth Postulate

The time evolution of the wave function preserves the normalization of the wave function. The time evolution of the wave function can be described by,

$$|\psi(t)\rangle = U(\hat{t},t_0)\,|\psi(t_0)\rangle \tag{14}$$

for a unitary operator $\hat{U}$. This postulate makes $|\psi\rangle$ follow a differential of the form,

$$i\hbar\frac{d}{dt}\,|\psi(t)\rangle = H\,|\psi(t)\rangle\,, \tag{15}$$

with H being the Hamiltonian. This is the time dependent Schrödinger equation.

2.2.2 Light

Light can be described by the Maxwell's equations. In free space they read,

$$\begin{aligned} \Delta \cdot \boldsymbol{E} &= \frac{\rho}{\epsilon_0} \\ \Delta \cdot \boldsymbol{B} &= 0 \\ \Delta \times \boldsymbol{E} &= \frac{\partial \boldsymbol{B}}{\partial t} \\ \Delta \times \boldsymbol{B} &= \mu_0(\boldsymbol{J_0} + \epsilon_0\frac{\partial \boldsymbol{E}}{\partial t}) \end{aligned} \tag{16}$$

with $\boldsymbol{E}$ the electric field vector, $\boldsymbol{B}$ the magnetic field vector and $\boldsymbol{J}$ the current vector. ρ is the charge density, ϵ_0 and μ_0 are the permittivity and permeability of free space. Maxwell's equations can be expressed by a scalar potential ϕ and a vector potential $\boldsymbol{A}$. They follow the relations,

$$\begin{aligned} B &= \nabla \times \boldsymbol{A} \\ E &= -\nabla\phi + \frac{\partial \boldsymbol{A}}{\partial t} \end{aligned} \tag{17}$$

These potentials are not uniquely determined. Any transformation of the potentials,

$$\begin{aligned} \boldsymbol{A} &\rightarrow \boldsymbol{A'} + \nabla\lambda \\ \phi &\rightarrow \phi' - \frac{\partial\lambda}{\partial t} \end{aligned} \tag{18}$$

with λ a scalar function, leaves the physical fields unchanged. This property is called gauge freedom. A common choice is the coulomb gauge used in this thesis,

$$\nabla \cdot \boldsymbol{A} = 0 \tag{19}$$

In this thesis the laser pulse used is,

$$\boldsymbol{A} = \frac{\boldsymbol{E_0}}{\omega} \sin\left(\frac{\pi t}{T}\right)^2 \sin(\omega t) \tag{20}$$

with E_0 the maximum electrical field, T the pulse duration and ω the angular frequency. E_0 can be obtained through the intensity of the laser with the relation,

$$E_0 = 5.338\sqrt{I_0}10^{-9} \tag{21}$$

2.2.3 The Model

The often used dipole approximation is based on the assumption that the dimensions of the system interacting with the laser are much smaller than the wavelength of the laser. This allows to ignore the spatial dependency of the laser field. The laser field is assumed to be only variant in time with the same magnitude across the system and no magnetic field component. In the standard non relativistic approach the evolution of the wave function ψ in a laser field $\boldsymbol{A}$ and a Coulomb potential V is described by the time dependent Schrödinger equation (TDSE),

$$i\hbar\frac{\partial}{\partial t}\psi = H\psi \tag{22}$$

In the minimal coupling picture the Hamiltonian is,

$$H = \frac{p^2}{2m} + V + \frac{q}{m}\boldsymbol{A}(r,t)p + \frac{q^2}{2m}\boldsymbol{A}(r,t)^2 \tag{23}$$

The Hamiltonian can be reformulated as [12],

$$H = \frac{p^2}{2} + V + \frac{q}{m}\boldsymbol{A}(r,t)\cdot\boldsymbol{p} + \frac{q^2}{2cm^2}\{\boldsymbol{A}(r,t)^2,\hat{\boldsymbol{k}}\cdot\boldsymbol{p}\} \tag{24}$$

where the unit vector $\hat{k}$ describes the direction of the laser propagation, c the speed of light and the curly brackets denote the anti commutator $\{a,b\} = ab + ba$. It is called the propagation gauge formulation [12]. Neglecting the spatial dependence of the vector potential the final form of the Hamiltonian used in this thesis is derived,

$$H = \frac{p^2}{2m} + V + \frac{q}{m}\boldsymbol{A}(t)\cdot\boldsymbol{p} + \frac{q^2}{2cm^2}A(t)^2\boldsymbol{k}\cdot\boldsymbol{p}, \tag{25}$$

the Hamiltonian for the Schrödinger equation that has to be simulated in time. The term linear in $\boldsymbol{A}$ in equation (25) describes the dipole effects, the term quadratic in $\boldsymbol{A}$ the beyond dipole effects including the main effect of the magnetic field.

2.3 Time propagating the TDSE

The time evolution of the Schrödinger equation for a hydrogen atom in a laser field is the main objective of this thesis. In this thesis the approach of choice will be to formulate the Hamiltonian in a specific basis. Afterwards a Lanczos propagator is used to execute the time evolution stated in the sixth-postulate. The physical problem can be formulated in a variety of basis. In this thesis the eigen state basis is chosen. The eigen state basis is generated in the B-spline basis. The description in the B-spline basis changes the eigen value problem, that the Schrödinger equation is, to a generalized eigenvalue problem.

2.3.1 B-splines

B-splines are piece wise polynomial functions that are non zero only on a finite interval. For a deep analysis on B-splines in atomic physics refer to [13].
A set of B-splines of order k over an interval is constructed for a sequence of knots $\{t_0,...,t_{n-1}\}$ sorted in non decreasing order. For a given order k, the i-th B-spline is computed from the two lower order B-splines,

$$B_{i,k+1}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k}(x) \tag{26}$$

For the first two orders the following relation is used,

$$B_{i,k}(x) = \begin{cases} 1, & \text{if } t_i \geq x \geq t_{i+1} \\ 0, & \text{else} \end{cases} \tag{27}$$

The derivatives of B-splines can be calculated analytically using the following relation,

$$\frac{d}{dr} B_i^k(r) = \frac{k-1}{t_{i+k-1} - t_i} B_i^{k-1}(r) + \frac{k-1}{t_{i+k} - t_{i+1}} B_{i+1}^{k-1}(r) \tag{28}$$

The key property of B-splines is that they and their derivatives are continuous.

2.3.2 Eigen state basis

The eigenstates of the hydrogen Hamiltonian can be written as the product of a radial wave function and a spherical harmonic. The reduced radial function is denoted as,

$$u_{kl}(r) = r R_{kl}(r) \tag{29}$$

with the azimuthal quantum number l and the wave number k. The wave number k replaces the n quantum number as an index for ionized states. Spherical harmonics are defined as,

$$Y_l^m(\theta,\phi) = \Delta_m \sqrt{\frac{2l+1(l-|m|)!}{4\pi(l+|m|)!}} P_l^{|m|}(cos(\theta)) e^{im\phi} \tag{30}$$

with $P_l^{|m|}$ an associated Legendre polynomial, the magnetic quantum number m, the spherical coordinates θ, ϕ and the Condon Schortley phase convention,

$$\Delta_m = \begin{cases} 1, & \text{if } m \leq 0 \\ (-1)^m, & \text{if } m > 0 \end{cases} \tag{31}$$

The eigenstates are then defined as,

$$\psi_{nlm}(\boldsymbol{r}) = \frac{u(r)_{nl}}{r} Y_l^m(\theta,\phi) \tag{32}$$

2.3.3 Basis transformation into B-splines

In the following section the transformation of an eigenvalue problem to a generalized eigenvalue problem in a different basis is shown. This is used to express the eigenstates in B-splines.
The time independent Schrödinger equation is an eigenvalue equation. If the eigenvalues are expanded in a B-spline basis, the eigenvalue problem becomes a generalized eigenvalue problem. The eigenvalue problem is of the form,

$$H\,|\psi\rangle = E\,|\psi\rangle \tag{33}$$

ψ the set of eigen vectors and E the set of eigenvalues. It is then possible to substitute ψ by a linear combination of basis functions,

$$|\psi\rangle = \sum_{i=1}^{\infty} c_i B_i Y_{lm} \tag{34}$$

In the computational practice the number of basis functions has to be restricted to a finite number N. This yields,

$$|\psi\rangle = \sum_{i=1}^{N} c_i B_i Y_{lm} \tag{35}$$

In the new basis the eigenvalue problem can be formulated as,

$$\sum_{n=1}^{N} c_i H B_i Y_{lm} = E \sum_{i=1}^{N} c_i B_i Y_{lm} \tag{36}$$

Multiplication with $B_j Y_{lm}$ and integration over the volume V yields,

$$\sum_{n=1}^{N} c_n \int_V dV B_j Y_{lm} H B_i Y_{lm} = \lambda \sum_{i=1}^{N} c_i \int_V dV B_j Y_{lm} B_i Y_{lm} \tag{37}$$

In bra ket notation the equation transforms to,

$$\sum_{i=1}^{N} c_i \langle B_j Y_{lm}| H |B_i Y_{lm}\rangle = \lambda \sum_{i=1}^{N} c_i \langle B_j Y_{lm}|B_i Y_{lm}\rangle \tag{38}$$

Lets denote,

$$H_{ji} = \langle B_j Y_{lm}| H |B_i Y_{lm}\rangle \tag{39}$$

$$S_{ji} = \langle B_j Y_{lm}|B_i Y_{lm}\rangle \tag{40}$$

Substituting in equation (38) yields,

$$\sum_{n=1}^{N} c_i H_{ji} = E \sum_{n=1}^{N} c_i S_{ji} \tag{41}$$

This system can then be written in matrix representation,

$$\begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1N} \\ H_{21} & H_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ H_{N1} & H_{N2} & \cdots & H_{NN} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \lambda \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1N} \\ S_{21} & S_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ S_{N1} & S_{N2} & \cdots & S_{NN} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix},$$

which then becomes the generalized eigenvalue problem,

$$Hc = ESc \tag{42}$$

Solving this generalized eigenvalue problem yields the eigenstates.

2.3.4 The Hamiltonian

As stated in the section model 2.2.3 the Hamiltonian consists of the kinetic energy, the radial potential energy operator and the interaction terms. In the absence of a laser field it supports bound states which have negative energy and unbound

states with positive energies. If the laser field is active only unbound states are supported. The energies obtained in equation (42) are the diagonal elements of the Hamiltonian. The off diagonal elements are represented by different momentum operators. In this thesis the operators p_z and p_x are needed to represent the Hamiltonian stated in (25).

The p_z operator

The transition probability for the momentum operator for z-polarization from a state ψ_{klm} to a state $\psi_{k'l'm'}$ is derived in the following section,

$$\begin{aligned}\langle\psi_{klm}|p_z|\psi_{k'l'm'}\rangle &= -i\,\langle R_{kl}(r)Y_{lm}(\theta,\phi)|\frac{\partial}{\partial z}|R_{k'l'}(r)Y_{l'm'}(\theta,\phi)\rangle \\ &= -i\left\langle \frac{u_{kl}(r)}{r}Y_{lm}(\theta,\phi)\middle|\frac{\partial}{\partial z}\middle|\frac{u_{kl}(r)}{r}Y_{l'm'}(\theta,\phi)\right\rangle\end{aligned} \tag{43}$$

In spherical coordinates,

$$\frac{\partial}{\partial z} = cos\theta\frac{\partial}{\partial r} - \frac{1}{r}sin\theta\frac{\partial}{\partial\theta} \tag{44}$$

In the paper [14] by Simonsen and Førre the following relations are stated,

$$\begin{aligned} cos\theta Y_{lm}(\theta,\phi) &= b_{l+1,m}Y_{l+1,m}(\theta,\phi) + b_{l,m}Y_{l-1,m}(\theta,\phi) \\ sin\theta\frac{\partial}{\partial\theta}Y_{lm}(\theta,\phi) &= b_{l+1,m}Y_{l+1,m}(\theta,\phi) - (l+1)b_{lm}Y_{l-1,m}(\theta,\phi) \\ b_{lm} &= \sqrt{\frac{l^2-m^2}{(2l-1)(2l+1)}} = \sqrt{\frac{l^2-m^2}{4l^2-1}}\end{aligned} \tag{45}$$

This yields,

$$\begin{aligned}\frac{\partial}{\partial z}\frac{u_{kl}(r)}{r}Y_{l'm'}(\theta,\phi) &= (cos\theta\frac{\partial}{\partial r} - \frac{1}{r}sin\theta)\frac{u_{kl}(r)}{r}Y_{l'm'}(\theta,\phi) \\ = (\frac{1}{r}\frac{du_{k'l'}}{dr} - \frac{l'+1}{r^2}u_{k'l'})b_{l'+1,m'}Y_{l'+1,m'} &+ (\frac{1}{r}\frac{du_{k'l'}}{dr} + \frac{l'}{r^2}u_{k'l'})b_{l',m'}Y_{l'-1,m'}\end{aligned} \tag{46}$$

Inserting in equation (43) delivers,

$$\begin{aligned}
&\langle\psi_{klm}|p_z|\psi_{k'l'm'}\rangle \\
&\qquad = -i\left\langle \frac{u_{kl}(r)}{r}Y_{lm}(\theta,\phi)\right| \Big(\frac{1}{r}\frac{du_{k'l'}}{dr} - \frac{l'+1}{r^2}u_{k'l'}\Big)b_{l'+1,m'}Y_{l'+1,m'} \\
&\qquad + \Big(\frac{1}{r}\frac{du_{k'l'}}{dr} + \frac{l'}{r^2}u_{k'l'}\Big)b_{l',m'}Y_{l'-1,m'}\Big\rangle \\
&\qquad = -ib_{lm}\delta_{l',l-1}\delta m',m\int_0^\infty u_{kl}\Big(\frac{du_{k',l-1}}{dr} - l\frac{du_{k',l-1}}{r}\Big)dr \\
&\qquad - ib_{l+1,m}\delta_{l',l+1}\delta m',m\int_0^\infty u_{kl}\Big(\frac{du_{k',l+1}}{dr} + (l+1)\frac{du_{k',l+1}}{r}\Big)dr
\end{aligned} \tag{47}$$

This is now a matrix element of the Hamiltonian. This calculation has to be done for every combination of k,l and m that the Hamiltonian should represent.

The p_x operator

The transition probability for the momentum operator for x-polarization from a state ψ_{klm} to a state $\psi_{k'l'm'}$ can be developed in an analogous fashion,

$$\begin{aligned}
\langle\psi_{klm}|p_x|\psi_{k'l'm'}\rangle &= -i\,\langle R_{kl}(r)Y_{lm}(\theta,\phi)|\frac{\partial}{\partial z}|R_{k'l'}(r)Y_{l'm'}(\theta,\phi)\rangle \\
&= ia_{lm}\delta_{l',l-1}\delta_{m',m-1}\int_0^\infty u_{kl}(\frac{du_{k',l-1}}{dr} - l\frac{u_{k',l-1}}{r})dr \\
&- ia_{l,-m}\delta_{l',l-1}\delta_{m',m+1}\int_0^\infty u_{kl}(\frac{du_{k',l-1}}{dr} - l\frac{u_{k',l-1}}{r})dr \\
&- ia_{l+1,-m+1}\delta_{l',l+1}\delta_{m',m-1}\int_0^\infty u_{kl}(\frac{du_{k',l+1}}{dr} + (l+1)\frac{u_{k',l+1}}{r})dr \\
&+ ia_{l+1,m+1}\delta_{l',l+1}\delta_{m',m+1}\int_0^\infty u_{kl}(\frac{du_{k',l+1}}{dr} + (l+1)\frac{u_{k',l+1}}{r})dr
\end{aligned} \tag{48}$$

with,

$$a_{lm} = \sqrt{\frac{(l+m)(l+m-1)}{16l^2-4}} \tag{49}$$

The Hamiltonian obtained is a sparse matrix. This is important for the implementation of the needed matrix vector operations.

2.3.5 Modern eigenvalue solvers

The simplest way to obtain the eigenvalues of a very small matrix is through the characteristic polynomial. For matrices with a greater dimension than four the

Abel-Ruffini theorem states that no algorithm can solve the eigenvalue problem within finite time using only elementary steps. As the Hamiltonians we study have a much larger dimension iterative algorithms that approximate the eigenvalues are the choice. In physics quite often not all eigenvalues of a matrix are needed but its most significant ones. A common approach for iterative algorithms are the Krylov subspace methods. In the following sections a short introduction of the Krylov space and the power iteration is given, leading to the formulation of the Lanczos algorithm used in the thesis.

Krylov space

The order-r Krylov subspace generated by a $n \times n$ matrix A and a vector b of dimension n is the linear space spanned by the images of b under the first r powers of A [15]:

$$K_r(A,b) = span\{b,Ab,A^2b,...,A^{r-1}b\} \tag{50}$$

Power iteration

For many modern sparse matrix eigenvalue solvers Krylov solvers play an important role. One method to generate the Krylov space is the method of "Power Iteration". Given a diagonalizable matrix A the algorithm will produce a number λ, which is the eigenvalue of greatest magnitude and a non zero vector v, which is a corresponding eigen vector to the eigenvalue λ.

$$Av = \lambda v \tag{51}$$

The most time consuming operation of the algorithm is the matrix vector product making the method applicable to large sparse systems.

Lanczos Algorithm

The Lanczos algorithm is a direct algorithm by Cornelius Lanczos [16] that is a variant of the power methods to find the k most significant eigenvalues and eigen vectors. Tending towards the lowest or highest eigenvalues of an $n \times n$ matrix where k is chosen to be less or equal to n. The dimension of the Krylov space is chosen to be approximately 1.5 times the number of desired eigenvalues.

The source code of the Lanczos algorithm used in GAP is,

```
"""
Input:
A: m x m matrix
y: initial vector
dimKryl: dimension of the Krylov space
m: dimension of the matrix A
Output:
Q: orthogonal Krylov space
h: tridiagonal matrix
"""

Q.col(0) = y
for i in range(dimKryl):
    b = A*y
    d[i] = y.adjoint() * b
    #reorthogonalization
    b = b - Q.col(0:i) * Q.col(0:i).adjoint() * b
    b = b - Q.col(0:i) * Q.col(0:i).adjoint() * b
    if i < dimKryl-1:
        od[i] = b.norm()
        if od[i] !=0:
            y = b/od[i]
            Q.col(i+1) = y

for i in range(dimKryl-1):
    h[i+1,i] = od[i]
    h[i,i+1] = od[i]

for i in range(dimKryl):
    h[i,i] = d[i]

return Q,h
```

2.3.6 Lanczos propagator

After the Hamiltonian is obtained now the method to propagate it is described. The Lanczos propagator works well for sparse Hamiltonians [17].
In postulate 6 in 2.2.1 the time evolution of the Schrodinger Equation was defined with the help of a unitary operator. The evolution operator is expressed by,

$$U(t,t+\Delta) = e^{-H(t)\Delta t} \tag{52}$$

In the Lanczos propagator the Lanczos algorithm is used to build up a suitable sub space. ψ is then propagated in this subspace. The Lanczos algorithm used in this thesis returns the tridiagonal matrix h and the orthonormal basis Q of the Krylov space. The eigenvalues of h are then calculated with a direct eigenvalue algorithm. In the following they will be referred to as w_j. The eigen vectors obtained are denoted as P_j. Q_j refers to the j-th column of the matrix of the basis vectors. The unitary time evolution operator in the subspace Q is then defined as:

$$U^{(Q)} = e^{-iH^{(Q)}\Delta t} \tag{53}$$

$$U^{(Q)} = \sum_j \langle P_j | \, exp(-iw_j\Delta t) \, |P_j\rangle \tag{54}$$

Inserting in equation (14) yields,

$$|\psi(t+\Delta t)\rangle = \sum_j |\psi_0\rangle \, \langle P_j | \, exp(-iw_j\Delta t) \, |P_j\rangle \tag{55}$$

Written in matrix notation with w the diagonal matrix of the eigenvalues of the tridiagonal matrix h is obtained,

$$|\psi(t+\Delta t)\rangle = |\psi_0\rangle \, Pexp(-i\mathrm{w}\Delta t)P^\dagger \tag{56}$$

The system has been described in a polynomial expansion by applying the Hamiltonian repeatedly to the wave function. This is an efficient way to propagate the Schrödinger equation in time. As the majority of the time is needed for the matrix vector product, while the unitary propagator takes negligible time. Therefore the next pages study efficient matrix vector multiplication.

2.4 Computational methods

2.4.1 Sparse matrix formats

The memory used to store a dense $n \times n$ matrix is given by,

$$n^2 \cdot sizeof(\text{datatype}) \tag{57}$$

In our case the data type is a double. A double is an eight Byte floating point number. This yields a memory consumption of,

$$\text{Memory cost} = n^2 \cdot 8 \tag{58}$$

Unless specified otherwise all memory is given in Bytes in this thesis.
The number of non zero elements is abbreviated by nnz. Now if our matrix is sparse,

$$nnz << n^2 \tag{59}$$

the dense matrix format becomes inefficient to store the matrix. A key factor to describe a sparse matrix is the sparsity of the matrix. It is defined as the number of zero-valued elements divided by n^2. The following relation is commonly used:

$$sparsity = 1 - nnz/n^2 \tag{60}$$

If the sparsity of a matrix is greater than 0.5 it is considered a sparse matrix. The higher this number is the more efficient will a sparse matrix representation be.

Coordinate list (COO)

An intuitive format to store a sparse matrix is the Coordinate list format(COO). In this format the row index, the column index and the value for every non zero cell of the matrix is stored. The maximum value of the index data type has to be greater than the maximum dimension of the matrix. For the simulations conducted in this thesis 4 Byte integers are sufficient as indices. The memory cost per occupied matrix cell is:

$$\text{Memory cost matrix element} = \text{sizeof(RowIndex)+sizeof(ColIndex)+sizeof(Value)} \tag{61}$$

For 8-Byte values this is 16 Bytes per cell. The total memory cost for the matrix is therefore,

$$\text{Memory cost} = nnz \cdot 16 \tag{62}$$

This format does have the ability to save a substantial amount of memory for a sparse matrix. Every element of the matrix can be accessed directly through their Row and Col indices. The time for inserting or removing elements is $\mathcal{O}(1)$.

CSR-Format

The Compressed Row Format is an improvement on the COO format and allows to further compress the data. The array of row indices is hereby substituted by an array of size $n+2$. As typical $nnz >> n$ the size of the CompressedRowIndex array can be neglected in the memory analysis. This reduces not only the size of the matrix but often also the computation time of sparse matrix vector (SpMV) products, as these operations are often memory/cache bound. The memory cost shrinks down to,

$$\text{Memory cost} = nnz \cdot 12, \tag{63}$$

thus saving 25% memory. To demonstrate the advantage of the CSR format a small example is given. The first step to transform a COO matrix to a CSR matrix is to sort the matrix first by the row entries and then within a row by the column entries. The CSR matrix offers an efficient access to row wise operations. These are often used in matrix vector and matrix matrix products. The downside is that a column wise access needs to be done via an element wise access. Using binary search the element wise access time is $\mathcal{O}(n) = log(\text{Elements in row})$. This small example describes a square matrix with dimension five. It has eight entries. This yields a sparsity of $1 - 8/25 = 0.68$. It is therefore a sparse matrix and a sparse matrix format is beneficial.

$$\begin{bmatrix} 20 & 10 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 40 & 30 & 0 \\ 60 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 70 & 80 \end{bmatrix} \tag{64}$$

Value	20	10	50	40	30	60	70	80
Col	0	1	1	2	3	0	3	4
Row	0	0	1	2	2	3	4	4

Table 3: Sparse matrix in COO format.

Value	20	10	50	40	30	60	70	80
Col	0	1	1	2	3	0	3	4
RowPointer	0	2	3	5	6	8		

Table 4: Sparse matrix in CSR format.

For this small example the CSR matrix saved two row elements compared to the COO format. As the matrix enlarges in dimension, the size of RowPointer grows with n. For most sparse matrices n is lower than nnz.
The CSC format follows the same principle but instead of compressing the row indices, the col indices are compressed. It offers well performing column wise access. This can be useful in matrix matrix products or when the transposed of a matrix is frequently needed for matrix vector operations. It might be best for the run time of the SpMV to save A as a CSR matrix and A^T as a CSC matrix. The downside lies in an increased higher memory consumption. Both CSR and CSC are highly efficient formats when it comes to linear algebra operations and memory consumption. Their downsides are a worse element wise access than COO and an overhead constructing them, mainly spent on sorting the matrix. The sorting can also require temporary buffer memory. There are some more specialized sparse matrix vector formats that can bring advantages for certain sparsity patterns or hardware architectures. Those formats are beyond the scope of this thesis. CSR and COO are the most commonly used sparse matrix formats across the HPC community.

2.4.2 Parallel programming models

As parallel programming becomes a necessity for larger simulations. A closer look at different programming models and high performance computing (HPC) hardware is taken. There are two parallel programming models that will be discussed in this thesis. To understand them a further look at modern cluster architecture is needed.

2.4.3 Distributed systems

Most modern high performance computers consist of a large number of so called nodes. Every node holds at least one CPU, RAM and network access. In most

clusters all nodes are equal in their hardware. Because the main memory (RAM) of the system is distributed across the nodes these systems are called distributed systems. This kind of architecture does come with certain challenges. Every node needs to have a part of the problem to be solved in its memory. Quite often synchronization is needed across the nodes. The exchange of data between the nodes becomes a major source of performance bottlenecks for larger simulations. When analyzing cluster networks the term of cluster topology is used. It describes what kind of network the cluster has. Following two examples for cluster typologies are given. Figure 2 shows a fully connected network. This offers the highest performance as every node can communicate to every other node on a full capacity connection. The obvious downside is that this is not feasible for a greater number of nodes as the amount of connections grows super exponential. In figure 3 a bus approach is shown. This can be realized by connecting the nodes to a router. This drastically reduces the amount of connections. The downside is that the paths between different nodes might share the same connection. If for example node 2, 3 and 4 sent data to node 1 then the bottleneck is the connection between the router and node 1. The theoretical speed would drop by a factor of three compared to a fully connected cluster. Most modern HPC cluster have multiple network layers of different speeds. They can combine different network topologies trying to maximize performance while staying at a reasonable cost. Designing a good network with low latency, high throughput at a reasonable cost is the key factor for running scalable applications on a supercomputer.

2.4.4 Shared memory systems

Shared memory systems allow all the cores on the systems to access the same memory. This does allow easier programming as the problem does not have to be distributed over multiple nodes. The downside is that with an increasing system the communication cost between cores and the main memory increases. This is often due to an increase in physical distance but also due to inefficient memory access. Therefore it is a general recommendation to use shared memory parallelism within a node and distributed parallelism for inter node communication.

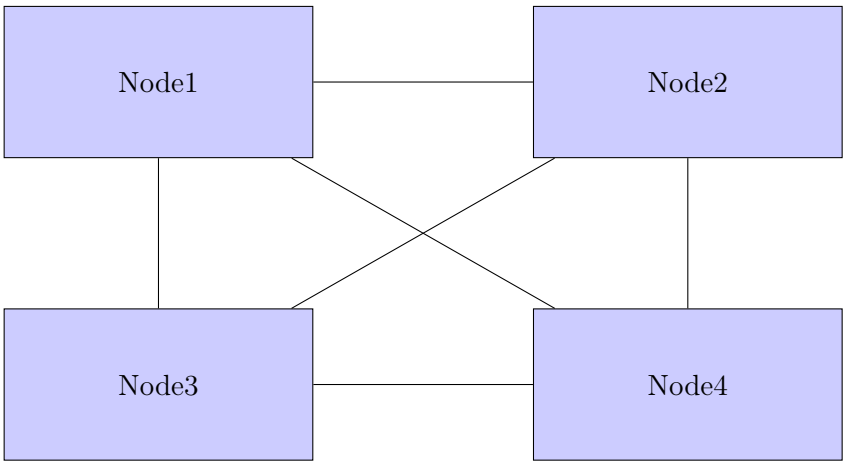


Figure 2: A cluster with 4 nodes as a fully connected network, all nodes are interconnected.

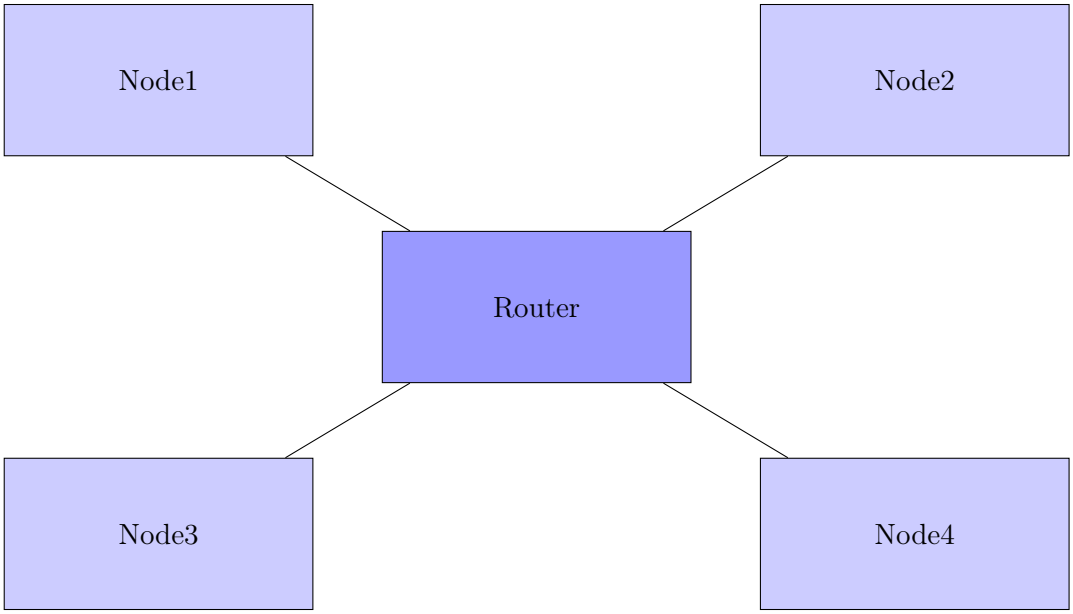
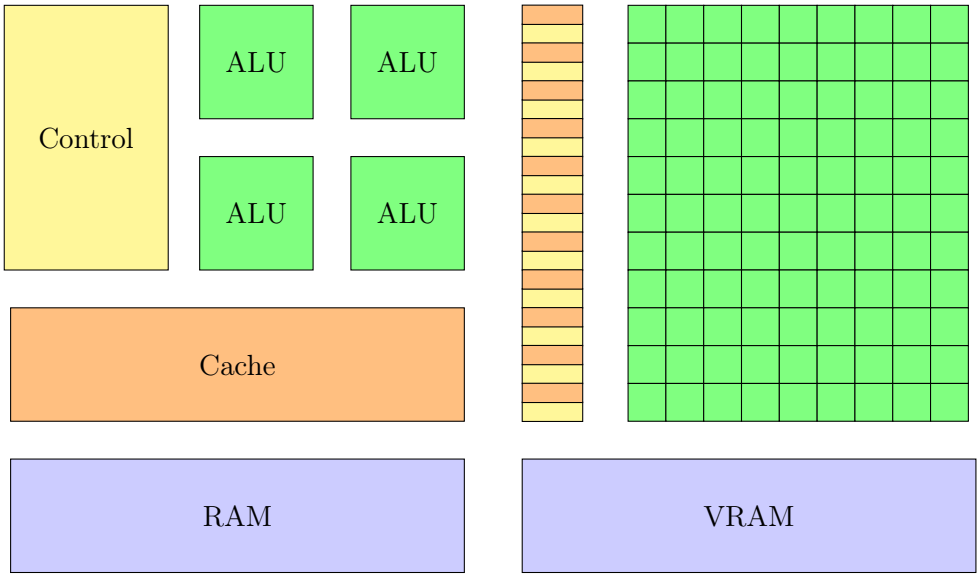


Figure 3: A cluster with 4 nodes as a bus connected network, all nodes are connected over a router.



(a) CPU (b) GPU

Figure 4: The difference in design between a GPU and CPU. The arithmetic logical unit (ALU) does the calculations. Control units are for data movement. Cache is a low latency, high throughput memory small in size.

2.4.5 GPUs

Graphics processing units (GPUs) were designed to create images for output to a display device. They have a highly parallel structure that makes them more efficient than CPUs for algorithms that can process large blocks of data in parallel. "The world's first GPU" was the GeForce 256 by Nvidia in 1999 [18]. To efficiently use a GPU the problem to be solved needs to express a very high level of parallelism. While the CPU uses the main memory to store data the GPU has its own memory. It is called the Video Random Access Memory (VRAM). Compared to the RAM of the CPU it has a higher throughput, in modern HPC GPUs up to 900 GB/s vs 10-50 GB/s of RAM. The downside is that the VRAM of GPUs is considerably smaller than RAM. For HPC GPUs it is typically in the range of 10-30 GB vs a 100-1000 GB for RAM. The GPU is then connected to the CPU via pci-express (pci-e). Nowadays pci-e of the fourth generation is the standard providing up to 15.7 GB/s if eight pci-e lanes are used.

2.4.6 A GPU node

In figure 5 the build up of traditional HPC GPU node is shown. A non GPU code is similar and might have more CPU cores or RAM. The main aspect of this figure is to illustrate the different speeds of the data transfer within a node. The throughput between VRAM and GPU is a factor of 12 higher than between RAM and the CPU. This is important for memory throughput bound operations. The second observation is that data transfer between RAM and VRAM is by a factor of 25 slower than the VRAM to GPU. This encourages data locality in the VRAM. The speed of the network connecting to other nodes is slower by another factor of 10. To develop fast and scalable GPU code data locality has to be a top priority. It is important to leave as much data as possible on the GPU. Try to avoid communication over the network even if it increases computational complexity.

2.4.7 Amazon elastic compute cloud(EC2)

The majority of the calculations in this thesis were conducted in the EC2. EC2 is a part of Amazon's cloud computing business that allows users to rent virtual machines and run their applications on them. The user can select from various Amazon machine images (AMI) and built their own software stack on top. Amazon and other third party vendors provide AMIs with preinstalled software and drivers. As there is no distinguished HPC AMI available, the machine learning AMI was used. It comes with GPU drivers and an implementation of MPI. An AMI can then be launched on a specific hardware configuration and is then called an instance. The hardware configuration of an instance can be changed after it has been shut down, leaving the integrity of the data intact. Amazon provides also a service called AWS ParallelCluster. It allows to launch multiple instances as a cluster similar to a traditional multi node HPC system. This is an interesting feature but was not used in this thesis. The instances used in this thesis are listed in table 5.

2.4.8 Parallel programming libraries

In the following paragraphs an overview over the parallel programming libraries used in this thesis is given. As the hardware architecture is quite different for CPUs and GPUs different programming libraries have to be used to utilize the hardware efficiently.

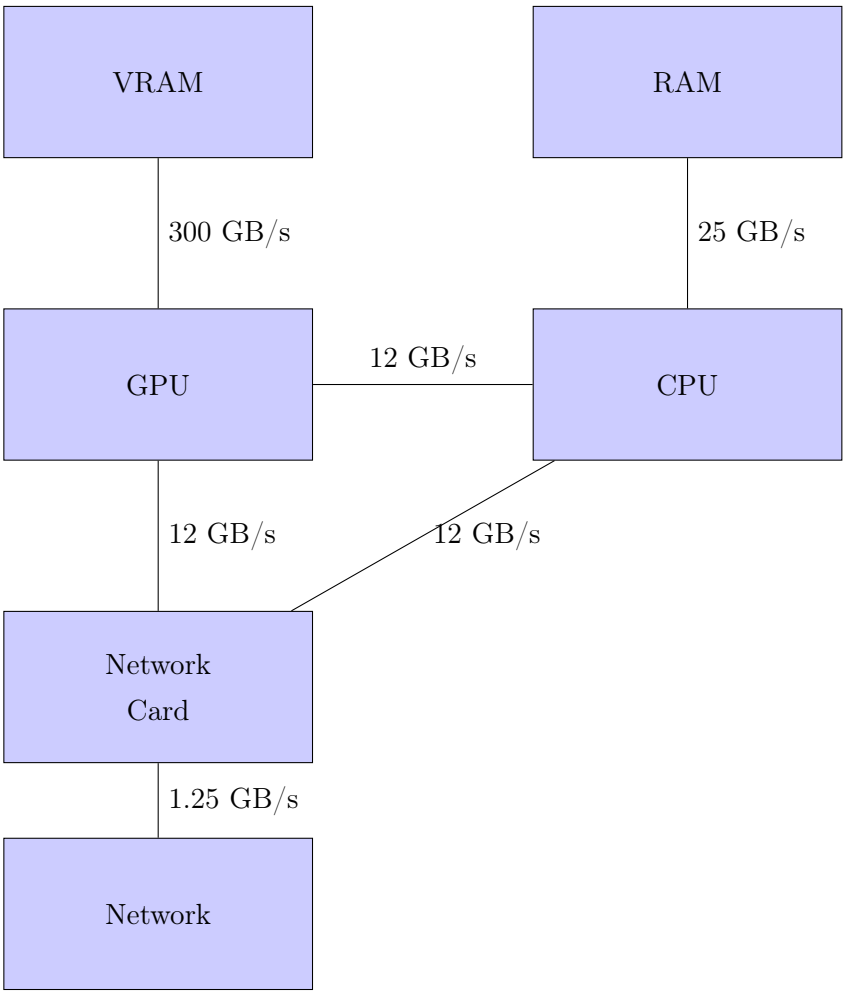


Figure 5: Hardware of a single GPU node and the speed of the connections between them. The connection speeds represent a p2.xlarge instance.

node	vCPU	RAM[GB]	GPUs	TFLOPS GPU	VRAM[GB]
ift40070	24	252	N/A	N/A	N/A
fram node	64	64	N/A	N/A	N/A
c5d.metal	96	192	N/A	N/A	N/A
p2.xlarge	4	61	1 K80	1.451	12
p2.8xlarge	32	488	8 K80	10.968	96
p2.16xlarge	64	768	16 K80	21.936	192
p3.2xlarge	32	61	1 V100	7.000	16
p3.8xlarge	64	244	4 V100	28.000	64

Table 5: Table of EC2 instances with their hardware configuration.

Message Passing Interface - MPI

The Message Passing Interface is the most common protocol in science and industry for distributed parallel programming. There are various libraries that implemented the MPI standard. In this thesis OPEN-MPI will be used. There are various HPC vendors that provide their own MPI implementation optimized to their systems.
MPI provides collective and point-to-point communication on so called communicators. Every MPI program starts with MPI_INIT() that initializes the communicator MPI_COMM_WORLD and spawns a specified number of ranks. Each of the ranks hold all the variables in their own private memory. Every rank executes the full program stated between MPI_INIT() and MPI_Finalize(). They can exchange data with MPI communication routines.
The programmer has to keep track of the different states of the same variable on multiple ranks. As every call to an MPI library is involved with a constant overhead the general approach is to keep most small variables redundant across the ranks. This might cost memory but allows for a cleaner program and often a better performance. If one of these variables change proper synchronization across the ranks has to be taken care of by the programmer.

openMP

openMP (Open Multi processing) supports multi platform shared memory multiprocessing programming in C, C++ and FORTRAN. It consists of a set of compiler

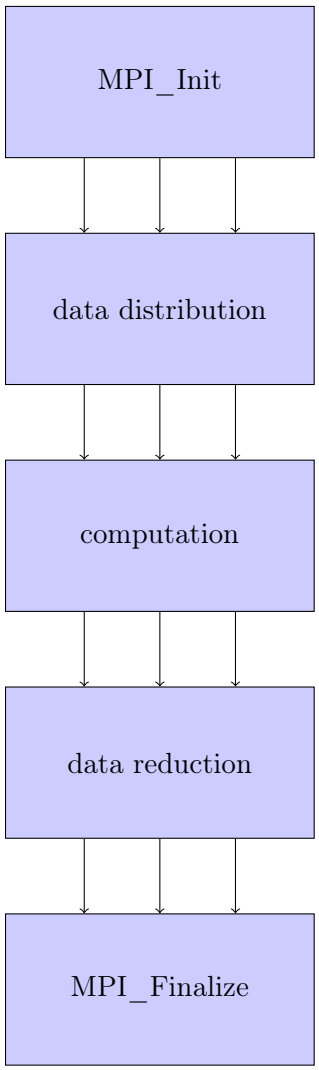


Figure 6: A simple MPI program with three ranks. The program is started with MPI_Init then data gets distributed or generated across the ranks. Tasks are performed on the data possibly involving synchronization. With MPI_Finalize all ranks but rank 0 get shut down.

directives, library routines and environment variables that influence run time behavior. OpenMP is a portable and scalable model that allows programmers to speed up code with simple compiler directives. It is used from desktop computers up to supercomputers.

The core idea behind openMP's multi threading model is a master thread that can spawn a specified number of worker threads. The master thread then divides the tasks among them. After a parallel segment is executed the worker threads join back to the master thread which continuous in a sequential manner. By default every thread executes the parallel section independently. Work sharing constructs can be used to divide the work in the parallel region among the threads. With openMP4.0 GPU offloading was introduced into the openMP standard. It is possible to target an accelerator device with the openMP target constructs. This approach has been tested on the Lanczos propagator. The results where part of a talk given at Super Computing 19 (SC19) and can be found in appendix .1. The performance depended highly on the compiler used with the clang compiler yielding lower run times than the gcc compiler. The main difficulty to achieve good run time lies within the nature of the GPU. A GPU is a highly specialized architecture that may vary drastically between generations and models in their capabilities and bottlenecks. The compiler has to try to deliver the best code with limited information given by the compiler directives. For now this optimization problem seems to be too great for the compilers. Therefore this approach is not used in this thesis.

Math Kernel Library - MKL

The Intel Math Kernel is a library that contains optimized math routines used in science and industry. Core math functions include BLAS, LAPACK and sparse solvers. As the library is published by the manufacture Intel the library is optimized for Intel processors. The MKL reaches a high performance on CPUs and has been an industry standard for performance for a long time. There are developments towards a cross platform approach called Intel oneAPI. This allows one code to be executed on multiple platforms.
As the MKL is proprietary software licensing can be an issue. As of now the MKL can be obtained for free.

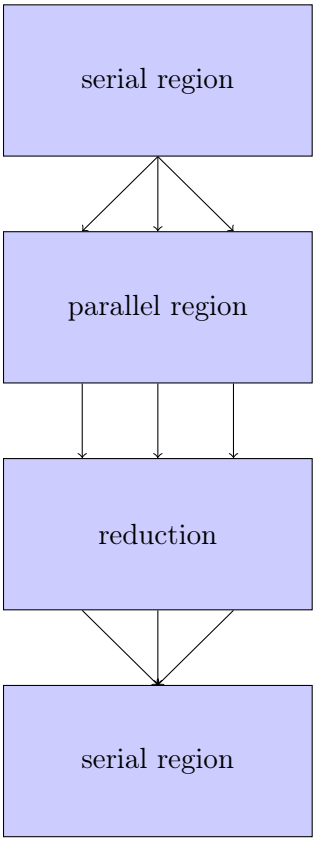


Figure 7: A program accelerated by openMP is a sequential program with parallel sections. When a parallel region is entered worker threads are spawned. At the end of a parallel region a reduction is common. The program continues in a sequential fashion until the next parallel region.

CUDA - cuBlas,cuSparse,cuSolver

CUDA (Compute Unified Device Architecture) is a parallel application computing interface(API) created by Nvidia. It allows programmers to use a CUDA enabled GPU for general purpose processing. This is called GPGPU (General-Purpose computing on GPUs). The CUDA platform is designed to work with the programming languages C/C++ and FORTRAN. The programs written in CUDA are called kernels. As GPUs are specialized chips the kernels are written in a low level fashion with a keen eye to memory management and reduction clauses. These kernels are often optimized for a certain architecture of graphic cards. When a different GPU with a different architecture is used adaptions to the code have to be made by the programmer to ensure optimal performance. Writing performant, elegant and stable codes for GPUs has proven to be a challenge that requires a skilled CUDA programmer. As physicists traditionally lack the time to become a skilled CUDA programmer a higher level approach than implementing directly in CUDA is to be preferred. Nvidia also publishes the numerical GPU libraries cuBlas, cuSparse and cuSolver. GAP was programmed by subsequently replacing C++ CPU optimized functions with the according GPU accelerated functions.

Multi GPU code

As few of the modern physics application fit inside a singe GPU it becomes necessary to build multi GPU code. An elegant solution is to use MPI to distribute the data over multiple GPUs. If MPI is used the code works also on multi node systems. Every GPU is assigned one MPI rank. It used to be necessary to copy code from the device memory to the host memory followed by the MPI call. With the introduction of CUDA aware MPI in 2013 it is now possible to call MPI function directly on device memory. This simplifies the program and allows the data to bypass the intermediate copy to host memory and thus speeding up MPI data movements significantly. To install a CUDA aware MPI the MPI library has to be compiled from source with a given CUDA library.

3 GPU Accelerated Propagator

This chapter describes the development of the GPU Accelerated Propagator (GAP) that was then used for a study in atomic physics. This chapter concludes with a performance study of GAP.

3.1 Design of GAP

GAP implements the physical model that was developed earlier. It takes a set of input parameters that can be either constraints or starting parameters for the simulation. GAP can be divided up in four blocks. These blocks resemble the steps described earlier. First a set of B-splines is generated. Then an eigenstate basis is formed. The Hamiltonian is then built in this eigenstate basis. The time evolution is then done with the Lanczos propagator. An overview over these blocks is given in figure 8. The first column resembles the blocks. The second block describes the operations that are the computational constraints for the performance. The third column describes the run time in $\mathcal{O}$ notation of the block. In the fourth column the overall constraint of the block is listed. It becomes clear that the generation of B-splines are performance-wise not an important calculation. The calculation of the eigen state basis and the construction of the sparse Hamiltonian depend, with the fourth power, on the number of break points (N_b). If the simulation does need a large number of break points because it allows for a high energy or uses a large box this can become a bottleneck. For the simulations in this thesis the construction of the eigen state basis and the sparse Hamiltonian were not a performance bottleneck. The performance-wise most demanding computational block is the Lanczos propagator. Therefore most of the GPU acceleration was used in this block.

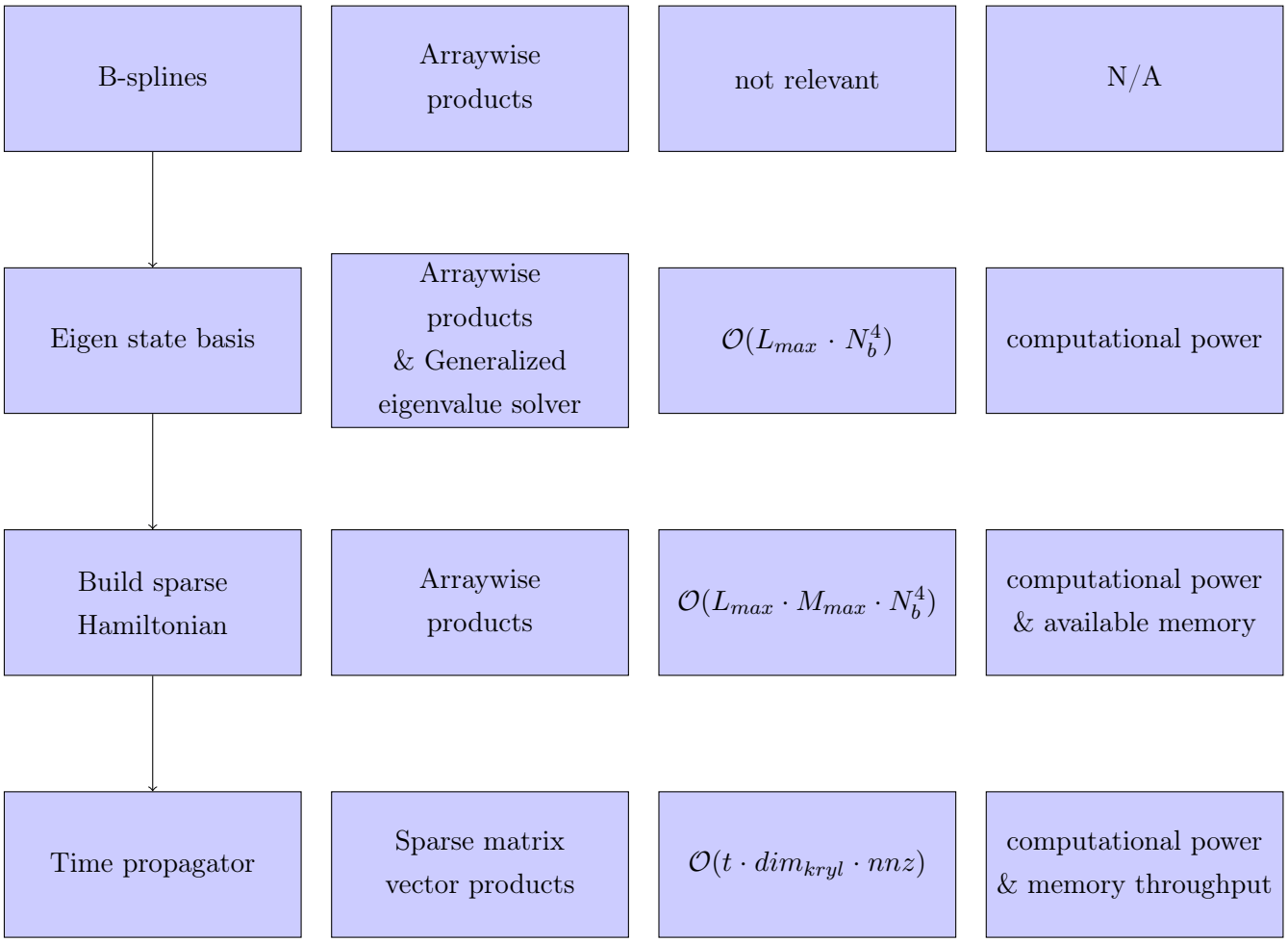


Figure 8: On the left side are the different functionality blocks that constitute GAP. The second column of blocks represents the operations that limit the performance. The third column gives the run time for the block. The last column describes the limiting hardware for the block.

3.1.1 Input parameters

GAP takes a variety of input parameters. Most describe the physical simulation. The other parameters help to optimize the computational performance. In table 7 a list of the input parameters of GAP and their descriptions are given. In the third column of the table the influence of the input parameter on the performance is given. Every simulation is constrained by the hardware it runs on. Two of the most dominant constrains are the overall run time and the maximum memory usage. The maximum memory usage is mainly dominated by the nnz. For the performance study parameters that do not affect the computational performance of GAP will be ignored. To achieve a simulation that converges similar input parameters have to be adjusted with respect to one another. The run time of the Lanczos propagator which is the computational-wise most demanding part of GAP is proportional to nnz, dim_kryl and the number of timesteps.

3.1.2 General thoughts on the implementation

GAP is implemented in C++. A major problem with software are inconsistencies between documentation and code. It is therefore desirable to represent the physics as directly as possible in the code. Pure C++ can not achieve that. Therefore GAP does make extensive use of the Eigen library. The Eigen library is a templated header only C++ library that allows to represent linear algebra operations in an almost pseudo code fashion. It has an outstanding performance and is widely used in the HPC community. It can make use of an openMP and a MKL back end. Code redundancies should be avoided whenever possible but the highest focus should be on the readability of the code. Extensive use of functions is therefore encouraged. Object orientated programming can be of great help but it should not overpower the code. GAP uses a class for the handling of the sparse matrix and one to store the input parameters.

3.2 Parallelization of GAP

While designing GAP there were two major challenges to tackle, the need for more memory and a faster computational speed. More memory means either using nodes with more memory or using more nodes and distributing the problem efficiently. Often the hardware to be used is given and limited in memory. Therefore the

name	data type	description	affect performance
nnz	int64_t	maximum number of nnz of Hamiltonian	yes
om	double	photon energy	no
I0	double	intensity of laser	no
T_cycle	int	number of optical cycles	no
t	int	number of time steps	yes
b	double	box size	yes
N_b	int	number of breakpoints	yes
L_min	int	minimum l-quantum number	yes
L_max	int	maximum l-quantum number	yes
M_min	int	minimum m-quantum number	yes
M_max	int	maximum m-quantum number	yes
E_max	double	highest energy state included	yes
n	int	order of b-splines	no
zComp	bool	if true then include zComp	no
xComp	bool	if true then include xComp	no
zft	int	if 1 polarization, if 2 propagation	yes
xft	int	if 1 polarization, if 2 propagation	yes
l_quantum	int	l quantum number of initial state	no
m_quantum	int	m quantum number of initial state	no
n_quantum	int	n quantum number of initial state	no
dim_kryl	int	dimension of the Krylov space	yes
Accelerator	string	accelerator [COO,MKL,GPU]	yes

Table 6: Table of Input parameters for GAP.

parameter	increase in N_b	increase in nnz
b	α	α^2
E_max	$> \alpha$	$> \alpha^2$
L_max	1	α if delta m = 0 α,else α^2
M_max	1	α^2

Table 7: Scaling of matrix size depending on input parameters. If a parameter is scaled with α the scaling of N_b and the resulting increase in nnz is shown.

programmer has to find the best multi node distribution of the problem. The best distribution of the problem does not only incorporate the most efficient way but also the simplest/cleanest and therefore most readable way. To parallelize across multiple nodes MPI is used. Within one node CPU accelerated approaches or a GPU accelerated approach is used. The CPU accelerated approaches make use of the openMP or the MKL library. The openMP approach uses a self written COO matrix vector product in the Lanczos propagator. The MKL version uses a CSR matrix vector product in the Lanczos propagator. The GPU accelerated approach makes use of cuSparse and cuBlas routines. The matrix vector product is a CSR cuSparse kernel. The whole Lanczos algorithm runs on the GPU.
The memory bottleneck within GAP is the sparse Hamiltonian. The idea is to split the matrix across the MPI ranks. Then the matrix vector product is performed on each rank and the results are gathered on rank 0,

$$Hy = (H_0 + H_1 + H_2 + H_3)y = b, \tag{65}$$

with H the complete Hamiltonian of the system and H_i the Hamiltonian on the i-th node. The best approach to obtain a node wise distributed Hamiltonian is to generate it in a distributed fashion. The parallelization of the array wise products needed is done with openMP for all implementations. The GPU accelerated version of GAP uses GPU kernels for the generalized eigenvalue problem, as it can become a bottleneck for larger N_b. The GPU accelerated version uses CUDA aware MPI. Every GPU does get matched with a MPI rank. The data transfer between GPUs is then handled by MPI. In the current production release of GAP the whole Lanczos algorithm is done on the GPUs.

3.3 Performance of GAP

In this section the performance of GAP is evaluated with the help of five test cases. These test cases were chosen to map the computational challenges of the simulations GAP was designed for. GAP is a cross platform simulation but the focus of this performance analysis is on the GPU scalability. As most of the run time is spent in the Lanczos propagator only the time spent in the propagator is measured for the performance evaluation. Every testcase uses 500 timesteps and a dimension of the Krylov space of 10. This results in 5000 matrix vector products. For an analysis that includes monetary efficiency refer to appendix .2.

testcase 1

The first test case maps a simulation in the dipole approximation that uses only z-polarization. This significantly reduces the size of the problem, as there are no transitions between m-states. The size of the matrix is small enough to fit into a single GPU. The hardware configuration for the GPU version is a p3.2xlarge with a single V100 and a c5.metal with 96vCPUs for the CPU accelerated version. The ift40070 is a workstation with 24vCPUs.

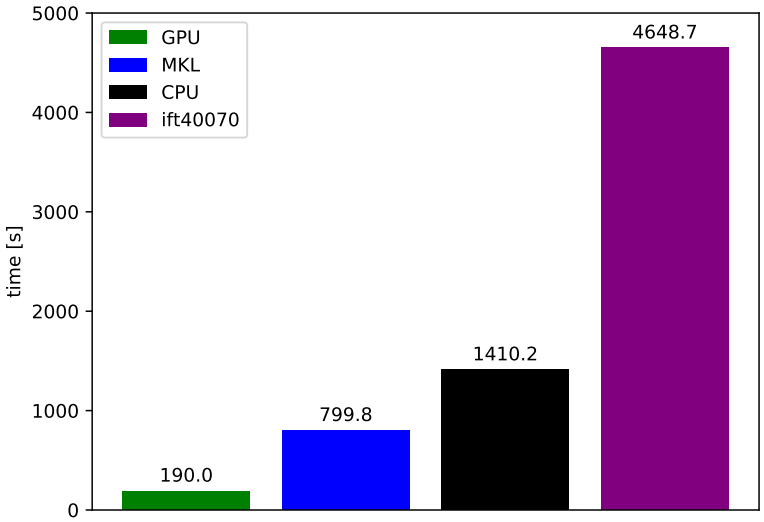


Figure 9: Time needed for a dipole calculation on a single V100 GPU. The COO and MKL code ran on 96 vCPUs. The ift40070 is a local workstation.

In figure 9 the time to complete the propagation is visualized. The GPU solution is the fastest. The MKL approach is a factor of 4.2 slower than the GPU approach. The COO version executed on the c5.metal instance is a factor of 7.5 slower than the GPU approach. The execution time of the COO approach on ift40070 does need 24.5 longer than the GPU approach.

testcase 2

The second test case is to map a beyond dipole simulation. These do require larger matrices that might not fit into the GPU memory of a single GPU. For multi GPU code to work efficiently the scaling of the code is of highest interest. This test case is designed to identify the scalability of GAP. The nnz are kept constant. The nnz are chosen to nearly fill the GPU memory of a single GPU. With an increase in the number of ranks every rank has less work to do. In figure 10 the time to complete the propagation is shown. For all versions of the code an increase in performance with an increase in the number of ranks can be seen. The speed up eventually slows down and approaches a limit. Generally the GPU version outperforms the

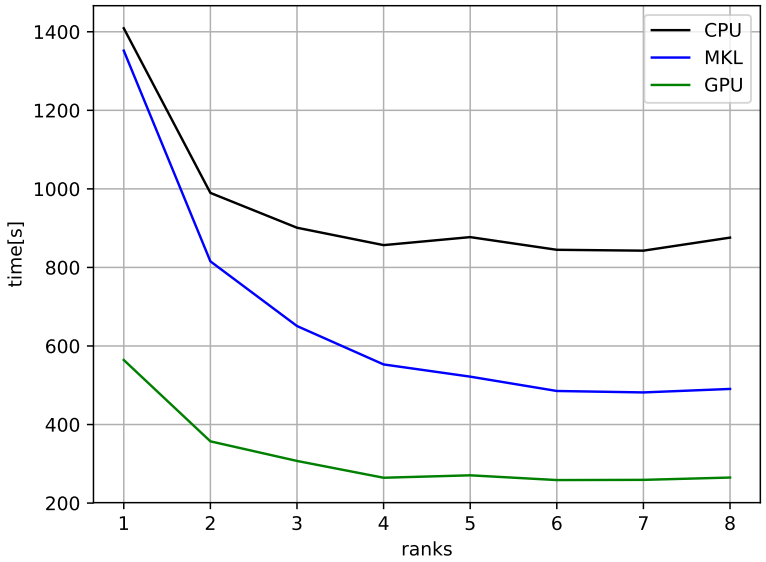


Figure 10: The time to complete testcase 2 is shown. One rank is one K80 GPU or 12 openMP threads for the MKL/COO version.

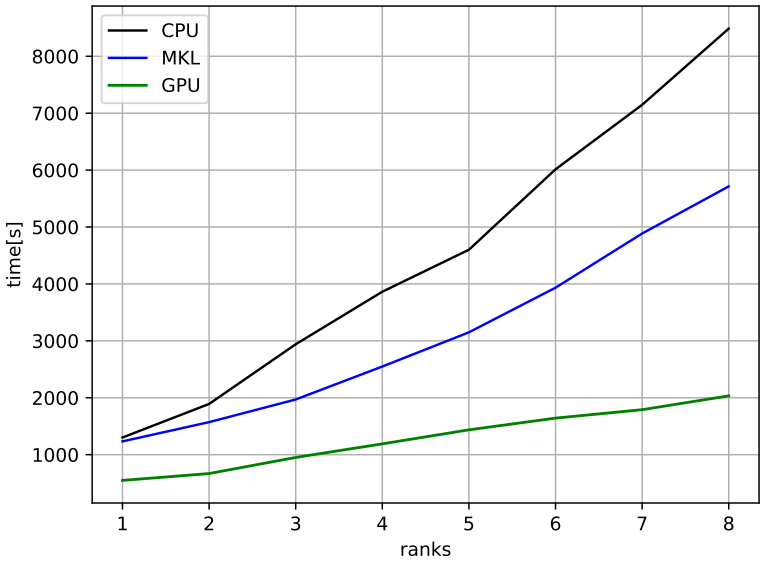


Figure 11: The time to complete testcase3 is shown. One rank is one K80 GPU or 12 openMP threads for the MKL/COO version.

MKL, and the MKL outperforms the COO version. For one rank the COO and MKL versions of the code deliver similar results. As more ranks are used the MKL code scales better. The COO and the GPU versions reach their performance limit with four ranks. The MKL version reaches the limit with six ranks. The COO curve shows a local maximum when five ranks are used.

testcase 3

The third test case also maps a beyond dipole simulation. In this test case all GPUs are filled to the maximum of their memory. This does allow the GPUs to operate at their maximum efficiency.
In figure 11 the time to complete the propagation increases with the problem size holding a constant ratio of nnz to number of ranks. This is mainly due to communication and synchronization costs. The slope for both the COO and MKL versions increase with time while the slope of the GPU version stays roughly constant after an initial increase. Again the GPU version outperforms the MKL and the MKL outperforms the COO version. Using five ranks a change of slope

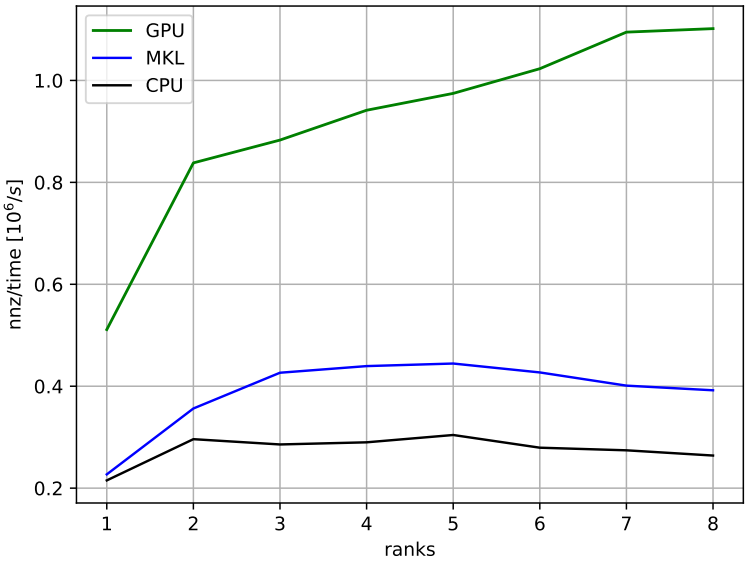


Figure 12: The throughput measured in nnz/time in testcase3. One rank is one K80 GPU or 12 openMP threads for the MKL and COO version.

can be observed in the COO curve.
In figure 12 the index nnz/time is displayed. It allows for a better understanding of the scalability. As long as it increases with an increase in ranks, more ranks should be used if available. The GPU version has a 65% performance increase using two GPUs. Afterwards its scales slower due to more communication and synchronization overhead. It reaches a near limit at around seven GPUs out of eight in the system. The COO and MKL reach their peak performance earlier. The MKL curve has a plateau between three and five ranks and then starts to decline in performance. The COO version has generally a lower performance after an initial increase it stays on a plateau and then declines after five ranks.
Figure 13 displays the ratio of the CPU and the GPU times. This is a good index to compare the direct performance. The COO/GPU increases with time. At rank five a small performance drop of the COO version is observed. The MKL/GPU curve is also overall rising but has the global minimum at three ranks. It then slowly rises again with six ranks having a lower performance ratio than one rank.

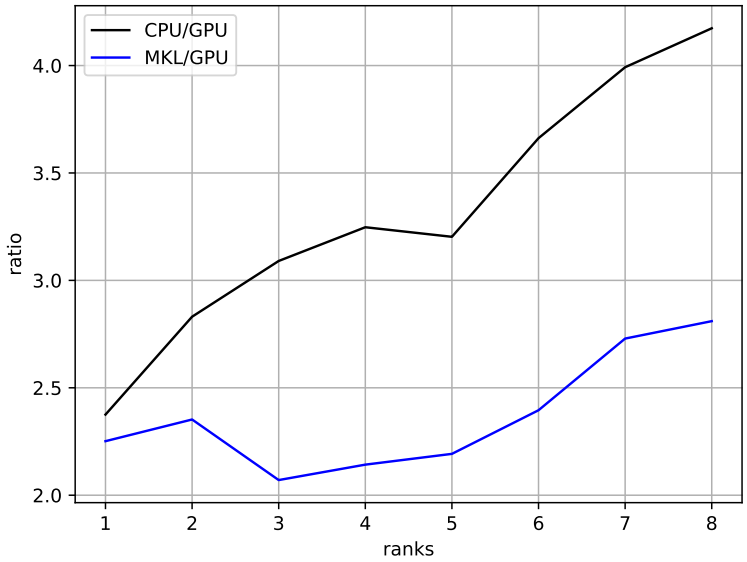


Figure 13: Testcase 3 the COO/GPU and MKL/GPU ratio. One rank is one K80 GPU or 12 openMP threads for the MKL/COO version.

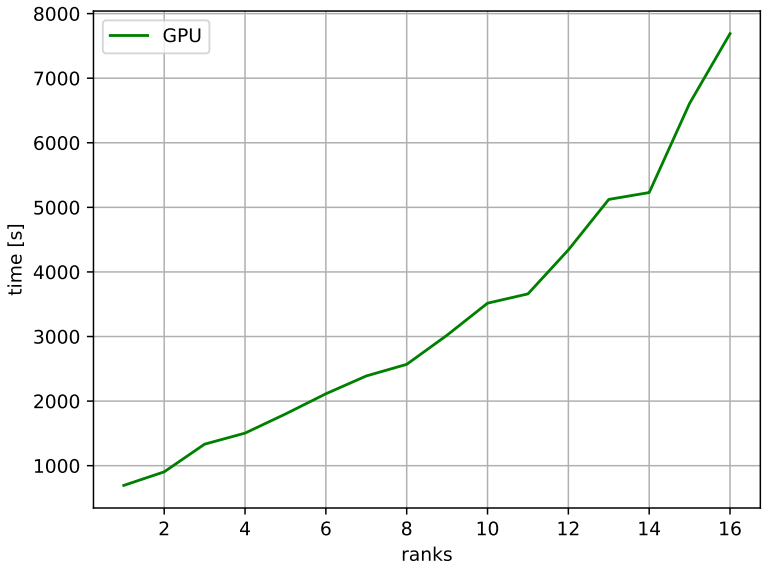


Figure 14: The time to complete testcase3 is shown. One rank is one K80 GPU or 12 openMP threads for the MKL/COO version.

testcase 4

The fourth test case is designed to look at the scalability of large problems on the GPU. It uses the p2.16xlarge instance that provides 16 K80 GPUs with a combined GPU memory of 192 GB. The GPUs are kept at full capacity. Figure 14 shows that the increase in time is near linear up to eleven ranks. Afterwards the growth in time becomes super linear. In figure 15 the index nnz/time is displayed. It shows that the peak performance is reached with eight ranks. Eleven ranks do reach a similar performance. Afterwards the performance drops significantly.

testcase 5

Test cases 1-4 have used instances accelerated by K80 GPUs for the GPU version. The other accelerator card available in the AWS cloud is the V100. The V100 has 16GB of GPU memory and about 4 times the raw computational power of a K80. The downside is that it costs 3.3 times as much as a K80. This testcase determines whether it possible to transfer the computational power to cost advantage of the

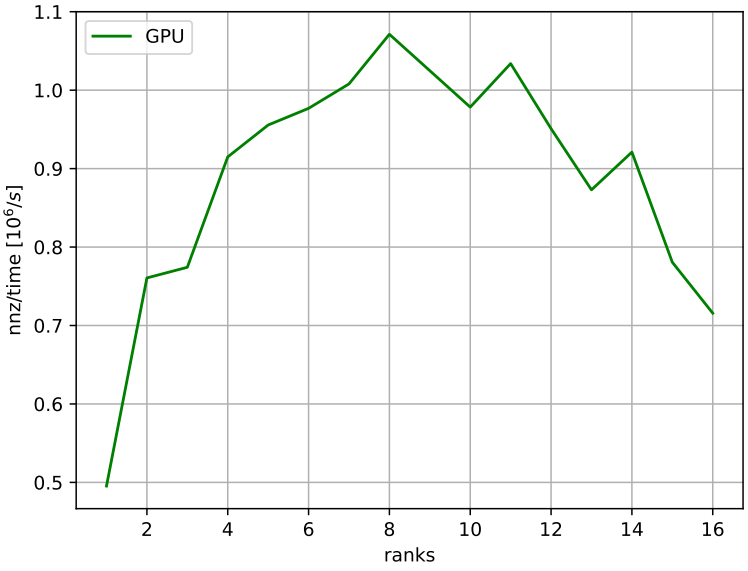


Figure 15: The throughput measured in nnz/time in testcase4. One rank is one K80 GPU.

V100 over the K80 into the simulations. The higher computational power to memory ratio of the V100 compared to K80 results in a factor of three lower computation time for the matrix vector product for a GPU at full memory capacity. This results in communication and synchronization times having a higher impact compared to the K80 GPUs. For this test case a p3.8xlarge instance with 4 V100 GPUs is used.

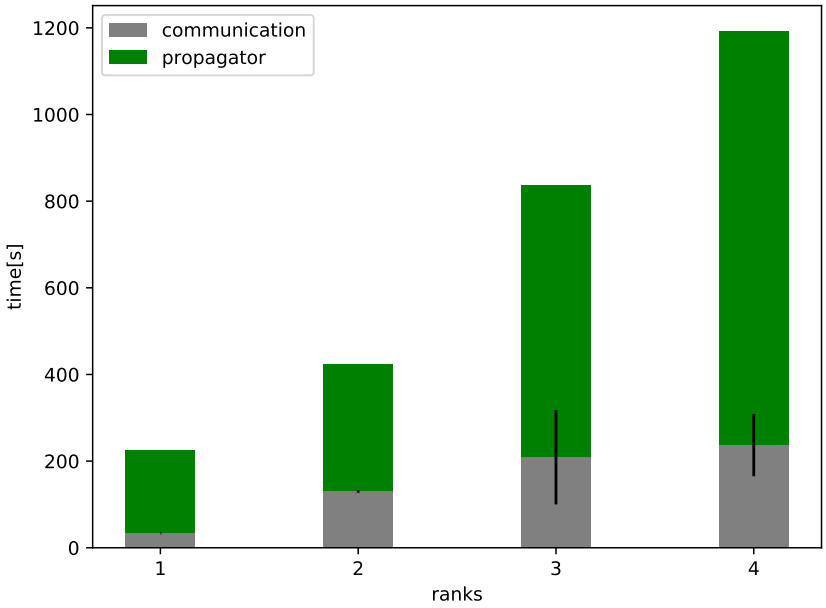


Figure 16: The time to complete testcase 5 and time spend on communication is shown. One rank is one V100.

Figure 16 shows that the run time of the algorithm increases drastically with an increase in the number of GPUs involved. As communication time is a candidate for the run time increase GAP was profiled with nvprof. For every GPU an own nvprof profile is obtained. The black error bars in the plot are the difference between the highest and the lowest value. The gray column represents the highest value as the slowest rank determines the speed of the algorithm.
Table 8 shows that the communication time varied for different GPUs. When three or four GPUs were used the differences between the communication times on each GPU were significant.

Number GPUs	rank0	rank1	rank2	rank3
1	38.2	-	-	-
2	130.0	129.8	-	-
3	196.2	209.1	72.1	-
4	149.9	237.0	207.7	213.5

Table 8: Table of the nvprof output for testcase 5. The entries are the sum of the communication time host to device, device to host and device to device given in seconds.

3.4 Evaluation

In this section the results obtained in the test cases are interpreted. For problems that fit in a single GPU the GPU version outperforms the CPU versions significantly. The GPU can benefit from its many cores and high memory throughput. The second testcase shows that it is useful to use more GPUs even if not every GPU is filled up to its capacity. To evaluate testcase 3 a closer look at the CPU hardware is needed. The CPU accelerated versions run on a c5.metal instance. This instance has a dual socket main board. It should be seen as two 48 vCPU CPUs with a fast interconnect. The limiting factor for the COO and MKL accelerated versions is the memory bandwidth between the main memory and the CPU. As the CSR matrix has a lower memory consumption 2.4.1 the MKL scales better than the COO version. In both testcase 2 and 3 a slight increase in run time for five ranks can be observed. This equals 60 vCPUs meaning that both CPUs have to be used and thereby a greater communication overhead has to be accounted for.

Test cases 4 and 5 reveal that the fundamental challenge for GPU scaling are the the synchronization times. Testcase 4 shows a drop in performance for a large matrix >100 GB. To find the reason for the synchronization delays further benchmarking would be needed. Unfortunately it is not feasible to benchmark the 16 K80 GPUs in testcase 4. As the same effect can be seen for fewer but faster V100 GPUs testcase 5 was further investigated with the Nvidia profiler. The overall communication time is quite reasonable and does not increase as much as expected. The communication time in Table 8 is the sum of host to device, device to host and device to device memory copies. The spread between the lowest

and the highest communication times become large with three and four GPUs. The nvprof profiles output gives us the accumulated communication time for a GPU over the 5000 iterations of the testcase. This is a major problem as it can not be determined if it is the same GPU getting low priority on the network for every matrix vector product and therefore has longer communication time, or if the lowest priority in the network is assigned by random for every matrix vector product. This does lead to synchronization issues that can not be detected easily. The differences in the communication times for different GPUs shown in table 8 are an indicator for synchronization problems that cause the performance drop. It shows that for 3 GPUs GPU2 was 2.5 times faster than GPU0 and GPU1. But for 4 GPUs GPU0 was the fastest by a factor of 1.6. As the overall compute load does not increase and the communication time stays within reasonable limits it is difficult to explain the worse scaling in testcases 4 and 5. The remaining factors are host device synchronization and synchronization between GPUs.

3.4.1 Guidelines

With the knowledge obtained the following guidelines to run GAP are formulated:

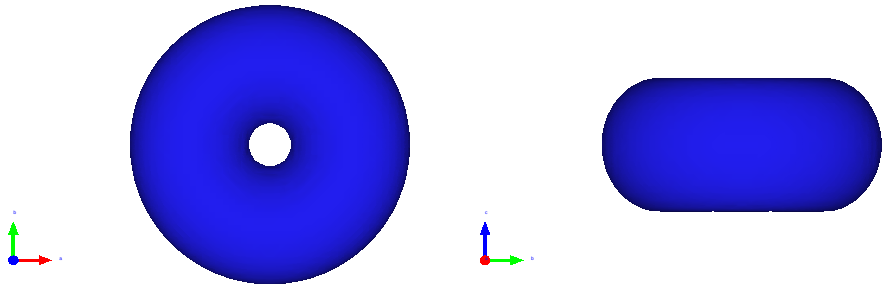
- If the problem fits within GPUs available use the GPU version.
- Use all GPUs available even if not every is filled up to maximum capacity.
- If the GPU version is not applicable but the MKL and enough memory to sort the matrix use the MKL version.
- If neither is available or memory is a constraint use the COO version.

4 A study on atomic stabilization in the 800 nm regime

GAP has proven to be a powerful tool to study hydrogen like problems. In this chapter a study of the 5g circular state interacting with a 800 nm laser is described. The results are in agreement with the data from a study in 2011 by S. Askeland [19].

4.1 Physical set up

The hydrogen atom has a binding energy of 13.6 eV and the absorption of a single photon exceeding that energy will induce a break up of the system. With an increase in flux multiple photons can interact. This allows multi photon processes and above threshold ionization (ATI) [19] [20]. Generally an increase in the break up probability is expected with an increase in laser intensity. 30 years ago theoretical studies of atomic hydrogen in ultra intense, high frequency laser fields showed a contradicting behavior of the break up probability. They showed that an atom might get more stable with an increase in the flux of photons. This phenomenon is called atomic stabilization [21]. As atomic stabilization is a high-frequency phenomenon, it is expected to be important at photon energies exceeding the the binding energy of the system. Atomic stabilization has been experimentally confirmed for low-lying Rydberg atoms [22].
In this work we investigate atomic stabilization on x and z-polarized laser pulses acting on the 5g state. The 5g state is a torus lying in the x,y plane. In figure 17a the initial 5g state of the simulations is plotted as a shot along the z-axis. Figure 17b shows a shot along the x-axis of the 5g state. The pulses are of short and intense character and within reach of conventional Ti:sapphire lasers [19]. The m quantum number is set equal to l for the initial state, this lets the electron "orbit"



(a) shot along z-axis. (b) shot along x-axis.

Figure 17: A visualization of the circular 5g state.

the nucleus. The maximum quiver velocity is defined as:

$$v_{max} = \frac{E_0}{\omega} \tag{66}$$

with E_0 the maximum laser field and ω the angular frequency of the photons. As it is less than 2.5% of the speed of light, relativistic effects can be neglected. The pulse duration was chosen as ten optical cycles.

4.1.1 Goal of simulation

In this study the goal is to look at the ionization and excitation probabilities and the probability for the electron to remain in the initial state. These probabilities will be referred to as p_{init}, p_{exc} and p_{ion}. Further the energy distribution at the end of the pulse is investigated.

4.2 Convergence tests

Earlier in this thesis a model to describe the laser atom interaction was defined. This model was then implemented on a computer. The researcher has to choose a grid on which the interaction will take place. If this grid is too tight the simulation becomes inaccurate. If however the grid is too large the simulations will not finish in a reasonable amount of time. Therefore it is immanent to do test runs for every parameter to be used in the simulations. The goal is to start with a guess for a parameter and then alter the parameter until the simulation returns the same

outcome. If an increase in the parameter approaches a limit convergence in this parameter is assumed. The value of convergence is then used in the simulation of the physical problem. In this study the following parameters have to be converged: the maximum energy E_{max}, the maximum l-quantum number L_{max}, the box size b and the number of *timesteps*. It is assumed that if the simulation converges for the highest intensity used in the simulation it converges also for lower intensities. Further it is assumed that if a parameter has converged in the beyond dipole approximation it is also converged in the dipole approximation. The convergence has to be tested separately for the x and the z polarization. In the following tests the convergence in p_{exc} and p_{ion} is investigated. If both are converged so has p_{init}. The convergence tests are conducted by keeping three parameters constant and varying one. As an initial guess the following parameters were chosen,

$$\begin{aligned} b &= 750\ a.u. \\ E_{max} &= 10\ a.u. \\ L_{max} &= 20 \\ timesteps &= 3000 \end{aligned} \tag{67}$$

This initial guess is chosen on the basis of experience.
In figure 18 the probabilities for the electron to be in an excited or an ionized state for a z-polarized laser field are shown. For the upper plot probabilities as function of E_{max} are plotted. Both curves are constant showing that E_{max} is converged in the lowest chosen value of $E_{max} = 0.5 a.u.$ The lower plot shows the dependency on the highest l quantum number allowed in the simulation. For l values smaller than 10 the probabilities fluctuate. From l equal to ten the probabilities are essentially converged.
In figure 19 the set up is equivalent to figure 18 but the laser is polarized in the x-direction. For the E_{max} a general trend can be seen. For a greater E_{max} a higher ionization probability is obtained. For the last value $E_{max} = 20 a.u.$ p_{ion} decreases slightly. It has to be noted that full convergence is not reached in E_{max}, but the discrepancy between $E_{max} = 10\ a.u.$ and $E_{max} = 20\ a.u.$ is less than one percent. The excitation probability shows a curve that is mirrored at the E_{max} axis. This indicates that p_{init} is constant. For the l values strong fluctuations can be seen for l smaller than 10. At $l = 12$ a local maximum for the ionization probability is reached. For higher l the ionization probability converges towards a limit. Full convergence has been reached for l greater than 20. p_{exc} shows a similar behavior

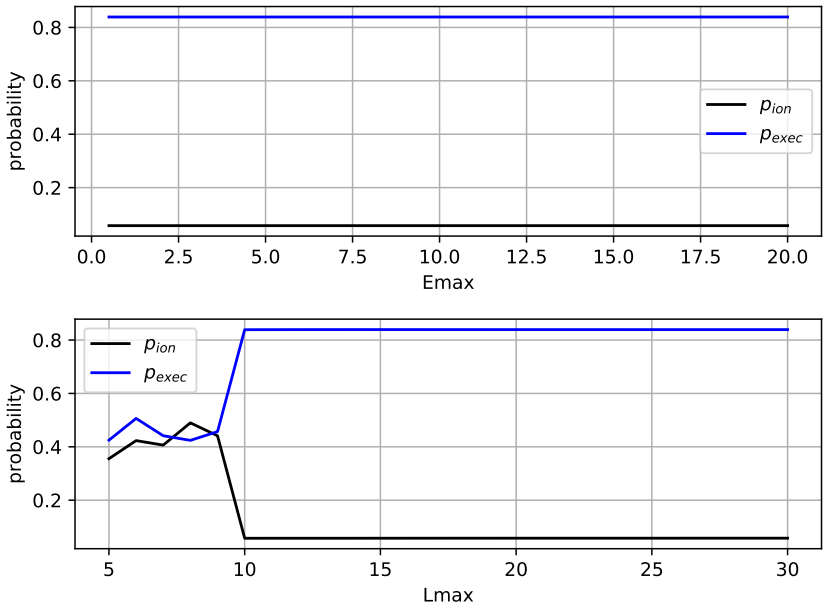


Figure 18: Convergence test for the z-polarization. The probability for the electron to be excited/ionized is plotted as a function of the maximum energy in the upper plot and as a function of L_{max} in the lower plot.

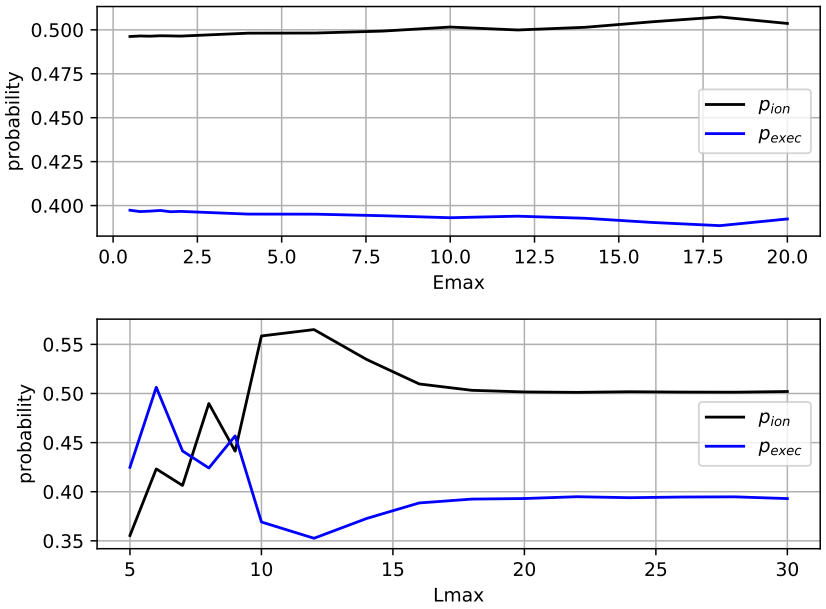


Figure 19: Convergence test for the x-polarization. The probability for the electron to be excited/ionized is plotted as a function of the maximum energy in the upper plot and as a function of L_{max} in the lower plot.

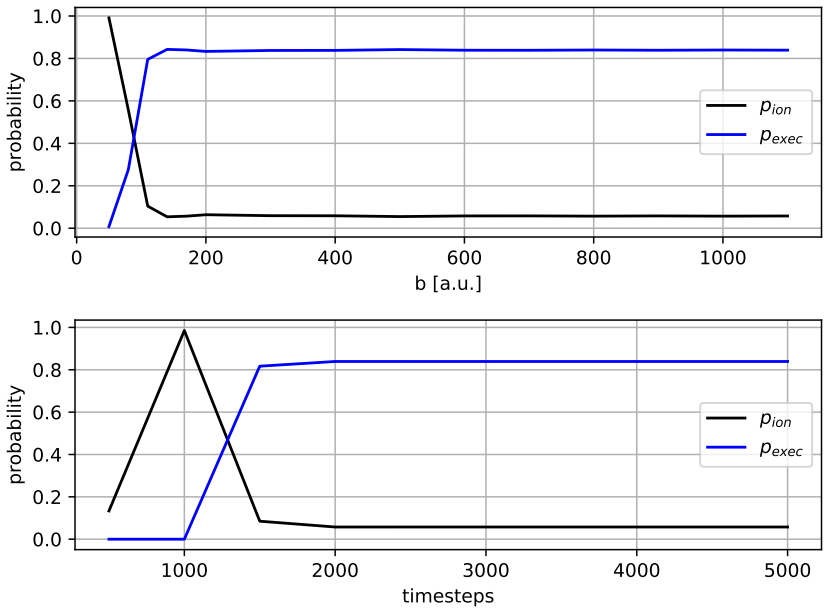


Figure 20: Convergence test for the z-polarization. The probability for the electron to be exited/ionized is plotted as a function of the box size in the upper plot and as a function of the number of timesteps in the lower plot.

with fluctuation for low l, a minimum for $l = 12$ and monotonous increase towards a limit at $l = 20$.

In figure 20 the probabilities for the electron to be in an excited or an ionized state are shown for a z-polarized laser field. In the upper plot probabilities as a function of b are plotted. For a small box the electron hits the wall leading to artificially high ionization values. The ionization probability then decreases and converges against a limit for a box greater than 160 a.u. The excitation probability start at 0 for a small box and then increases to a limit for a box greater than 160 a.u. The lower plot shows the dependency of the excitation and ionization probabilities on the number of timesteps. Both curves fluctuate for a low number of timesteps. They reach convergence for simulations with more than 2000 timesteps.
In figure 21 the probabilities for the electron to be in an excited or an ionized state are shown for an x-polarized laser field. In the upper plot probabilities as a function of b are plotted. For a small box size the excitation and the ionization probabilities fluctuate. They converge for a box greater than 200 a.u. The lower plot shows

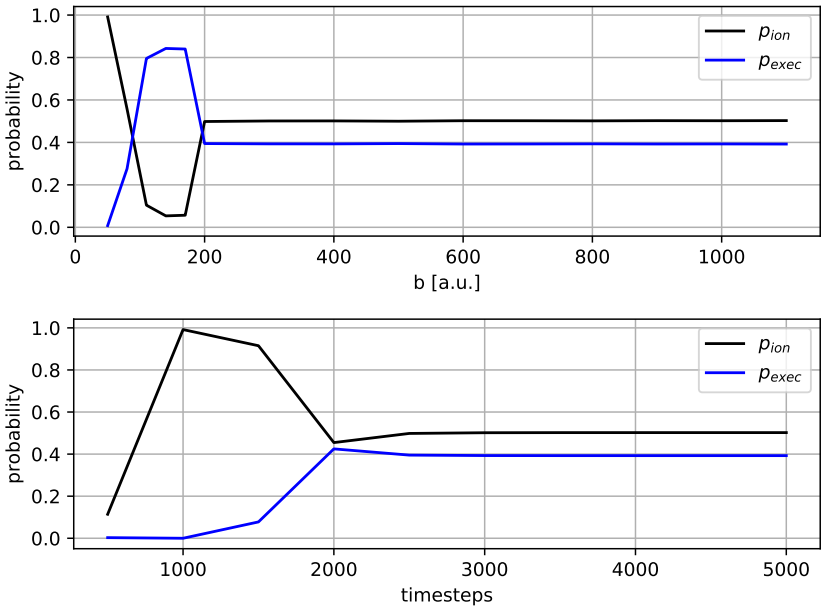


Figure 21: Convergence test for the x-polarization. The probability for the electron to be exited/ionized is plotted as a function of the box size in the upper plot and as a function of the number of timesteps in the lower plot.

the dependency of the excitation and ionization probability as a function of the number of timesteps. Convergence is reached if more than 2500 timesteps are used. If less timesteps are used the probabilities fluctuate.

4.2.1 Evaluation of convergence tests

In general the x and the z-polarization have different minimal parameters. In our case the x-polarization needs larger grid dimension. All simulations were conducted on the larger x-polarization grid, leading to the following parameters,

$$\begin{aligned} b &= 500 \\ E_{max} &= 10 \\ L_{max} &= 20 \\ timesteps &= 3000 \end{aligned} \tag{68}$$

The parameters were chosen larger than the minimal parameters of convergence. The reason is that the convergence test is a coarse guess due to the fact of changing only one parameter at the time and only checking for one intensity. Therefore a larger parameter provides a certain degree of "safety". The computational time was still below one hour for every data point. After the parameters for which the simulation converges have been determined the laser interaction with the 5g state will now be studied.

4.3 Results of the study

In this section the results of the study are presented. The first part is about the initial, excitation and ionization probability as a function of the intensity. The second part is about the energy distribution for a medium and for a high intensity.

4.3.1 Intensity sweeps

The simulations conducted in this thesis were in an intensity range of $[0.01,9.5] \cdot 10^{14}\ W/cm^2$. For every simulation the wave function was analyzed at the end of the pulse to obtain the probabilities for the electron to be in the initial, in the excited and in the ionized state. Both the dipole and the beyond dipole approximation were simulated. The simulations were conducted with an x-polarized laser field and with a z-polarized laser field.

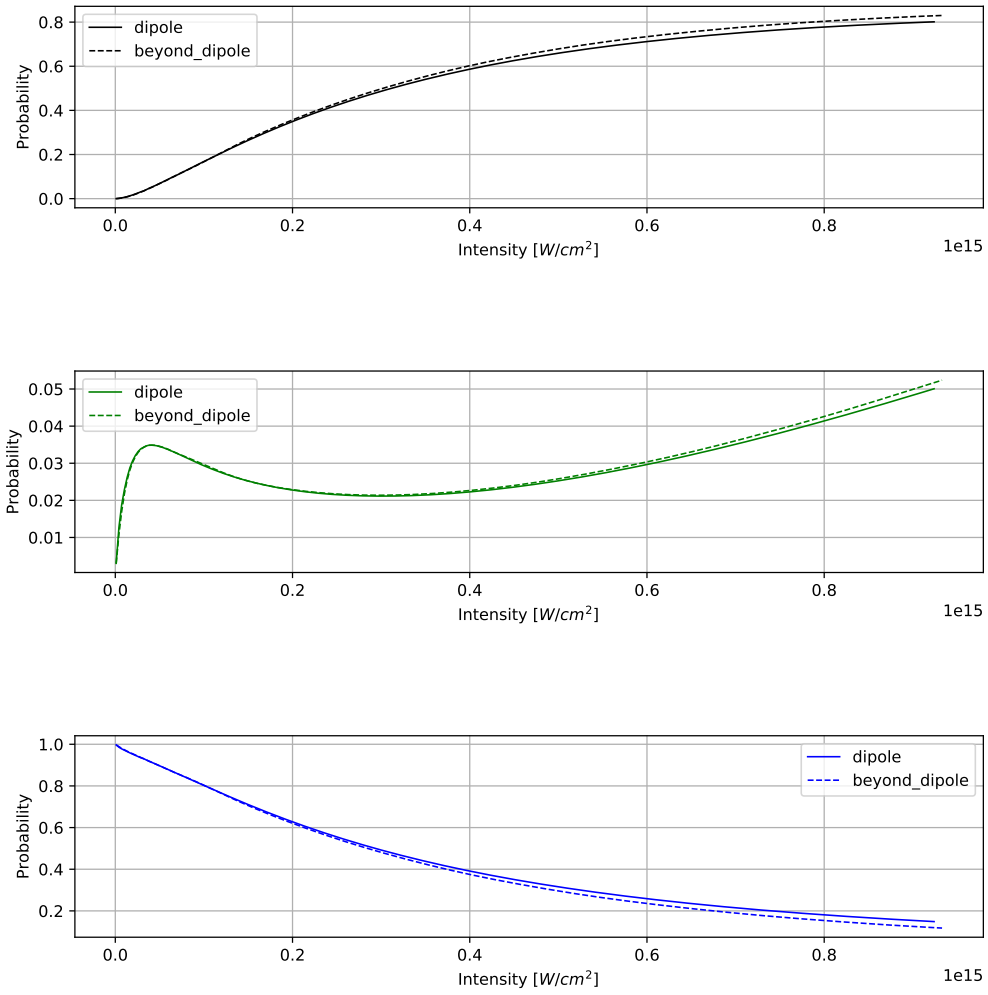


Figure 22: Intensity sweep for z-polarization. The upper panel shows the probability for the electron to be in the excited state. The middle panel shows the probability to be in an ionized state. The lower panel shows the probability to be in an initial state. The dotted lines are the beyond dipole simulations. The continuous ones are the dipole simulations.

In figure 22 the probabilities for the electron to be in the initial, an excited or an ionized state is shown. The laser field is polarized in z direction. For an increase in the intensity, the excitation probability increases monotonously in the dipole and the beyond dipole approximation. In the beyond dipole approximation the excitation probability is higher, with the difference growing with the intensity. The ionization probability has a similar shape in the dipole and the beyond dipole approximation. With the beyond dipole approximation returning greater probabilities than the dipole approximation, most visible for larger intensities. The dipole and the beyond dipole both reach a local maximum at intensities of $0.5 \cdot 10^{14}\, W/cm^2$. At $I_0 = 3.0 \cdot 10^{14}\, W/cm^2$ a local minimum is reached followed by a monotonous increase of the ionization probability. The drop in the ionization probability with increasing intensity is called atomic stabilization. The initial probabilities are high for low intensities. They decline monotonously approaching 0 for high intensities. The beyond dipole approximation returns lower probabilities. The beyond dipole effects enlargen with higher intensities.
In figure 23 the probabilities for the electron to be in an initial, an excited or an ionized state is shown. The laser field is polarized in x direction. There are no beyond dipole effects visible. Therefore only the dipole curves will be discussed. The excitation probability increases monotonously reaching a limit for intensities higher than $8 \cdot 10^{14}\, W/cm^2$. The probability for the electron to be ionized increases with an increase in the intensity until a local maximum is reached at $I_0 = 2 \cdot 10^{14}\, W/cm^2$. After the local maximum has been surpassed the probability declines converging towards a limit. This is the atomic stabilization. The initial probability is high for low intensities. It declines monotonously approaching 0 for higher intensities. The ionization probability for the x-polarization is an order of magnitude higher than the ionization probability for the z-polarization. As the 5g state is a torus in the x,y-plane the x-polarization makes the wave packet hit the nucleus resulting in ionization. In the z-polarization there is less interaction between the wave packet and the nucleus resulting in a lower ionization probability.

4.3.2 Energy distributions

To get further insides of the physical system the energy distribution is analyzed. It is analyzed for intensities of $1 \cdot 10^{14}\, W/cm^2$ and for $9.5 \cdot 10^{14}\, W/cm^2$. As there are no great differences between the dipole and the beyond dipole simulations, only the results in the beyond dipole approximation are shown. Both a z-polarized and

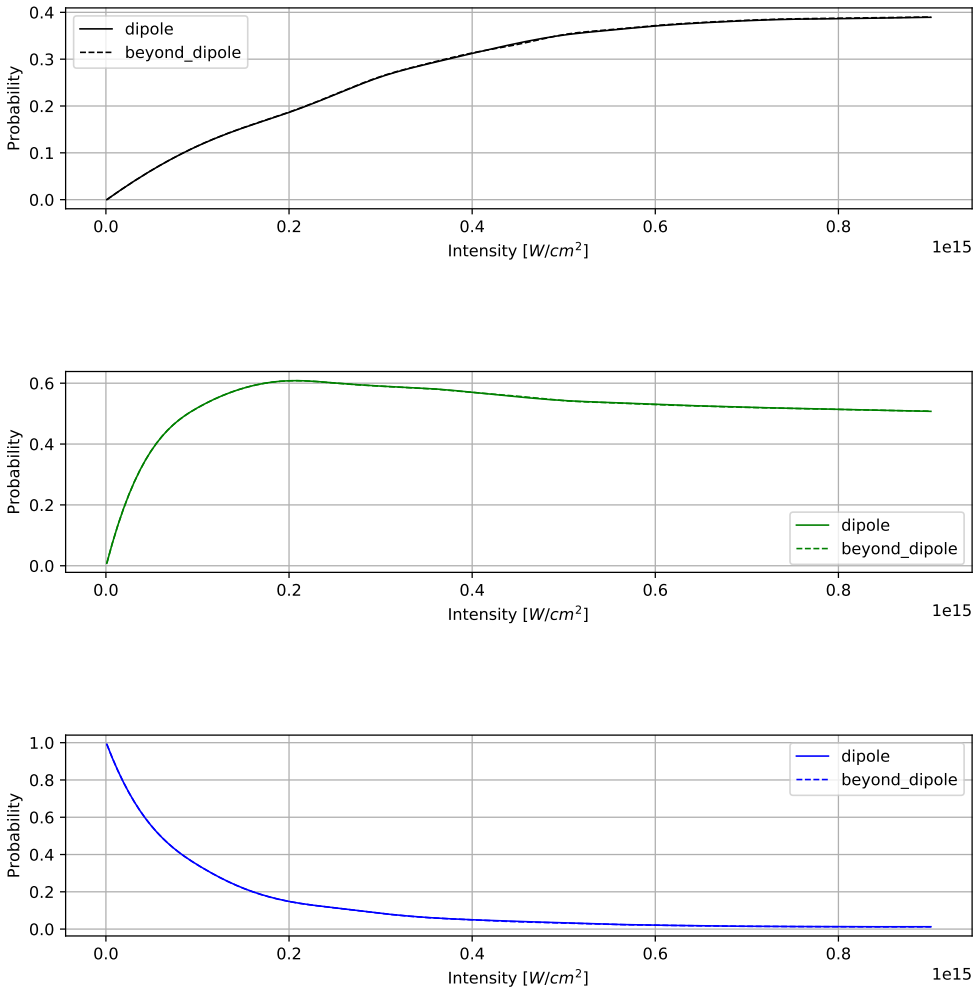


Figure 23: Intensity sweep for x-polarization. The upper panel shows the probability for the electron to be in the excited state. The middle panel shows the probability to be in an ionized state. The lower panel shows the probability to be in an initial state. The dotted lines are the beyond dipole simulations. The continuous ones are the dipole simulations.

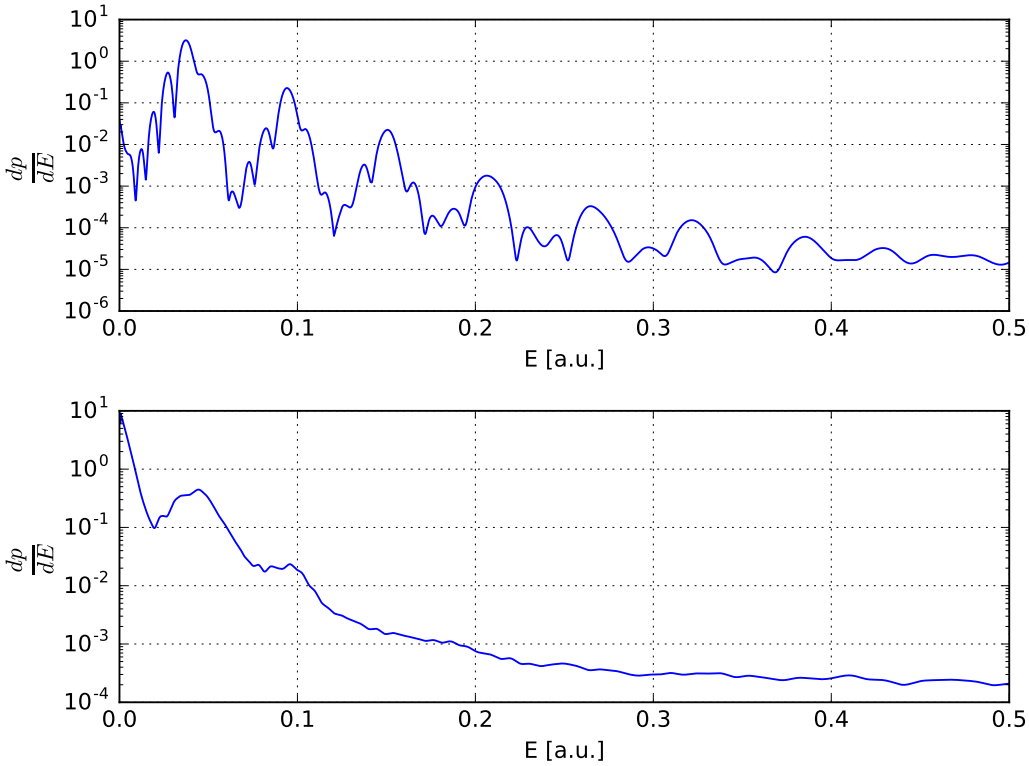


Figure 24: The energy distribution for a z-polarized laser field. The upper panel shows the distribution for an intensity of $1.0 \cdot 10^{14} W/cm^2$. The lower panel shows the distribution for an intensity of $9.5 \cdot 10^{14} W/cm^2$.

an x-polarized laser field were used.

In figure 24 the energy distribution in the z-polarized laser field is shown. The upper plot shows results for the intensity $1 \cdot 10^{14} W/cm^2$, the lower shows results for the intensity $9.5 \cdot 10^{14} W/cm^2$. For the upper plot multi photon resonances can be observed. The distance between the peaks is the expected $dE = 0.057 a.u.$ The first seven photon resonances can be seen. On top of the resonance interference can be observed. For the lower panel an increase in low energy electrons can be observed. The first two photon resonances are still visible but they are less dominant. The washing out of the energy distribution is an indicator for atomic stabilization [23].

In figure 25 the energy distribution in an x-polarized laser fields is shown. The upper plot shows the results for the intensity $1 \cdot 10^{14}\, W/cm^2$, the lower shows the

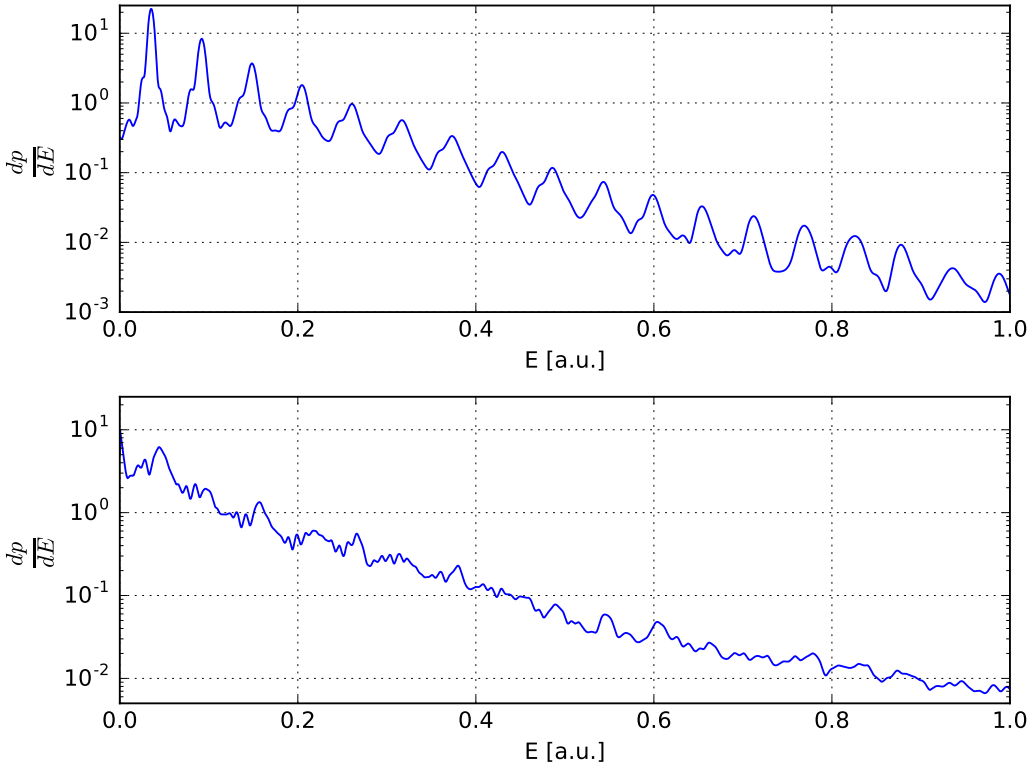


Figure 25: The energy distribution for a x-polarized laser field. The upper panel shows the distribution for an intensity of $1.0 \cdot 10^{14} W/cm^2$. The lower panel shows the distribution for an intensity of $9.5 \cdot 10^{14} W/cm^2$.

results for the intensity $9.5 \cdot 10^{14}\, W/cm^2$. For the upper panel the multi photon resonances can be observed for more than 15 orders. For the higher intensity shown in the lower panel the photo resonances is suppressed by atomic stabilization and interference. An increase in the number of low energy electrons can be observed.

4.4 Interpretation of results

The ionization sweeps have shown atomic stabilization both for the x and for the z-polarization. For the z-polarization a local maximum of the ionization probability was observed at $I_0 = 0.5 \cdot 10^{14}\, W/cm^2$. For the x-polarization the local maximum was at higher intensity of $I_0 = 2 \cdot 10^{14}\, W/cm^2$. While the local minimum was observed for the z-polarization at $I_0 = 3 \cdot 10^{14}\, W/cm^2$ it was not observed for the x-polarization. It is expected to be at intensities greater than $I_0 = 9.5 \cdot 10^{14} W/cm^2$. The suppression of the photon resonances for the energy distribution for higher intensities can be observed for z and for x-polarization. This can be explained through atomic stabilization eliminating the higher order photon peaks. The presence of low energy electrons for the higher intensities indicates atomic stabilization [23].
For high intensities the energy distribution shows that the photo resonance in the z-polarization is stronger suppressed than in the x-polarization. This is due to the z-polarization having passed the atomic stabilization region. While the x-polarization is in the build up of the atomic stabilization region.

5 Conclusion

A GPU accelerated framework that allows the study of laser atom interactions was developed. The framework was then used in the cloud and tested on its performance. For the different versions of code running in the cloud on similar hardware the GPU version outperformed the multi CPU versions. For smaller problems that fit within a single GPU a speed up of a factor of 7.4 for the openMP version and a factor of 4.4 for the MKL version was achieved. For larger problems that require multiple GPUs a speed up factor of 4.4 for the openMP and 2.8 for MKL was obtained. The most powerful local workstation used was a factor of 24.5 slower. For larger problems that require more than eight K80 GPUs synchronization problems were detected. To counter this a propagator that runs fully on the GPU should be used. It has already been developed and validated but it has not yet been through performance testing. Understanding the time lost in synchronization is a key factor for the further performance improvements of GAP.
The framework was then used in a study of the excitation and ionization of a circular hydrogen Rydberg state, the 5g (m=4) state, by a 800 nm laser pulse. Atomic stabilization was shown for both z and x-polarization. The x-polarization entered the stabilization region at higher intensities than the z-polarization. Beyond dipole effects were negligible for the x-polarization. In the z-polarization small beyond dipole effects were observed. For higher intensities the suppression of multi photon resonances due to interference and atomic stabilization and an increase in low energy electrons was observed. The interference phenomena should be investigated further for higher intensities and for circular states with a higher l-quantum number.

6 Appendix

.1 Talk SC19

Modern HPC approaches to Solve Time Dependent Quantum Mechanical Equations

SC19 Theater Talks

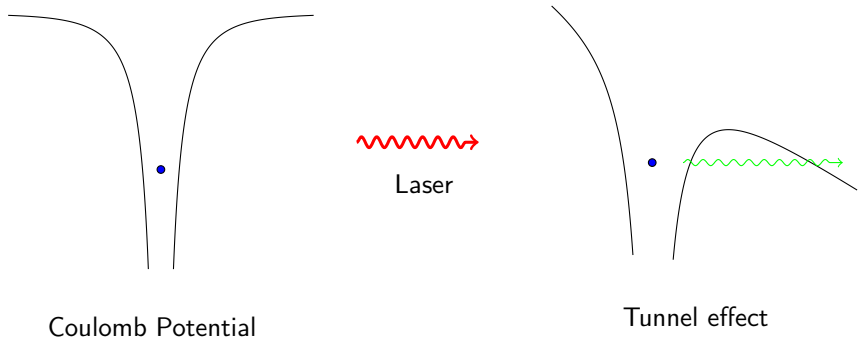
Konstantin Rygol

University of Bergen

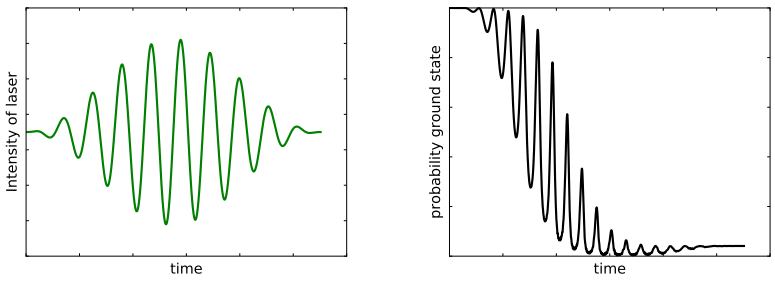
19 November 2019

Table of Contents

Laser atom interaction



Simulation of a fully ionized electron



Physical foundations

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = \mathcal{H}(t) |\psi(t)\rangle \quad (1)$$

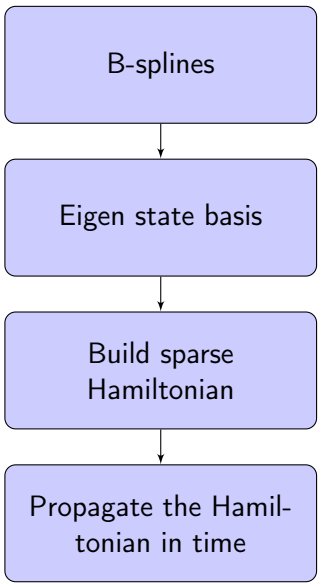
$$\psi_{n,l,m} \quad (2)$$

$$\mathcal{H} = \frac{p^2}{2m} + V - \frac{q}{m} A p_z \quad (3)$$

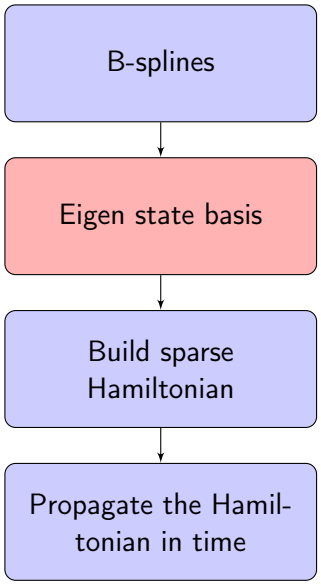
- Time dependent matrix
- Two entries per cell: Value and factor ft(t)
- Problem: Most numerical libraries can not handle that

Foundations of the simulation

- Particle is a placed in a multidimensional box
- With dimensions Lmax,Mmax,Emax,Boxsize
- Grid points are given by N_b
- The photon energy influences the timescale
- Convergence depends on time steps and laser intensity

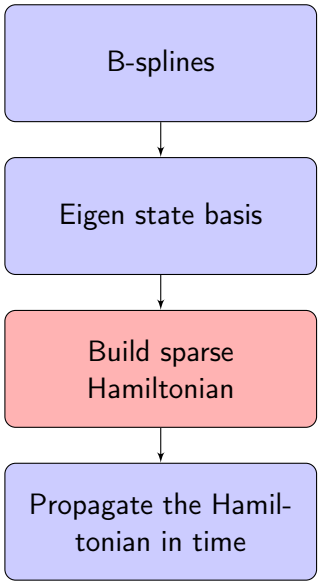


Building the eigenstate basis



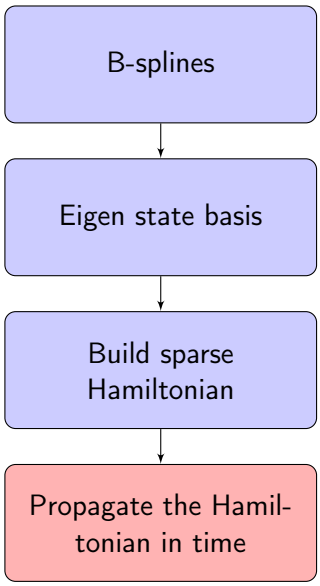
- Called Lmax times
- Arrays wise multiplication: $\Rightarrow \mathcal{O}(N_b^4)$
- Generalized Eigenvalue solver: $\Rightarrow \mathcal{O}(N_b^3)$
- $\Rightarrow \mathcal{O}(Lmax \cdot N_b^4)$
- Bottleneck: computation power

Building the Hamiltonian



- $Lmax \cdot Mmax \cdot Bricks$
- For each Brick: $\mathcal{O}(N_b^2)$ elements
- For each Element: $\mathcal{O}(N_b^2)$ operations
- $\Rightarrow \mathcal{O}(N_b^4 \cdot Lmax \cdot Mmax \cdot Bricks)$
- Bottleneck: computation power & size of memory

Propagate the Hamiltonian



- Number of time steps t
- Dimension of the Krylov space
- Number of non zero Elements of the matrix
- $\Rightarrow \mathcal{O}(t \cdot dim\_kryl \cdot nnz)$
- Bottleneck: computation power

Different kind of simulations

- Dipole approximation $\equiv$ Mmax=0
- Beyond dipole Mmax $\leq$ Lmax
- Relativistic cases $\Rightarrow$ more bricks
- Different wavelengths $\Rightarrow$ different timescales
  $\Rightarrow$ difficulties of convergence
- What should we focus on ?

General thoughts on the simulation

- Represent Physics in the code
- Write code that is readable
- Maintainable
- Good performance
- Solution: Eigen library
- Can use MKL as back end + OPEN-MP

MPI parallelization + OPEN-MP

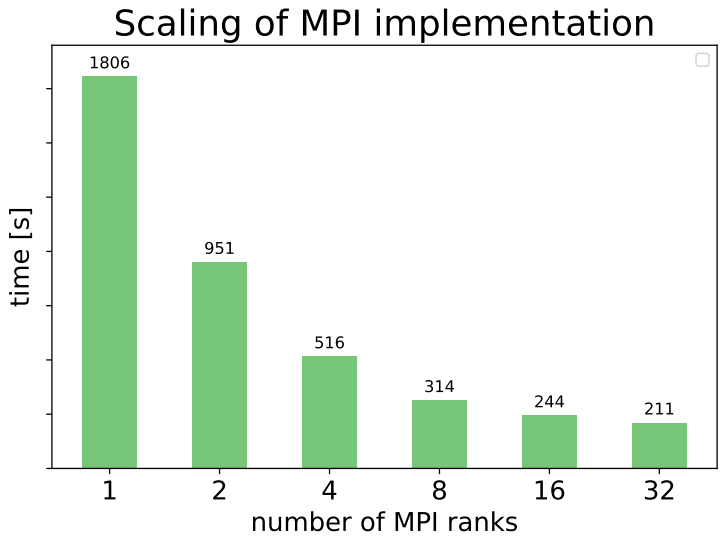
- Generate the matrix distributed
- Every node holds its share of the matrix

$$H = H_0 + H_1 + ... + H_n \tag{4}$$

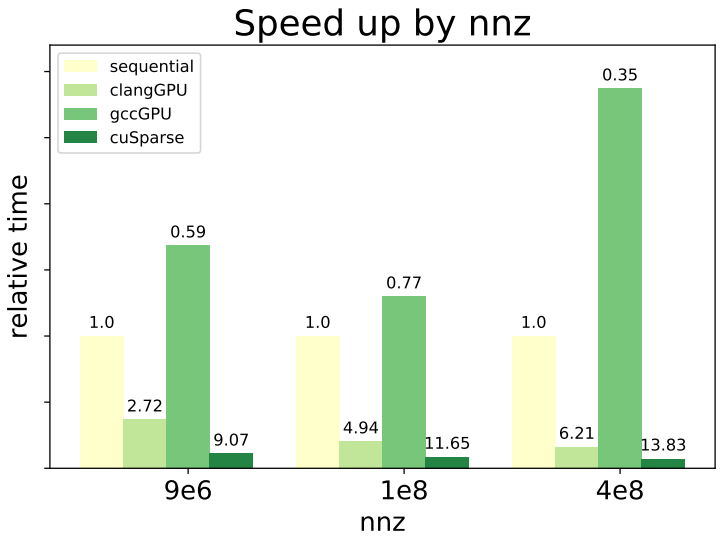
$$b = Hx = H_0x + H_1x + ... + H_nx \tag{5}$$

- The vector b is small
- OPEN-MP allows us two express second level of parallelism
- Gives us great speed up in the generation of the matrix

MPI Benchmark



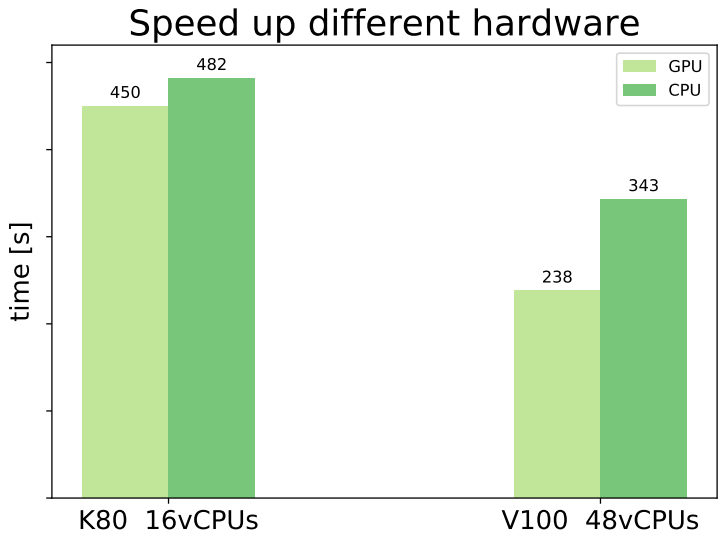
Benchmark Matrix Vector product



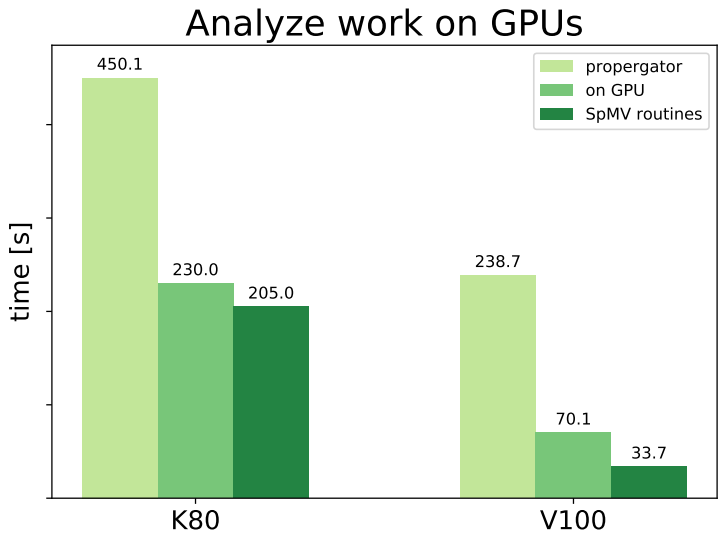
MPI + cuSparse + OPEN-MP

- std::vector<SpMatrix> (nmbrLEGO);
- Matrix is generated in COO sent to device and compressed to CSR
- Try to hold the matrix constantly in GPU memory
- Make use of CuSparse SpMV routines
- Downside: adds further level to code complexity

Benchmark of GPU vs CPU code



Analyzing the propagator



Conclusion & Future Work

- Different needs in hardware for different calculations
- Rent the hardware you need for the job
- Better OPEN-MP GPU support for open source compilers
- thrust library
- Other form of accelerators

.2 Intermediate report for AWS

Intermediate Report on the GPU Accelerated Propagator (GAP)

This is an intermediate report to show the results obtained up to this point.

Abstract

Over the last months GAP has been developed with funding from the AWS Cloud Credits for Research program. It shows the power of GPU accelerated HPC in Atomic physics. It is used to simulate the Time Dependent Schrödinger Equation for laser atom interaction. In combination with the AWS Cloud results, that used to need weeks to be calculated, can be calculated in hours.
In the era of HPC in the cloud, scalability and monetary efficiency are the limiting factors for HPC simulations. The idea behind this project is to offload parts of the simulation using the high level GPU libraries cuBlas, cuSolver and cuSparse. Thereby the development time is reduced significantly compared to a CUDA implementation of a multi GPU code. The maximum speed up obtained over a CPU implementation is 6.5 and an increase in monetary efficiency by nearly a factor 10 was achieved.

Implementation of the Simulation

GAP is written in C++ making great use of the Eigen library. Eigen is a templated C++ CPU library that is highly performant, well documented and produces clean, readable code.

GAP has a certain amount of time steps t. For every time step it runs the Lanczos algorithm on the time dependent Hamiltonian. The Lanczos algorithm creates a Krylov space. The eigenvalues of this space represent the eigenvalues of the Hamiltonian and are then used to calculate the next time step.

The Lanczos algorithm iterates until the maximum dimension of the Krylov space is reached. Every iteration start with broadcasting the current vector to all MPI ranks. The SpMV product is then executed. Then a MPI_Reduce is called to get the result to GPU 0. The newly obtained vector is then orthogonalized twice against all the vectors that were calculated previously in the same time step. The SpMV and reorthogonalization are the expensive parts of the Lanczos iteration. For the next Lanczos iteration the last acquired Krylov vector is used.

Parallelization of the Simulation

The Hamiltonian is a sparse matrix in the range from some hundred MB to several hundred GB depending on the physical system to be studied. It is

generated across the nodes with every node holding an equal part of it. This nodewise parallelization is done with MPI. If the GPU acceleration is used every GPU is attached to one MPI rank. Resulting in possibly multiple ranks per node. The Hamiltonian is kept constantly in GPU memory.
For the CPU version the parallelization within a node is done with OPEN-MP. The choice to use accelerators is made at run time through a command line option.

The computationalwise intensive parts are within the propagator. The computational wise most expensive part is the SpMV operation required by the Lanczos algorithm. The Lanczos propagator has been ported completely to GPU. The SpMV is calculated using multiple GPUs. The rest of the Lanczos algorithm is calculated on the GPU0 of the multi GPU system. Finally a computationalwise small part is done on the CPU to obtain the next time step. To obtain lower MPI communication delays, CUDA aware OPEN-MPI is used.

Test cases

All calculations are performed with 500 time steps and dimension of the Krylov space of 10.

Single GPU performance

A dipole approximation simulation results in a comparatively small matrix size of a couple hundred million nnz. In that case the matrix fits in one GPU, so we choose a p3.xlarge instance that has 100V card. For the CPU computation we use a c5.metal instance with 96 vCPUs.

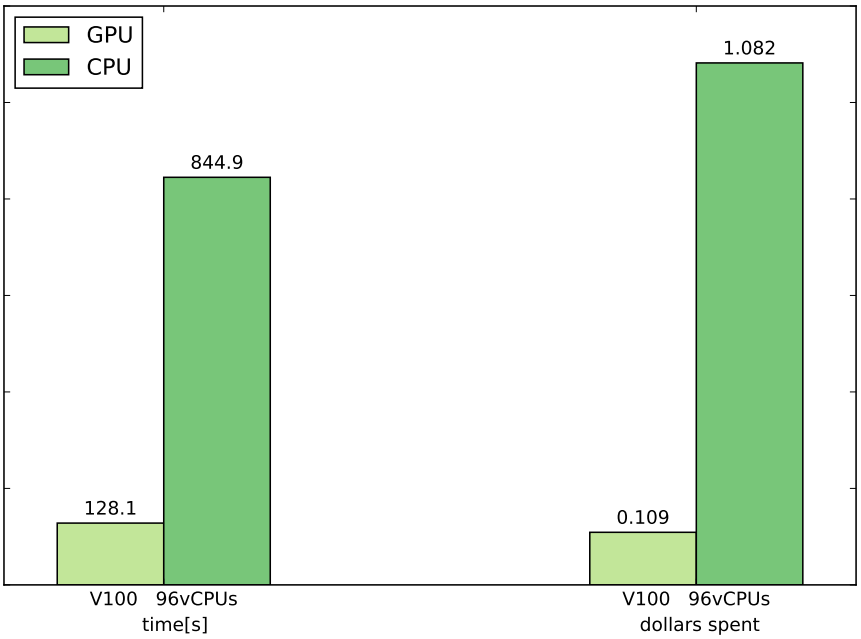


Figure 1: Dipole Approximation single GPU performance

The GPU is faster by a factor of 6.5 . But it is not only faster, it also has a higher monetary efficiency. The cost of the CPU instance is $4.61, the GPU instance costs $3.06. This results in the GPU having nearly 10 times the monetary efficiency over the CPU solution.

Scaling for a fixed size small problem

A simulation in the beyond dipole approximation results in a larger matrix with more than a billion elements. This requires a multi GPU instance. For this test case we use a p2.8xlarge, as it provides a good ratio of VRAM per Dollar. For the p2.8xlarge one K80 GPU is attached to one MPI rank. How does GAP scale for using multiple GPUs keeping the matrix size constant ? Each GPUs holds nnz/#GPUs elements.
To get a comparison for the CPU version, a c5.metal instance is used. Every rank owns a K80 (p2.8xlarge) or 12 OPEN-MP (c5.metal) threads.

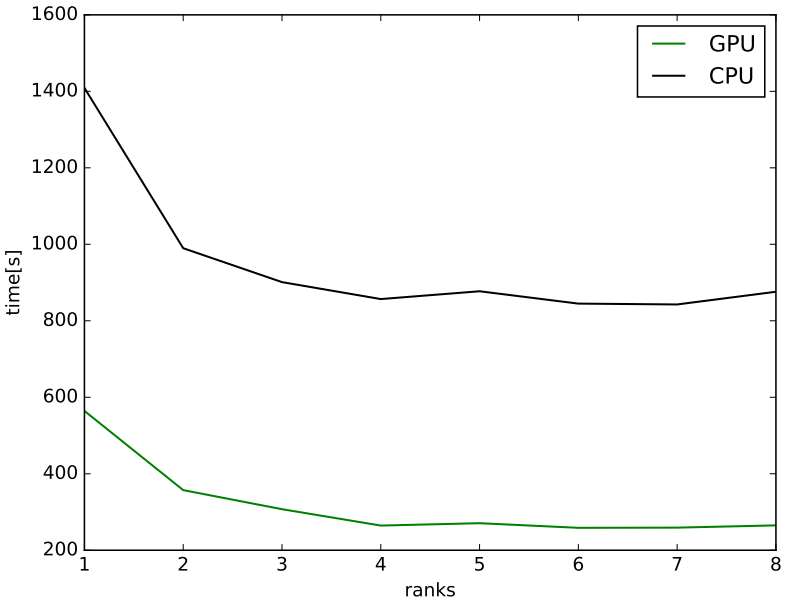


Figure 2: Multi GPUs with fixed small matrix size

Scaling for an enlarging problem

For this problem size we enlarge the matrix as we use more ranks. The idea is to fill every GPU to its near maximum of GPU memory. This is a good test for scalability. The matrix size is linear in the number of ranks used.

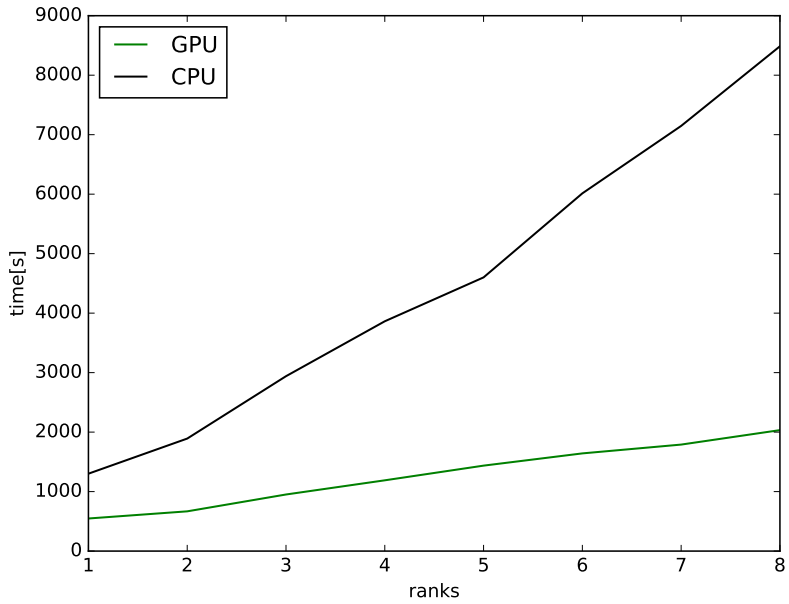


Figure 3: Multi GPUs, the matrix size increases linearly in the number of ranks: Time

To get a better overview over the scalability, the time per million nnz is ploted against the number of ranks. This number should be as low as possible and decrease with an increase in ranks.

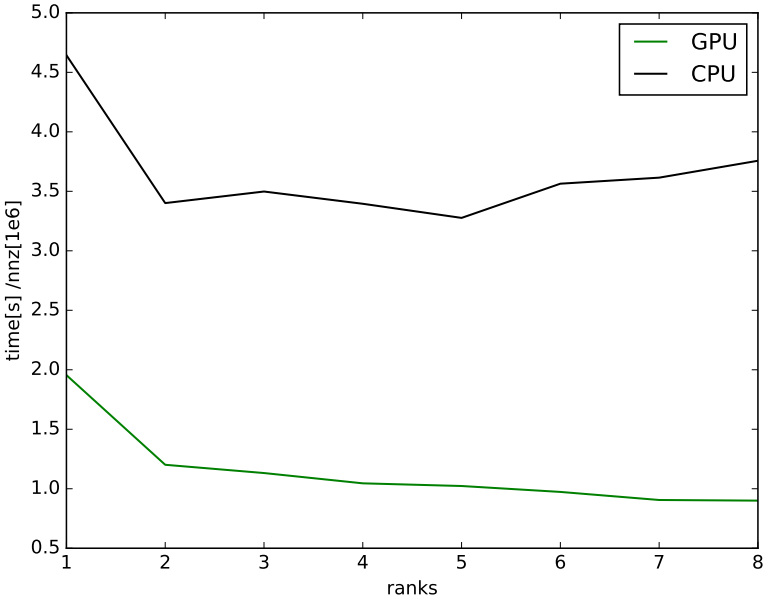


Figure 4: Multi GPUs, the matrix size increases linearly in the number of ranks: The time divided by nnz

To get a better understanding, the relative speed up of the GPU over the CPU code the ratio is plotted. The monetary efficiency is also plotted in the figure. The p2.8xlarge is more expensive with $7.20 than the c5.metal at $4.61.

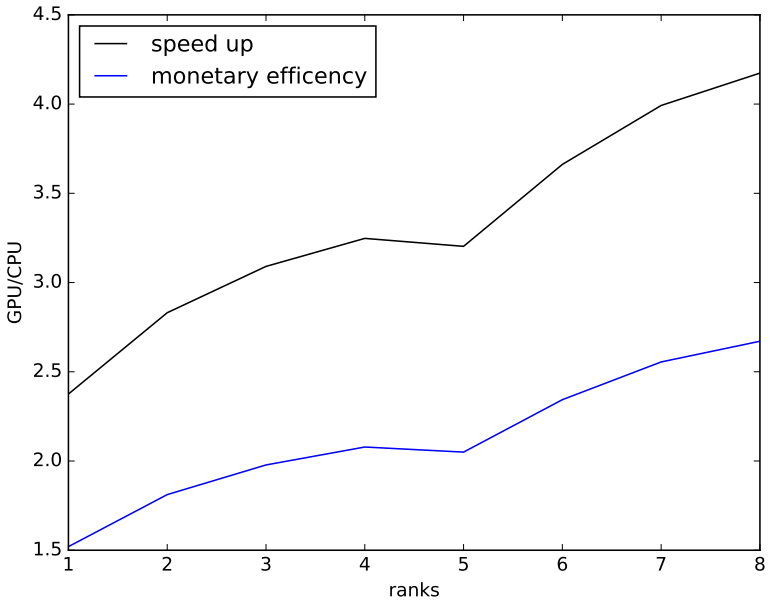


Figure 5: Multi GPUs, the matrix site increases linearly in the number of ranks: Ratio between CPU and GPU times

Results

The GPUs greatly decrease the time needed for simulations and thus greatly increase research output. They further increase the monetary efficiency of the calculations in the AWS cloud.

GPU accelerated computing can also be less power consuming than traditional CPU computing. This allows the reduction of the carbon footprint of our research. This however is difficult to quantify without insights on the IT infrastructure used in the data center.

Future Work

The next step is to build an AWS Cluster to compare node scalability between CPU and GPU nodes. There is also the larger p2.16xlarge instance that has not yet been used.

The V100 cards do offer a great performance with a fast network and NVLink as an interconnect between the GPUs. They may be more scalable by lowering

communication time. However they offer considerably less memory per dollar than the K80 cards.

I thank AWS for supporting this research through the AWS Cloud Credits for Research program.

7 Bibliography

[1] B. H. Bransden, C. J. Joachain, and T. J. Plivier, *Physics of atoms and molecules*. Pearson education, 2003.

[2] R. A. Millikan, "On the elementary electrical charge and the avogadro constant," *Physical Review*, vol. 2, no. 2, p. 109, 1913.

[3] R. Hooke, *Extracts from Micrographia: Or, Some Physiological Descriptions of Minute Bodies Made by Magnifying Glasses, with Observations and Inquiries Thereupon*. No. 5, Alembic Club, 1912.

[4] D. C. Cassidy, G. Holton, and J. Rutherford, *Understanding Physics: Student Guide*. Springer, 2005.

[5] J. C. Maxwell, *A treatise on electricity and magnetism*, vol. 1. Clarendon press, 1881.

[6] K. S. Tiwari and A. G. Kothari, "Design and implementation of rough set algorithms on fpga: A survey," *International Journal of Advanced Research in Artificial Intelligence*, vol. 3, no. 9, 2014.

[7] J. M. P. Cardoso, J. G. de Figueiredo Coutinho, and P. C. Diniz, *Embedded Computing for High Performance: Efficient Mapping of Computations Using Customization, Code Transformations and Compilation*. Morgan Kaufmann, 2017.

[8] J. L. Aurentz, V. Kalantzis, and Y. Saad, "Cucheb: a gpu implementation of the filtered lanczos procedure," *Computer Physics Communications*, vol. 220, pp. 332–340, 2017.

[9] P. A. M. Dirac, "A new notation for quantum mechanics," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, pp. 416–418, Cambridge University Press, 1939.

[10] H. S.-M. Wolff *et al.*, "Topological vector spaces," *Graduate Texts in Mathematics*, vol. 3, 1999.

[11] R. L. Jaffe, "Supplementary notes on dirac notation,quantum states, etc.," 1996. `http://web.mit.edu/8.05/handouts/jaffe1.pdf`.

[12] M. Førre, "Breakdown of the nonrelativistic approximation in superintense laser-matter interactions," *Physical Review A*, vol. 99, no. 5, p. 053410, 2019.

[13] H. Bachau, E. Cormier, P. Decleva, J. Hansen, and F. Martín, "Applications of b-splines in atomic and molecular physics," *Reports on Progress in Physics*, vol. 64, no. 12, p. 1815, 2001.

[14] A. S. Simonsen, T. Kjellsson, M. Førre, E. Lindroth, and S. Selstø, "Ionization dynamics beyond the dipole approximation induced by the pulse envelope," *Physical Review A*, vol. 93, no. 5, p. 053411, 2016.

[15] Y. Saad, *Iterative methods for sparse linear systems*, vol. 82. siam, 2003.

[16] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[17] W.-C. Jiang and X.-Q. Tian, "Efficient split-lanczos propagator for strong-field ionization of atoms," *Optics express*, vol. 25, no. 22, pp. 26832–26843, 2017.

[18] nvidia, "corperate timeline." `https://www.nvidia.com/en-us/about-nvidia/corporate-timeline/`.

[19] S. Askeland, S. A. Sørngård, I. Pilskog, R. Nepstad, and M. Førre, "Stabilization of circular rydberg atoms by circularly polarized infrared laser fields," *Physical Review A*, vol. 84, no. 3, p. 033423, 2011.

[20] P. Agostini, F. Fabre, G. Mainfray, G. Petit, and N. Rahman, "Phys rev lett 42: 1127;(b) kruit p," *Kimman J, Muller HG, van der Wiel MJ (1983) Phys Rev A*, vol. 28, p. 248, 1979.

[21] M. Pont, N. Walet, M. Gavrila, and C. McCurdy, "Dichotomy of the hydrogen atom in superintense, high-frequency laser fields," *Physical review letters*, vol. 61, no. 8, p. 939, 1988.

[22] N. Van Druten, R. Constantinescu, J. Schins, H. Nieuwenhuize, and H. Muller, "Adiabatic stabilization: Observation of the surviving population," *Physical Review A*, vol. 55, no. 1, p. 622, 1997.

[23] M. Førre, S. Selstø, J. Hansen, and L. Madsen, "Exact nondipole kramers-henneberger form of the light-atom hamiltonian: An application to atomic stabilization and photoelectron energy spectra," *Physical review letters*, vol. 95, p. 043601, 08 2005.