HAL open science

Experimenting with hybrid quantum optimization in HPC software stack for CPU register allocation

Brice Chichereau, Stéphane Vialle, Patrick Carribault

▶ To cite this version:

Brice Chichereau, Stéphane Vialle, Patrick Carribault. Experimenting with hybrid quantum optimization in HPC software stack for CPU register allocation. Third International Workshop on Integrating High-Performance and Quantum Computing (WIHPQC 2023) in IEEE Quantum Week 2023 (QCE), Sep 2023, Bellevue, United States. hal-04272048

HAL Id: hal-04272048

https://universite-paris-saclay.hal.science/hal-04272048v1

Submitted on 9 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Experimenting with hybrid quantum optimization in HPC software stack for CPU register allocation

Brice Chichereau
CEA, DAM, DIF
F-91297 Arpajon, France
Université Paris-Saclay, CEA,
Laboratoire en Informatique
Haute Performance pour
le Calcul et la simulation
Bruyères-le-Châtel, France
brice.chichereau@cea.fr

Stéphane Vialle
CentraleSupélec
Gif-sur-Yvette, 91192, France
Université Paris-Saclay, CNRS,
Laboratoire Interdisciplinaire
des Sciences du Numérique
Orsay, France
stephane.vialle@centralesupelec.fr

Patrick Carribault
CEA, DAM, DIF
F-91297 Arpajon, France
Université Paris-Saclay, CEA,
Laboratoire en Informatique
Haute Performance pour
le Calcul et la simulation
Bruyères-le-Châtel, France
patrick.carribault@cea.fr

Abstract

Quantum computers exploit the particular behavior of quantum physical systems to solve some problems in a different way than classical computers. We are now approaching the point where quantum computing could provide real advantages over classical methods. The computational capabilities of quantum systems will soon be available in future supercomputer architectures as hardware accelerators called Quantum Processing Units (QPU). From optimizing compilers to task scheduling, the High-Performance Computing (HPC) software stack could benefit from the advantages of quantum computing. We look here at the problem of register allocation, a crucial part of modern optimizing compilers. We propose a simple proof-of-concept hybrid quantum algorithm based on QAOA to solve this problem. We implement the algorithm and integrate it directly into GCC, a well-known modern compiler. The performance of the algorithm is evaluated against the simple Chaitin-Briggs heuristic as well as GCC's register allocator. While our proposed algorithm lags behind GCC's modern heuristics, it is a good first step in the design of useful quantum algorithms for the classical HPC software stack.

Keywords— compiler optimization, quantum computing applications, register allocation, graph coloring

1 Motivation and goals

Quantum Computing has gained a lot of traction in the past decade due to its theoretical ability to outperform classical computers for some tasks. These include but are not limited to: integer factoring [1], linear system solving [2], database searching [3], and quantum system simulation [4]. While the concept of quantum computing is more than 40 years old, there has been a recent surge of interest in this domain. This is due to the recent realization of quantum devices capable of performing basic computations. The class of current quantum hardware is called NISQ for Noisy Intermediate Scale Quantum [5]. Right now these devices cannot outperform classical computers on general tasks due to many physical limitations, but it is believed that we might see a NISQ device that reaches this crucial milestone soon.

The potential advantages of quantum computing have led to an increasing interest from the High-Performance Computing community in the integration of quantum processing power in new HPC architectures. While full-scale fault-tolerant quantum computers might only become a reality in the far future, NISQ devices will soon be integrated into existing supercomputers. These devices are usually called Quantum Processing Units (QPU) and can be based on a variety of quantum systems, ranging from electron spin to photon polarization.

The computing power of QPUs could be leveraged in many ways when they are connected to a supercomputer. A path of particular interest to us is the HPC software stack i.e. the set of tools and software necessary for the efficient operation of HPC

applications. This software stack covers everything from development tools to runtime software and operating systems. Optimizing compilers are a crucial part of this stack as they are used to convert human-readable source code to architecture-specific machine instructions. Compilers are a good target for improvement through quantum computing as they are composed of a large set of computationally hard tasks, often related to combinatorial optimization problems. One such example would be register allocation which is closely related to the NP-hard problem of graph coloring [6].

With quantum accelerators soon to be integrated into some HPC clusters, we raise the question of how to leverage the capabilities of QPUs for the HPC software stack. Some of the many computationally hard tasks in this stack could benefit from the advantages of quantum computing. As a proof-of-concept for this idea, this paper proposes an integration of a hybrid quantum register allocation algorithm in GCC [7]. The performance of this approach will be evaluated by compiling HPC proxy apps for a classical x86_64 CPU architecture.

The main contributions of our work are as follows:

- We propose a register allocation method centered around a hybrid quantum optimization algorithm. We implemented the algorithm using a quantum computing framework and integrated it as a replacement for the graph coloring step of GCC's register allocation.
- We evaluate the quality of the solutions given by our algorithm by comparing them to a simple classical heuristic and GCC's allocation algorithm. We show improvement over the simple heuristic but do not yet reach the performance of GCC.

The rest of this paper is organized as follows: Section 2 lays down the background and discusses related work. Section 3 introduces the hybrid register allocation algorithm. Section 4 describes the implementation of the algorithm and its integration into an existing compiler. The performance of our algorithm on real HPC codes is studied in Section 5 and, finally, Section 6 concludes this paper.

2 Background

2.1 Register Allocation problem

Compilation is a long process that usually ends with "code generation" which consists of translating the program into architecture-specific instructions [8]. As part of code generation, the compiler needs to determine where to store each variable of the program. Modern computer architectures are composed of a variety of types of storage, each with its size and speed. Registers are small memory slots in the CPU which are extremely fast to access, most can be used directly in an operation without any load cycles.

While there can be any number of variables in a program, processors contain a finite number of registers. If two variables are in use (or "live") simultaneously, they cannot be stored in the same register. Optimizing compilers need to decide which variables to store in registers and which to store in system memory. If not all variables can be stored in registers, the ones stored in system memory are said to have been "spilled" [6].

A well-established method of register allocation is based on graph coloring [6, 9]. The core component of this technique is the interference graph: an undirected graph $G = (V, E)$ where nodes represent the variables of the program. There is an edge between two variables if and only if they are alive simultaneously in at least one statement. If the nodes representing two variables are not connected by an edge, the live ranges of the variables are distinct and we can allocate both to the same register. Register allocation boils down to finding a valid coloring of this graph using registers as colors.

However, the simple k-coloring problem only captures a very basic description of the register allocation problem. In modern CPU architecture, each variable in a program can only be allocated to a specific subset of all available registers. The problem that needs to be solved then becomes the list coloring of the interference graph with, for each variable v, L_v the list of registers that it can be allocated to. Each variable can either be allocated to a register or spilled to system memory, let us then define $L = \{mem\} \cup \bigcup_{v \in V} L_v$ the list of all spaces where a variable could be stored.

The final component we use to define register allocation here is a cost function $C_{\text{alloc}} \colon V \times L \mapsto \mathbb{R}$ that assigns an allocation cost for each possible combination of variables and register (or memory). We can then define register allocation as a combinatorial optimization problem detailed in Equation 1.

$$\min_{r \colon V \mapsto L} \quad \sum_{v \in V} C_{\text{alloc}}(v, r(v)) \tag{1a}$$

$$\text{s.t.} \quad \forall (v, v') \in E,\ r(v) \neq r(v') \text{ or } r(v) = mem \tag{1b}$$

$$\forall v \in V,\ r(v) \in L_v \cup \{mem\} \tag{1c}$$

We want to find an allocation $r \colon V \mapsto L$ that minimizes the overall allocation cost (1a) while allocating different registers to variables that are live at the same time or spilling them (1b). We also need to make sure that each variable is allocated to a register to which it can indeed be allocated (1c).

2.2 Quantum Computing for optimization

A category of problems that could benefit from the advantages provided by quantum computing is combinatorial optimization problems. The Quantum Approximate Optimization Algorithm (QAOA) [10] is a hybrid quantum algorithm aiming at providing improvements for combinatorial optimization tasks over classical counterparts –in terms of approximation ratio and/or execution time. Given a cost function $\mathcal{C} \colon \{0, 1\}^n \mapsto \mathbb{R}$, it tries to find an approximate minimum of $\mathcal{C}$ by searching for the ground state of a cost Hamiltonian $H_{\mathcal{C}}$ defined by $\langle x | H_{\mathcal{C}} | x \rangle = \mathcal{C}(x)$ for $x \in \{0, 1\}^n$. To construct it from the cost function a simple conversion can be obtained by taking $\mathcal{C}(x)$ and applying the map $x_i \leftrightarrow (I - \sigma_i^Z)/2$ where σ_i^Z is the Pauli Z gate applied on qubit i.

QAOA is a type of Variational Quantum Algorithm [11]: the core of the algorithm is a parameterized quantum circuit for which we will try to find the "best" parameters using a classical optimization loop. The state prepared by the circuit is $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ and is parameterized by $2p$ real angles $(\gamma_1, ..., \gamma_p)$ and $(\beta_1, ..., \beta_p)$. The circuit is made of p layers of gates derived from $H_{\mathcal{C}}$ and the angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$.

An approximate solution to the optimization problem is found by finding parameters $\boldsymbol{\gamma}^*$ and $\boldsymbol{\beta}^*$ that minimize the measured cost:

$$\mathcal{C}(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \boldsymbol{\gamma}, \boldsymbol{\beta} | H_{\mathcal{C}} | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle \tag{2}$$

This classical optimization loop can be realized using algorithms such as Nelder-Mead [12] or COBYLA [13] for example.

The final result of the algorithm is obtained by sampling the state $|\boldsymbol{\gamma}^*, \boldsymbol{\beta}^*\rangle$, a bitstring x that minimizes $\mathcal{C}$ will be found with high probability given a large enough p.

2.3 Related work

As mentioned in Section 2.1, the register allocation problem can be considered a variant of graph coloring. Specific quantum circuits and implementations to solve this problem using Variational Quantum Algorithms have been proposed in the past [14]. These quantum algorithms solving max-k-coloring have also been tested on NISQ devices [15], the results seem to suggest that QAOA for such optimization problems could be beneficial. However, register allocation as we defined is a different, more complex problem than max-k-coloring. It is closer to list-coloring where each node can be assigned to a specific list of colors while also adding constraints such as spilling [6].

While quantum algorithms for the specific problem of register allocation are a new field of research, there has been extensive work regarding classical algorithms that solve this optimization problem. Chaitin's method of allocating registers by coloring the interference graph [6] (and the following improvement by Briggs [9]) is foundational and still widely used in modern compilers. The basic principle of the algorithm is to successively remove nodes that satisfy $\deg(v) < k$ from the interference graph and add them to a stack. If there are no such nodes, nodes are spilled and removed from the graph until a node satisfies the condition. When all nodes have been removed, they are popped from the stack one by one and assigned an available color.

Integration of Quantum and High-Performance Computing is an emerging field of research. Some groundwork has been laid regarding the possible architectures and challenges of HPC/QC integration [16, 17]. The specific area of using quantum algorithms to solve problems in the HPC software stack is novel as far as we have been able to find. However, the idea of using quantum algorithms was already put forward in the case of quantum circuit compilation [18].

3 Quantum Algorithm for Register Allocation

3.1 Independent set extraction

If we look back at the register allocation problem as we defined it in Equation 1, it is quite clear that it is a complex combinatorial problem. Converting constraints (1b) and (1c) as well as the allocation cost in (1a) into a single global cost Hamiltonian to be used for QAOA is left as future work. Instead, we decided to find a simple quantum hybrid algorithm to solve the register allocation problem that would lead to smaller quantum circuits.

A known method to color graphs consists of sequentially extracting large independent sets from the graph and assigning the same color to each node of the set as there are no edges between them [19, 20]. We will apply this idea here with one specificity being that we will use QAOA to find large independent sets in the graph. To use QAOA we need a cost Hamiltonian for the weighted Maximum Independent Set (w-MIS) problem. The cost Hamiltonian is derived from the cost function described in Equation 3 where $x_v = 1 \iff v$ is in the independent set. We apply the map $x_v \to (I - \sigma_v^Z)/2$ to obtain the final Hamiltonian [21], which is described in Equation 4.

$$\mathcal{C}_{\text{w-MIS}}(\boldsymbol{x}) = \sum_{(v,v')\in E} x_v x_{v'} - \sum_{v\in V} w_v x_v \tag{3}$$

$$H_{\text{w-MIS}} = \sum_{(v,v')\in E} \sigma_v^Z \sigma_{v'}^Z + \sum_{v\in V} (w_v - \deg(v))\sigma_v^Z \tag{4}$$

One qubit will correspond here to one vertex of the graph and a qubit measured in state "1" will mean that the node is in the set.

As noted in Section 2.1, each variable can only be allocated to a specific subset of possible registers. In other words, each register t can only store variables that are in the set V_t defined as: $V_t = \{v \in V : t \in L_v\}$. For each register we will then try to find a large independent set in $G[V_t] = (V_t, E_t)$, the subgraph of G induced by $V_t \subseteq V$.

To minimize the total allocation cost we would like to allocate variables that would be costly to spill into memory first. To take this into account, we will extract for each register an independent set that maximizes the potential cost of spilling the variables to memory instead of storing them in the register. We define, for each register t, the cost of spilling a variable v as the cost of storing it in memory minus the cost of storing it in the register (Equation 5).

$$C_{\text{spill}}(v,t) = C_{\text{alloc}}(v, mem) - C_{\text{alloc}}(v,t) \tag{5}$$

The order in which the registers are allocated is decided using a cost function described in Section 3.2. Algorithm 1 describes the full register allocation algorithm with the step that is performed on the QPU in italics at line 6.

Algorithm 1 Greedy MIS QAOA

Require: Interference graph $G = (V, E)$, lists of available registers $(L_v)_{v\in V}$, spill cost C_{spill}.

1: $L := \bigcup_{v\in V} L_v$
2: Initialize $r(v) := mem$ for all $v \in V$
3: **while** $|L| > 0$ and $|V| > 0$ **do**
4: $\quad t' := \arg\min_{t\in L} c(t)$
5: $\quad$ **if** $|E_{t'}| > 0$ **then**
6: $\quad\quad$ *[On QPU] Use **QAOA** to find Max Independent Set S of $G[V_{t'}]$ with weights $w_v = C_{spill}(v, t')$*
7: $\quad$ **else**
8: $\quad\quad$ $S := V_{t'}$
9: $\quad$ **end if**
10: $\quad$ **for all** $v \in S$ **do**
11: $\quad\quad$ $r(v) := t'$
12: $\quad\quad$ Remove v from G
13: $\quad$ **end for**
14: $\quad$ Remove t' from all L_v
15: $\quad$ Update $L := \bigcup_{v\in V} L_v$
16: **end while**
17: **return** Allocation r

We never explicitly decide to spill a variable to memory here. Instead, spills happen when there are no registers left to which the variable could be allocated.

3.2 Allocation order

As mentioned earlier, the registers are assigned in a specific order related to what we call a "register allocation cost" $c\colon L \mapsto \mathbb{R}$. This cost should reflect how risky it would be to assign this register in terms of potential spills. Spills happen when a variable is not included in the independent set while having only one available register left. The allocation cost function c should reflect the probability of this situation happening for each register.

To take into account this risk, the cost function reflects how dense the subgraph $G[V_t] = (V_t, E_t)$ is. We want to penalize graphs with a large number of edges and few nodes as it could negatively impact the maximum size of the extracted independent set. We define this "register allocation cost" as follows:

$$c(t) = \frac{|E_t|}{|V_t|} \tag{6}$$

4 Adding a Quantum Computing routine to GCC

4.1 Architecture of GCC's compilation pipeline

The previous section described a hybrid quantum register allocation algorithm based on QAOA. The next step is to implement this algorithm and integrate it into a modern optimizing compiler. We decided to integrate our algorithm into GCC [7] as it is one of the most widely used compilers in both research and industrial settings.

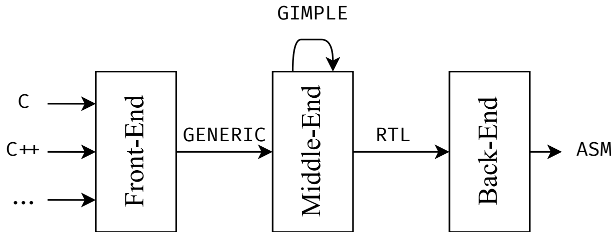

Figure 1: Simplified internal architecture of GCC compiler. The various Intermediary Representations are written over each arrow connecting the internal compiler blocks.

GCC's internal architecture can be split into three main parts (see Figure 1). The front-end parses and analyses code written in a specific language and converts it to a generic Intermediary Representation (IR) called GENERIC. The Middle-End first converts the GENERIC code into another IR called GIMPLE, the program represented this way will then go through many optimization and transformation passes. Most of the architecture-independent optimization will happen at this time. Finally, the program is converted into a final IR called Register Transfer Language (RTL) before entering the final steps of compilation. Those final steps are all inside the Back-End in which architecture-specific transformations and optimizations will happen before the final Assembly (ASM) code generation.

GCC offers an extensive plugin system that allows the integration of new transformation passes without having to directly modify the compiler. However, this plugin system is most suited for Middle-End passes and we could not use it to integrate our modified back-end register allocation pass. We directly integrated our hybrid algorithm into the existing pass that handles this task in GCC. This pass is called the Integrated Register Allocator (IRA) as it also performs tasks such as live range splitting and register coalescing on the fly during coloration.

To integrate our algorithm into GCC we override the coloration step of the IRA with an external routine as described in Figure 2. We do so for every interference graph in the compiled code ($\sim$1 per function/loop). As the number of qubits required to run the quantum algorithm scales with the number of nodes in the input interference graph, we want to work on as small graphs as possible. To do so the interference graph is first decomposed into its connected components[1] before running the algorithm on each component. The final allocation result is gathered from the results of all subgraphs.

Quantum devices have a limited number of available qubits that can be used to run a quantum algorithm. To take this constraint into account, we only use our algorithm to color the graphs that require fewer qubits than a fixed threshold. The maximum number of qubits required by our algorithm is given by the number of nodes of the largest subgraph induced by any available register t:

$$\mathrm{N}_{qubits}(V, L) = \max_{t \in L} |V_t| \tag{7}$$

Every individual subgraph that requires fewer qubits than a fixed threshold will be allocated using the hybrid quantum algorithm. The components that would require too many qubits will be colored by GCC's regular register allocator.

4.2 Quantum routine implementation

The core of our hybrid register allocation algorithm is the approximate solving of the Maximum Independent Set problem using QAOA. We implemented this quantum algorithm using a high-level quantum computing framework developed by Atos/EVIDEN called myQLM [23]. We chose this framework as it offers a generic quantum circuit programming model as well as integrated quantum circuit emulators. The main benefit of using a generic quantum computing framework is the ability to be hardware-agnostic. This allows us in theory to be able to run the quantum algorithm on any given quantum device that supports the gate-based model of quantum computation.

[1] A connected component of the graph G is a subgraph in which each pair of nodes is connected via a path. The list of all connected components of a graph can be obtained by Breadth-First Search [22].

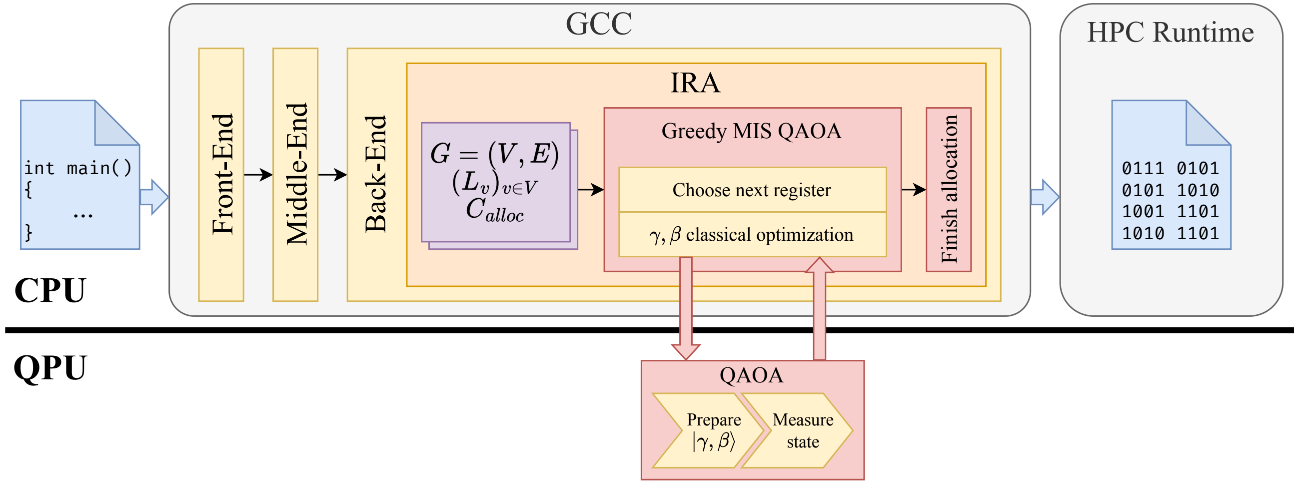

Figure 2: Architecture of the modified GCC compilation stack with the integration of the quantum-assisted register allocator. The coloration step of GCC's Integrated Register Allocator is overridden to use Algorithm 1. The quantum state preparation and measurement operations are executed on an external QPU.

Listing 1: myQLM Max Independent Set QAOA

```
# Initialize empty Hamiltonian
h = Observable(G.number_of_nodes())

# First term of the Hamiltonian
for u, v in G.edges():
  h.add_term(
    Term(1, "ZZ", [u, v])
  )

# Second term of the Hamiltonian
for u in G.nodes():
  h.add_term(
    Term(weights[u] - G.degree[u], "Z", [u])
  )

# Generate quantum circuit from h
circ = AnsatzFactory.qaoa_circuit(h, p)

# Run the QAOA optimization loop
qpu = get_default_qpu()
stack = ScipyMinimizePlugin() | qpu
job = circ.to_job(observable=h)
result = stack.submit(job)
```

To implement QAOA in myQLM, we could either manually write the whole quantum circuit or use built-in methods to generate said circuit. We chose the latter for simplicity and leave the problem of optimizing the quantum circuit for our specific problem to future work. The built-in QAOA circuit generation methods take as input the cost Hamiltonian and generate a "quantum job" that can then be run on any supported QPU. The quantum job can also be run using a quantum circuit emulator instead of a real QPU.

An example code excerpt that uses myQLM to implement QAOA for the Max Independent Set problem is provided in Listing 1.

5 Experimental results evaluation

5.1 Benchmark methodology

To validate the concept of integrating quantum algorithms into the classical HPC software stack, we measured the performance of the new hybrid register allocation algorithm. The metric we use to evaluate the performance of register allocation is the allocation cost computed by GCC which is similar to the one defined in Equation 1a. As no real quantum device was available to us, the quantum jobs were run using the `CLinalg` simulator included in myQLM which is based on state-vector simulation. We chose to use noiseless quantum circuit simulations for this study to serve as we meant it to be a proof-of-concept for hybrid quantum methods in the HPC software stack.

To evaluate the performance of our hybrid algorithm we used the modified GCC to compile three HPC proxy apps: miniFE, miniMD –both of which are part of the Mantevo project [24]– and LULESH [25]. We chose these codes as they are representative of some of the most common HPC use cases. As a point of comparison, we also performed the register allocation step of the compilation of these codes using the classical Chaitin-Briggs [6, 9] register allocation algorithm as well as GCC's allocation algorithm. GCC was used in all cases to generate the interference graph and to perform the other steps of compilation.

We chose to run the simulation with a fixed maximum number of qubits set to 13 to limit the total simulation time. The number of variables that could be allocated using our algorithm with 13 qubits are shown in Table 1. This limited number of available qubits means that only small interference graphs can be colored with the hybrid quantum algorithm. Modern CPU architectures usually contain a few dozen hardware registers which are more than the number of nodes of the processed graphs. This would make the coloring task trivial by assigning a different register to each node. To emphasize the differences in performance between our algorithm and the classical algorithms, we decided to restrict the set of available CPU registers by passing the `-ffixed-<reg>` option to the compiler for each unwanted register `<reg>`.

Table 1: Comparison of the number of variables handled by our algorithm with 13 available qubits.

Code	**Total variables**	**Variables handled by QAOA**
LULESH	11010	1343 (12.2%)
miniFE	23426	3700 (15.79%)
miniMD	25885	9376 (36.22%)

The global benchmark parameters were as follows:

- Codes: LULESH [25], miniFE , miniMD [24]
- Target CPU architecture: `x86_64` (`AMD64`)
- Used registers: `rax`, `rcx`, `rdx`, `xmm0`, `xmm1`, `xmm2`
- QAOA classical optimizer: COBYLA optimizer [13]
- Classical optimizer max number of iterations: 200
- QAOA initial parameters: Taken from a TQA schedule [26] with $\Delta t = 0.75$
- QAOA measurement shots[2]: 1000

5.2 Performance of the algorithm on HPC apps

We measured the allocation cost computed by GCC for our hybrid algorithm as well as the simple Chaitin-Briggs heuristic [6,9] and GCC's internal register allocation algorithm. The results of the comparison in terms of total allocation cost are given in Figure 3. We see that our hybrid quantum algorithm outperforms the basic Chaitin-Briggs allocation heuristic for all HPC codes to a varying degree. The difference is most noticeable on miniFE, this could be caused by many factors such as the shape and size of the encountered interference graphs.

We see however that GCC's internal allocation algorithm always outperforms our simple hybrid algorithm. This can be explained in multiple ways: first of all, the idea of greedily extracting the largest independent graph could be largely suboptimal compared to optimal coloring. Also, GCC performs other optimizations during the coloring step of allocation that could explain another part of the difference in performance. Finally, there is no clear theoretical proof of a clear advantage provided by QAOA for optimization problems, its potential inherent weaknesses may take part in the lackluster performance. This opens many paths forward to implement our method to address these points, these improvements are left as future work.

6 Conclusion and Future Work

In this work, we proposed an application of Quantum Computing for High-Performance Computing in the form of a hybrid quantum register allocation algorithm based on QAOA. We allocate all variables of a code to registers by extracting Maximum Independent Sets from the interference graph using QAOA. We implemented this algorithm using the high-level myQLM quantum computing framework and integrated it directly inside GCC's register allocation pass.

We demonstrated improved register allocation performance over the basic Chaitin-Briggs allocation algorithm on three sample HPC codes: LULESH, miniFE and miniMD. The experiments were run on a noiseless quantum circuit simulator with a small number of available qubits. The performance of the algorithm was however not as good as the internal GCC register allocator. This can be explained in multiple ways but it raises the question of the practical interest of the current version of our algorithm.

We believe there are multiple paths forward regarding useful applications of quantum algorithms for the HPC software stack. First of all, for the problem of register allocation, we could employ a column generation method on a Linear Programming version of the problem which then uses a quantum algorithm for the pricing sub-problem. Such a method has already been demonstrated for the problem of minimum vertex coloring [27] using a neutral atom-based QPU. Other ideas could revolve around embedding the constraints of the optimization problem into the quantum circuit [28]. We could also leverage a larger amount of qubits to cover a bigger percentage of the compiled code as only a small part of the codes is covered now (see Table 1). Finally, other interesting hard optimization problems in the HPC software stack may be better suited for hybrid quantum methods. We believe in particular that many scheduling tasks are present in various tools and runtime software that could benefit from quantum computing.

[2] The number of shots is the number of times each quantum measurement is repeated to obtain a probability distribution.

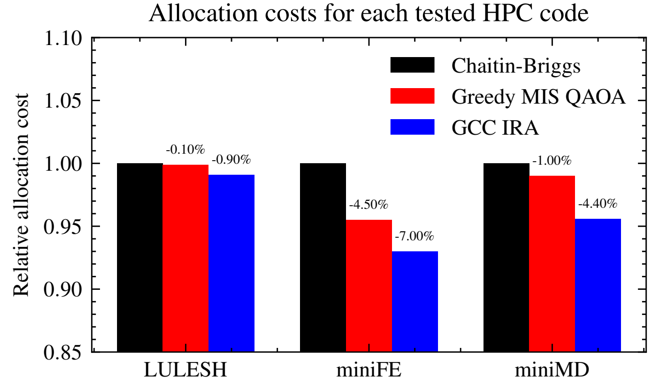

Figure 3: Allocation cost computed by GCC for each register allocation method and each evaluated code. The cost is taken as the average over 3 simulations.

Acknowledgments

This work is part of HQI initiative (www.hqi.fr) and is supported by France 2030 under the French National Research Agency award number “ANR-22-PNCQ-0002”.

References

[1] P. Shor, “Algorithms for Quantum Computation: Discrete Logarithms and Factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Nov. 1994, pp. 124–134, https://doi.org/10.1109/sfcs.1994.365700.

[2] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Solving Linear Systems of Equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, Oct. 2009, http://arxiv.org/abs/0811.3171.

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, Jul. 1996, pp. 212–219.

[4] R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, Jun. 1982, https://doi.org/10.1007/BF02650179.

[5] J. Preskill, "Quantum Computing in the NISQ Era and Beyond," *Quantum*, vol. 2, p. 79, Aug. 2018, https://quantum-journal.org/papers/q-2018-08-06-79/.

[6] G. J. Chaitin, "Register Allocation & Spilling via Graph Coloring," *ACM SIGPLAN Notices*, vol. 17, no. 6, pp. 98–101, Jun. 1982, https://doi.org/10.1145/872726.806984.

[7] "GCC, the GNU Compiler Collection - GNU Project."

[8] S. Muchnick and M. and Associates, *Advanced Compiler Design Implementation*. Morgan Kaufmann, Aug. 1997.

[9] P. Briggs, "Register Allocation via Graph Coloring," Ph.D. dissertation, Rice University, Apr. 1992, https://scholarship.rice.edu/handle/1911/96426.

[10] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," Nov. 2014.

[11] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational Quantum Algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, Sep. 2021, http://arxiv.org/abs/2012.09265.

[12] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, Jan. 1965, https://doi.org/10.1093/comjnl/7.4.308.

[13] M. J. D. Powell, "A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation," in *Advances in Optimization and Numerical Analysis*, ser. Mathematics and Its Applications, S. Gomez and J.-P. Hennart, Eds. Dordrecht: Springer Netherlands, 1994, pp. 51–67, https://doi.org/10.1007/978-94-015-8330-5_4.

[14] Y.-H. Oh, H. Mohammadbagherpoor, P. Dreher, A. Singh, X. Yu, and A. J. Rindos, "Solving Multi-Coloring Combinatorial Optimization Problems Using Hybrid Quantum Algorithms," Dec. 2019.

[15] Z. Tabi, K. H. El-Safty, Z. Kallus, P. Hága, T. Kozsik, A. Glos, and Z. Zimborás, "Quantum Optimization for the Graph Coloring Problem with Space-Efficient Embedding," *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 56–62, Oct. 2020, http://arxiv.org/abs/2009.07314.

[16] K. A. Britt and T. S. Humble, "High-Performance Computing with Quantum Processing Units," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 13, no. 3, pp. 39:1–39:13, Mar. 2017.

[17] T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum Computers for High-Performance Computing," *IEEE Micro*, vol. 41, no. 5, pp. 15–23, Sep. 2021.

[18] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *Quantum*, vol. 3, p. 140, May 2019.

[19] M. Chams, A. Hertz, and D. de Werra, "Some Experiments with Simulated Annealing for Coloring Graphs," *European Journal of Operational Research*, vol. 32, no. 2, pp. 260–266, Nov. 1987, https://www.sciencedirect.com/science/article/pii/S0377221787801480.

[20] Q. Wu and J.-K. Hao, "Coloring Large Graphs Based on Independent Set Extraction," *Computers & Operations Research*, vol. 39, no. 2, pp. 283–290, Feb. 2012, https://www.sciencedirect.com/science/article/pii/S0305054811000979.

[21] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, 2014.

[22] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient Algorithms for Graph Manipulation," *Communications of the ACM*, vol. 16, no. 6, pp. 372–378, Jun. 1973, https://doi.org/10.1145/362248.362272.

[23] "Quantum Application Toolset — myQLM Documentation Documentation."

[24] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, vol. 3, 2009.

[25] I. Karlin, J. Keasler, and R. Neely, "LULESH 2.0 Updates and Changes," Livermore, CA, Aug. 2013.

[26] S. H. Sack and M. Serbyn, "Quantum annealing initialization of the quantum approximate optimization algorithm," *Quantum*, vol. 5, p. 491, Jul. 2021.

[27] W. d. S. Coelho, L. Henriet, and L.-P. Henry, "A quantum pricing-based column generation framework for hard combinatorial problems," Jan. 2023.

[28] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz," *Algorithms*, vol. 12, no. 2, p. 34, Feb. 2019, http://arxiv.org/abs/1709.03489.