Usage example for pme-fusion

To illustrate how to operate the `pme_fusion` library and `gen_invariants.pl` script, we will demonstrate how to input a simple example, namely a Cholesky factorizaiton fused with a lower-triangular solve. The operations involved are finding a $\widetilde{L} = CHOL(\hat{L})$ such that $\widetilde{L}\widetilde{L}^T = \hat{L}$ and finding $\widetilde{B}$ such that $\hat{L}\widetilde{B} = \hat{B}$, for input matrices $\hat{L}$ and $\hat{B}$, which are overwritten by $\widetilde{L}$ and $\widetilde{B}$. The triangular solve is notated with $\hat{L}$ since, if we were not fusing these operations, the L that is in input to the solve would be the $\widetilde{L}$ from the Cholesky factorization.

Once you have selected a partitioning for each operand (this choice is often dictated by the structure of your problem and data — for example, L should be partitioned into a 2×2 grid to exploit its symmetric structure), you can begin to form the input to the program. This process begins by taking your problem specifications and substituting in the partitioned variables. For example, once we've chosen to partition L in a 2×2 form and B into a 2×1 grid, we have the equations.

$$\left(\begin{array}{c|c} \widetilde{L}_{TL} & 0 \\ \hline \widetilde{L}_{BL} & \widetilde{L}_{BR} \end{array}\right)\left(\begin{array}{c|c} \widetilde{L}_{TL}^T & \widetilde{L}_{BL}^T \\ \hline 0 & \widetilde{L}_{BR}^T \end{array}\right) = \left(\begin{array}{c|c} \hat{L}_{TL} & * \\ \hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)$$

and

$$\left(\begin{array}{c|c} \hat{L}_{TL} & 0 \\ \hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)\left(\begin{array}{c} \widetilde{B}_T \\ \hline \widetilde{B}_B \end{array}\right) = \left(\begin{array}{c} \hat{B}_T \\ \hline \hat{B}_B \end{array}\right).$$

In order to be able to search for loop invariants, we need to translate these equations into an single expression where each region contains an equation of the from $\widetilde{M_R} = \ldots$. In these examples, we can begin by carrying out a multiplication

$$\left(\begin{array}{c|c} \widetilde{L}_{TL}\widetilde{L}_{TL}^T & * \\ \hline \widetilde{L}_{BL}\widetilde{L}_{TL}^T & \widetilde{L}_{BL}\widetilde{L}_{BL}^T + \widetilde{L}_{BR}\widetilde{L}_{BR}^T \end{array}\right) == \left(\begin{array}{c|c} \hat{L}_{TL} & * \\ \hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)$$

and multiplying by an inverse

$$\left(\begin{array}{c} \widetilde{B}_T \\ \hline \widetilde{B}_B \end{array}\right) = \left(\begin{array}{c} \hat{L}_{TL}^{-1}\hat{B}_T \\ \hline -\hat{L}_{BR}^{-1}\hat{L}_{BL}\hat{L}_{TL}^{-1}\hat{B}_T + \hat{L}_{BR}\hat{B}_B \end{array}\right)$$

Within the equations for the Cholesky factorization, we can identify (with some raarrangement required on the bottom right) regions of the form $\widetilde{L}_R\widetilde{L}_R^T = X$, which we will instead write as $\widetilde{L}_R = CHOL(X)$ so that we can express things in the requisite format. In

addition, we must isolate $\widetilde{L}_{BL}$ with a solve. This leaves us with the expression

$$\left(\begin{array}{c|c} \widetilde{L}_{TL} & * \\ \hline \widetilde{L}_{BL} & \widetilde{L}_{BR} \end{array}\right) = \left(\begin{array}{c|c} CHOL(\hat{L}_{TL}) & * \\ \hline \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & CHOL(\hat{L}_{BL} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T}) \end{array}\right)$$

which has the form we want once we move the equalities into each region.

For the lower-triangular solve, we can observe that expression for $\widetilde{B}_T$ appears in the expression for $\widetilde{B}_B$. Then, we can factor out the $\hat{L}_{BR}$ and rewrite all the $\widetilde{B}_R = L_S^{-1}Y$ expressions to $\widetilde{B}_R = TRSM(\hat{L}_S, Y)$ to obtain

$$\left(\begin{array}{c} \widetilde{B}_T = TRSM(\hat{L}_{TL}, \hat{B}_T) \\ \hline \widetilde{B}_B = TRSM(\hat{L}_{BR}, \hat{B}_B - \hat{L}_{BL}\widetilde{B}_T) \end{array}\right).$$

We now need to convert these expressions into a form that the `pme_fusion` software can operate on. One transformation that you need to make is to break expressions that can be partially computed (those where there is a subexpression which could overwrite the input while allowing the computation to proceed) into *tasks*. For example, we can convert $\widetilde{L}_{BR} = CHOL(\hat{L}_{BR} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T})$ into the two tasks $L_{BR,0} \coloneqq \hat{L}_{BR} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T}$ and $\widetilde{L}_{BR} \coloneqq_O CHOL(L_{BR,0})$. The second task is written $\coloneqq_O$ because it is a task that performs the underlying operation of the algroithm, in this case, a Cholesky factorizaiton. These tasks need to be marked so the algorithm can operate correctly.

There will be cases where you can split an expression into tasks in multiple different ways. For example, the expression $\widetilde{L}_{BL} = -\hat{L}_{BR}^{-1}\hat{L}_{BL}\widetilde{L}_{BL}$ from equations for a lower-triangular inverse could be written either as $A_{BL,0} \coloneqq -\hat{L}_{BR}^{-1}\hat{L}_{BL}$ and $\widetilde{L}_{BL} \coloneqq A_{BL,0}\widetilde{L}_{TL}$ or as $A_{BL,0} \coloneqq -\hat{L}_{BL}\widetilde{L}_{TL}$ and $\widetilde{L}_{BL} \coloneqq \hat{L}_{BR}^{-1}A_{BL,0}$. To solve this problem without generating duplicate solutions, we have the $A_{R,(n,x)}$ syntax for partially-updated regions that do not need to be in a strict sequential order, as well as the $(A_{R_1} \vee A_{R_2} \ldots \vee B_{R_k})$ syntax for specifying operations that could read from multiple different inputs. Using this syntax, the tasks for the multiplication are $L_{BL,(0,a)} \coloneqq -\hat{L}_{BR}^{-1}(\hat{L}_{BL} \vee L_{BL,(0,b)})$ and $L_{BL,(0,b)} \coloneqq (\hat{L}_{BL} \vee L_{BL,(0,a)})\widetilde{L}_{TL}$.

We need to note that the operands to a $\vee$ must be memory states (translation into a more software-friendly expression will not work for $(-\hat{A}_R \vee A_{R,(0,b)})$). Similarly, if the whole expression can be split into a series of either-order tasks, like in the example shown above, there is no need to explicitly define $\widetilde{A}_R4$, since it will be implicitly defined by the logic of the algorithm (it is fine to refer to $\widetilde{A}_R$ in other expressions in this case). If you want to explicitly include $\widetilde{A}$, use a `comes_from` task to do so. If the expression that was split is a subexpression that needs the entire split part to be computed, you should represent this as (for example) $A_{R,1} \coloneqq F((A_{R,(0,a)} \wedge A_{R,(0,b)}), \ldots)$ (the exact representation is arbitrary).

Once you have converted each equation into one or more tasks, you can translate the tasks for each operation you want to fuse into a task list that can be read by the `pme_fusion` library. The translation operates as follows:

- Map each region (or other object) to an atom, which is either an alphanumeric (with underscores) symbol that starts with a lowercase letter or an arbitrary string in single quotes. For our examples, we will be mapping A_R to `a_r`

- Map the memory state $\hat{A}_R$ to `hat(a_r)`, $\widetilde{A}_R$ to `tilde(a_r)`, $A_{R,n}$ to `during(a_r, n)`, and $A_{R,(nx)}$ to `during(a_r, n, x)` (where `x` is an atom)

  Regions that are only read during the whole fusion problem and never written do not need memory-state specifiers and can be treated as constants However, if they are annotated with `hat(Region)`, the system will assume that they come from an object you plan to iterate over and restrict its analysis to account for this.

- Write the task $X \coloneqq f(Y)$ as (translating the components) `eq(X, f(Y))`.

  In general, the second argument to `eq` can be an arbitrarily nested sequence of functions (whose names are atoms) and atoms, so long as functions are not moved into a `hat()`, `during()`, or `tilde()`. The first argument must be a memory state. For example, the first task in the bottom right of the Cholesky factorization can be written as `eq(during(a_br, 0), sub(hat(a_br), mul(tilde(a_bl), tr(tilde(a_bl)))))`.

- If a task is an operation task (written $\coloneqq_O$), use `op_eq` and not `eq` in the translation

- If you have an equation that writes to multiple matrices, pick one as the output and use the task `comes_from(Region, Expr)` to represent a noop (see `examples/general_inverse.pl` and `examples/general_inverse.txt` for an example)

- Collect the tasks for each operation you want to fuse into a list (which are written in `[]`s)

- Sequence these lists into a larger list, which in the initial input to the library

Once you have such a list, you can present it to the software, which will print all fusable collections of loop invariants for the given expressions, as well as warning you about common issues with the inputs. The simple way to present such an input is to call the `gen_invariants.pl` script, which will read a software-friendly list of task lists from a given file (or standard input) and print the results of the analysis. The outer list in the file must be followed by a period (`.`), and can contain Prolog-style comments, which begin with `%`. For the Cholesky and linear solve, a potential input file, which can be found in `examples/chol_trsm.txt` has the following form:

As an example, we will show how the PMEs given above for the Cholesky factorization and map to task expressions

$$\left(\begin{array}{c|c} \widetilde{L}_{TL} \coloneqq_O CHOL(\hat{L}_{TL}) & * \\ \hline \widetilde{L}_{BL} \coloneqq \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & \begin{array}{l} L_{BR,0} \coloneqq \hat{L}_{BR} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T}; \\ \widetilde{L}_{BR} \coloneqq_O CHOL(L_{BR,0}) \end{array} \end{array}\right)$$

$\rightarrow$

$$\left(\begin{array}{c|c} \texttt{op\_\_eq(tilde(l\_\_tl), chol(hat(l\_\_tl)))} & * \\ \hline \texttt{eq(tilde(l\_\_bl), trsm(tilde(l\_\_tl), hat(l\_\_bl)))} & \begin{array}{c} \texttt{eq(during(l\_\_br, 0), sub(hat(l\_\_br), mul(tilde(l\_\_bl), tr(tilde(l\_\_bl)))))} \\ \texttt{op\_\_eq(tilde(l\_\_br), chol(during(l\_\_br, 0)))} \end{array} \end{array}\right)$$

$$\left(\begin{array}{c}\widetilde{B}_T \coloneqq TRSM(\hat{L}_{TL}, \hat{B}_T) \\ \hline B_{B,0} \coloneqq \hat{B}_B - \hat{L}_{BL}\widetilde{B}_T \\ \widetilde{B}_B = TRSM(\hat{L}_{BR}, B_{B,0})\end{array}\right)$$

$$\rightarrow$$

$$\left(\begin{array}{c}\texttt{op\_eq}(\texttt{tilde}(\texttt{b\_t}), \texttt{trsm}(\texttt{hat}(\texttt{l\_tl}), \texttt{hat}(\texttt{b\_t}))) \\ \hline \texttt{eq}(\texttt{during}(\texttt{b\_b}, 0), \texttt{sub}(\texttt{hat}(\texttt{b\_b}), \texttt{mul}(\texttt{hat}(\texttt{l\_bl}), \texttt{tilde}(\texttt{b\_b})))) \\ \texttt{op\_eq}(\texttt{tilde}(\texttt{b\_b}), \texttt{trsm}(\texttt{hat}(\texttt{l\_br}), \texttt{during}(\texttt{b\_1b}, 0)))\end{array}\right)$$

If we put these expressions together, we obtain the following input file, which can be found in `examples/chol_trsm.txt`

```
% Cholesky + Lower triangular solve
[
 % Cholesky
 [
     op_eq(tilde(l_tl), chol(hat(l_tl))),

     eq(tilde(l_bl), trsm(tilde(l_tl), hat(l_bl))),

     eq(during(l_br, 0), sub(hat(l_br), mul(tilde(l_bl), tr(tilde(l_bl))))),
     op_eq(tilde(l_br), chol(during(l_br, 0)))
 ],
 % Solve
 [
     op_eq(tilde(b_t), trsm(hat(l_tl), hat(b_t))),

     eq(during(b_b, 0), sub(hat(b_b), mul(hat(l_bl), tilde(b_t)))),
     op_eq(tilde(b_b), trsm(hat(l_br), during(b_b, 0)))
 ]
].
```

The other option is to call the `pme_fusion` library directly. This is very useful for cases where constructing the set of tasks by hand is tedious — an example of this is available in `examples/tze_meng_sym_multiply.pl`, which shows the fusion of $\widetilde{C}((AM) + (AM)^T)$ with $\widetilde{C} = \hat{C} - MM^T$ The typical way call the library is to use the `gen_invariants(TaskLists)` function, which performs the same operations as the script of the same name.

For a lower-level interface, which allows more control over filtering and printing, see the library documentation, especially `make_pmes` and `fused_invariants`.

Once the software has found loop invariants, you can derive the algorithm by **TODO describe this?**.