Parallel Minimum degree ordering

When factoring sparse matrices, new non-zero entries are created. This phenomenon, known as "fill-in", has a significant impact on performance. This amount of extra non-zero entries created depends on column ordering. Determining good column orderings is therefore an extremely precomputing phase of any sparse matrix factorization.

1 Elimination game

The **elimination game** consists in simulating the factorization itself while monitoring newly created non-zero entries. Based on this knowledge, it is possible to design heuristics intenting to reduce the amount of fill-in.

Sparse matrix row-column dependencies are represented using a graph structure $\mathcal{G} = (V, E)$. Each node $v \in V$ corresponds to a column in the sparse matrix. Every edge $(u, v) \in E$ between two nodes u and v represents a dependency between these two columns, i.e. a non zero entry in (u, v) and (v, u) entries of the sparse matrix.

At every step, the elimination game processes as follows:

1. Pick a node v and eliminate it.
2. Create a clique between all nodes adjacent to v

2 Minimum degree algorithm

2.1 Parallelization schemes

- algorithms not explicitely changing the graph
- algorithms changing the graph

3 Finding indistinguishable nodes when computing reachable set

4 Performance

Input: v is the node eliminated at step s. R_v is its reachable set.
marker and *label* arrays, *tag*

$label(v) = s$
$indistCount = 0$
$tag = tag + 1$
$tag_v = tag$
forall the *node* $u \in R_v$ **do**
    $mask(u) = tag_v$
end
forall the *node* $t \in R_v$ **do**
    $tag = tag + 1$
    $indist, deg(t) \leftarrow$
    $update\_degree(t, v, deg(v), label, marker, tag, tag_v, mask)$
    if *indist* **then**
        $s = s + 1$
        $label(t) = s$
        $indistCount \leftarrow indistCount + 1$
    end
end
forall the *node* $t \in R_v$ **do**
    if $label(t) = 0$ **then**
        $deg(t) \leftarrow deg(t) - indistCount$
    end
end

Algorithm 1: Sketch of the MDO algorithm calling *update_degree*

Input:
1. u, node of which we're computing the reachable set (starting point of the exploration).
2. v, node eliminated at current step.
3. $deg(v)$, degree of v.
4. $label$, array of size n indicating if a node has been labeled or not.
5. $marker$, array of size n used to mark explored nodes with value tag.
6. tag_v, special tag value used to mark nodes in R_v.
7. $mask$, array of size n used to mark nodes in R_v with tag_v.

Output:
1. $indist$, boolean indicating if v and u are indistinguishable.
2. $\bar{deg}(u)$, updated degree of u after the elimination of v.

```
deg̅(u) ← deg(v) − 1
explore ← {u}
indist ← true
count ← 1
forall the node t in explore do
    forall the node x in Adj_t do
        if marker(x) ≠ tag then
            if label(x) ≠ 0 then
                if x ≠ v then
                    explore ← explore ∪ {x}
                end
            end
            else
                if mask(x) ≠ tag_v then
                    indist ← false
                    deg̅(u) ← deg̅(u) + 1
                end
                else
                    count ← count + 1
                end
            end
            marker(x) = tag
        end
    end
end
if indist = true AND count + 1 ≠ deg(v) then
    indist ← false
end
```

Algorithm 2: *update_degree*

P	Reach	Reach & comp.	Reach, comp. & mass.	Explicit	Explicit & mass.
1		4.9444	5.2108	3.5210	0.2097
4		1.9232	2.1678	1.4694	0.2074
8		1.1575	1.4402	0.9910	0.1702
16		0.8842	1.3877	0.8215	0.1937
24		0.8289	1.3861	0.8287	0.2133