TASK-BASED SPARSE CHOLESKY SOLVER ON TOP OF RUNTIME SYSTEM

Iain S. Duff, Jonathan D. Hogg and **Florent Lopez**

Sparse Days, 2016

Rutherford Appleton Laboratory
NLAFET Project

Objective

Solve $Ax = b$, where A is large and sparse, on modern architectures.

Exploiting modern platforms is challenging:

- Multicore processors and deep memory hierarchy.
- Heterogeneous e.g. CPU & GPU or Xeon Phi.
- Distributed memory systems.

Use Direct Method: Sparse Cholesky factorization $A = LL^T$

▲ Numerically robust and general purpose

▼ High memory usage and computational cost

The numerical factorization of A rely on an *elimination tree* expressing data dependencies in the factor L. Each node, referred to as *supernode*, is a dense (lower trapezoidal) submatrix of the factor L.
The tree is traversed in a topological order, and each node is factorised using dense algorithm.
Updates between node is handle using a supernodal scheme i.e. updates are applied directly to the target supernode.

Sequential Task Flow (STF) programming model:

- Tasks are submitted to the runtime system following the sequential algorithm.
- The runtime analyses manipulated data and infers task dependencies in order to ensure the sequential consistency of the parallel code.
- The DAG is executed via a dynamic scheduling of the (ready) tasks on the architectures.
- The runtime may be capable of automatically handling the data transfer across the architecture.
- Superscalar analysis in processors: dependency detection between instructions in order to issue them in parallel.

The STF Sparse Cholesky Factorization

```
forall nodes snode in post-order
   ! allocate data structures
   call alloc(snode)
   ! initianlize node structure
   call submit(init, snode:W)
end do

forall nodes snode in post-order

  ! factorize node
  do k=1..n in snode
    ! factorize diagonal block
    call submit(factorize, snode:R, blk(k,k):RW)

    do i=k+1..m in snode
       ! perform triangular solve w.r.t diag block
        call submit(solve, blk(k,k):R, blk(i,k):RW)
    end do

    do j=k+1..n
       do i=k+1..m
          call submit(update, blk(j,k):R, blk(i,k):R, blk(i,j):RW)
       end do
    end do

    forall ancestors(snode) anode
      ! udpate ancestor nodes
      call submit(update_between, snode:R, anode:RW)
    end do

  end do
end do
```

Parametrized Task Graph (PTG) programming model:

- Uses a compact representation of the DAG which is problem size independent.
- The dataflow between tasks is explicitly encoded (i.e. task dependencies are explicitly given).
- Under some hypothesis, the dataflow information can be automatically extracted from the sequential code using a dedicated compiler: not in our case unfortunately.