TASK-BASED SPARSE CHOLESKY SOLVER ON TOP OF RUNTIME SYSTEM

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

PMAA'16, 2016

Rutherford Appleton Laboratory
NLAFET Project

Objective

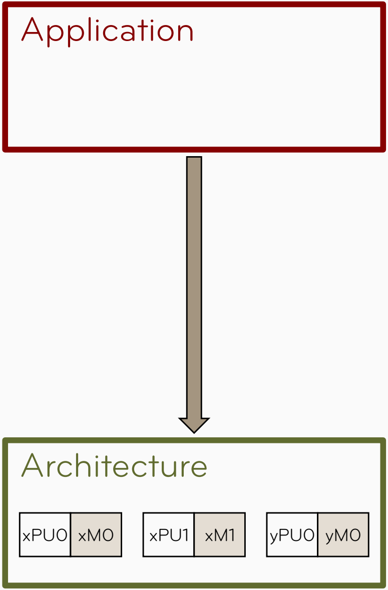
Solve $Ax = b$, where A is large and sparse, on modern architectures.

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

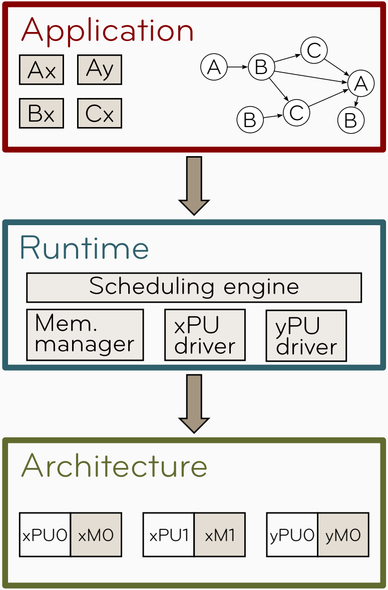
- ▲ Numerically robust and general purpose
- ▼ High memory usage and computational cost

Exploiting modern platforms is challenging:

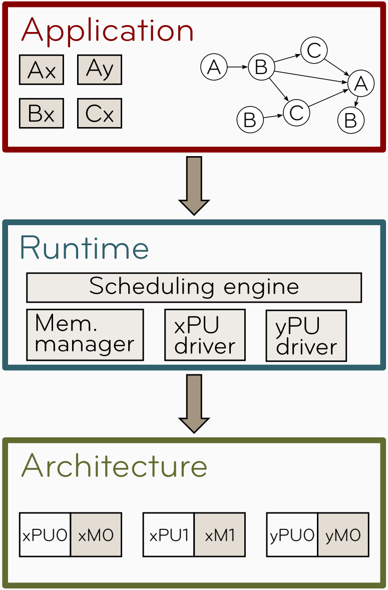
- Multicore processors and deep memory hierarchy.
- Heterogeneous e.g. CPU & GPU or Xeon Phi.
- Distributed-memory systems.



- The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which.
  - programming costs.
  - is difficult to maintain and update.
  - is prone to (performance) portability issues.



- The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which.
  - programming costs.
  - is difficult to maintain and update.
  - is prone to (performance) portability issues.
- runtimes provide an abstraction layer that hides the architecture details.

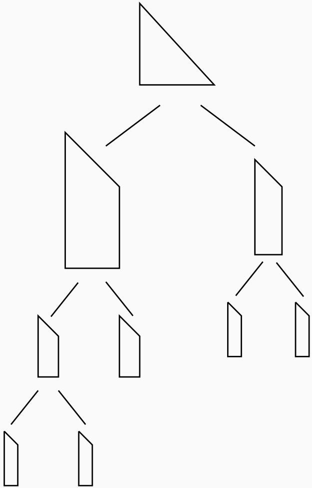


- The classical approach is based on a mixture of technologies (e.g., MPI+OpenMP+CUDA) which.
  - programming costs.
  - is difficult to maintain and update.
  - is prone to (performance) portability issues.
- runtimes provide an abstraction layer that hides the architecture details.
- the workload is expressed as a DAG of tasks.

In numerical factorization of A the *elimination tree* expresses data dependencies in the factor L. Each node, referred to as *supernode*, is a dense lower trapezoidal submatrix of L.

The tree is traversed in a topological order, and each node is factorized using dense Cholesky algorithm.

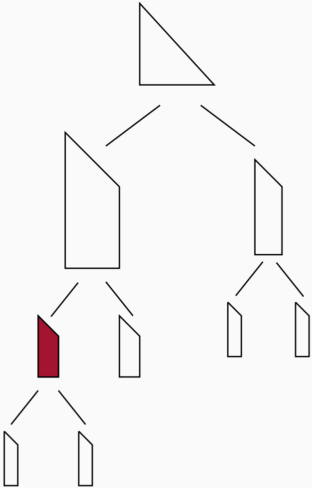
Updates between node are handled using a supernodal scheme i.e. updates are applied directly to the target supernodes.



In numerical factorization of A the *elimination tree* expresses data dependencies in the factor L. Each node, referred to as *supernode*, is a dense lower trapezoidal submatrix of L.

The tree is traversed in a topological order, and each node is factorized using dense Cholesky algorithm.

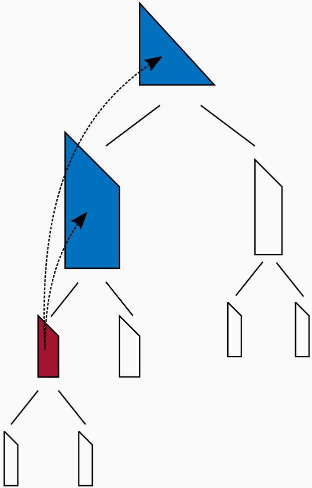
Updates between node are handled using a supernodal scheme i.e. updates are applied directly to the target supernodes.



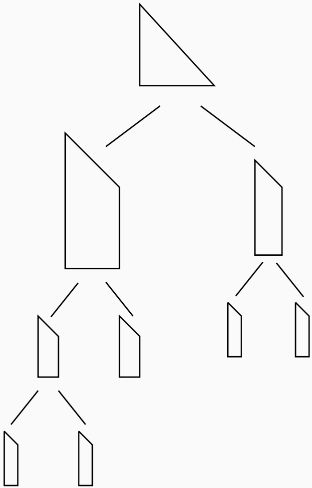
In numerical factorization of A the *elimination tree* expresses data dependencies in the factor L. Each node, referred to as *supernode*, is a dense lower trapezoidal submatrix of L.

The tree is traversed in a topological order, and each node is factorized using dense Cholesky algorithm.

Updates between node are handled using a supernodal scheme i.e. updates are applied directly to the target supernodes.

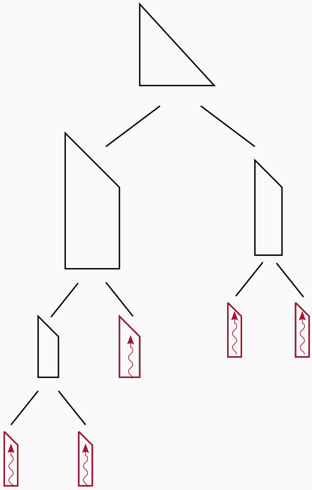


Sources of parallelism in the elimination tree:



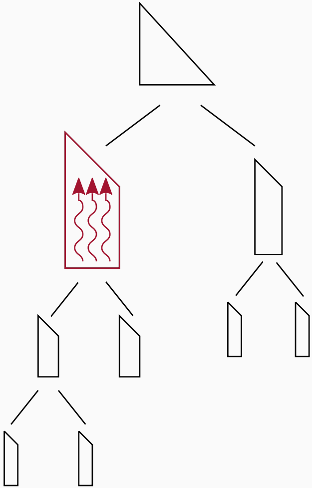
Sources of parallelism in the elimination tree:

- Tree parallelism: Supernode in independent branches can be processed concurrently.

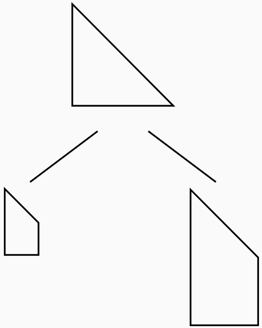


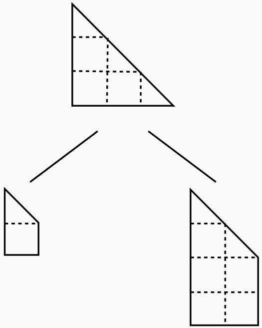
Sources of parallelism in the elimination tree:

- Tree parallelism: Supernode in independent branches can be processed concurrently.
- Node parallelism: When a supernode is large enough, it may be processed in parallel.

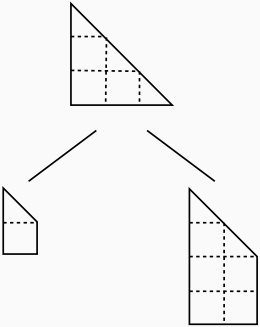


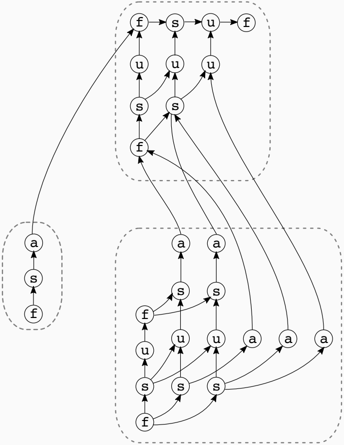
Task-based Sparse Cholesky factorization





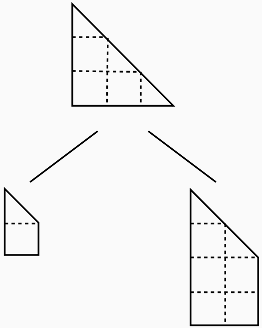
Supernodes are partitioned into square blocks (`nb x nb`) on which operations are applied (`factorize`, `solve`, `update`, `update_between`).

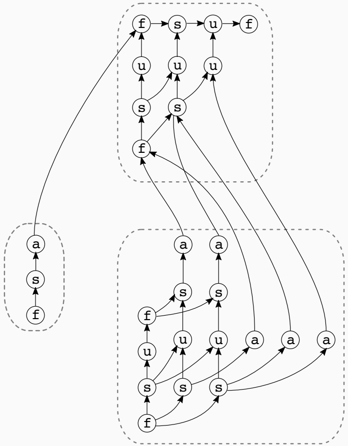




Supernodes are partitioned into square blocks (**nb x nb**) on which operations are applied (**factorize**, **solve**, **update**, **update_between**). The DAG replaces the elimination tree for representing the dependencies.

Task-based Sparse Cholesky factorization

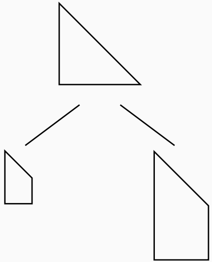




Supernodes are partitioned into square blocks (`nb x nb`) on which operations are applied (`factorize`, `solve`, `update`, `update_between`). The DAG replaces the elimination tree for representing the dependencies.
Implemented in the HSL package MA87.

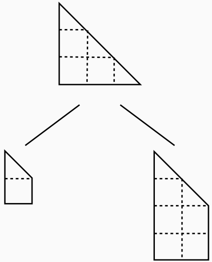
Task-based Sparse Cholesky factorization

```
forall nodes snode in post-order
   call alloc(snode) ! allocate data structures

   call init(snode) ! initianlize node structure
end do

forall nodes snode in post-order
  ! factorize node
   call factorize(snode)

   ! update ancestor nodes
   forall ancestors(snode) anode
     call update_btw(snode, anode)
   end do

  end do
end do







!
```



Task-based Sparse Cholesky factorization

```
forall nodes snode in post-order
  call alloc(snode) ! allocate data structures

  call init(snode) ! initianlize node structure
end do

forall nodes snode in post-order
  ! factorize node
  do k=1..n in snode
    call factorize(blk(k,k)) ! factorize block
    ! solve block
    do i=k+1..m in snode
       call solve(blk(k,k), blk(i,k))
    end do
    ! update block
    do j=k+1..n in snode
      do i=k+1..m in snode
        call update(blk(j,k), blk(i,k), blk(i,j))
      end do
    end do

    ! update ancestor nodes
    forall ancestors(snode) anode
      do j=k+1..p(anode) in snode
        do i=k+1..m in snode
           call update_btw(blk(j,k), blk(i,k),
                           a_blk(rmap(i), cmap(j)))
        end do
      end do
    end do

  end do
end do
```



The Sequential Task Flow Model

Sequential Task Flow (STF) programming model:

- In the parallel code, tasks are submitted to the runtime system following the sequential algorithm.
- The runtime analyses the manipulated data and infers task dependencies in order to ensure the sequential consistency of the parallel code.
- Superscalar analysis in processors: dependency detection between instructions in order to issue them in parallel.
- The DAG is executed via a dynamic scheduling of the (ready) tasks on the architecture.
- The runtime may be capable of automatically handling the data transfers on the architecture (e.g. CPU/GPU memory nodes).