Winter School

Multi-Threaded Hybridization of the OpenFOAM Linear Algebra Libraries

(A story of prototype implementations and future intentions)

Massimiliano Culpo[1,2]

[1]CINECA - SuperComputing Applications and Innovation Department

[2]PRACE1IP - Partnership for Advanced Computing in Europe

Motivation to the work

PRACE 1IP - Task 7.2

. . . will provide petascaling expertise to ensure that key application codes can effectively exploit Tier-0 systems. Will identify opportunities to enable applications through engagement with selected scientific communities, industrial users and specific application projects . . .

PRACE 1IP - Task 7.5

. . . will work with users to implement new programming techniques, paradigms and algorithms for Tier-0 systems, which have the potential to facilitate significant improvements in the performance of their applications. This task will work in close collaboration with tasks 7.1 and 7.2 . . .

icoFoam: Cavity Flow Benchmark

	OpenFOAM 1.7.1 (GNU)		OpenFOAM 1.7.1 (Intel)	
Task #	Time	% MPI	Time	% MPI
1	19259	-	18565	-
12	4005	2.56%	4020	2.43%
24	2608	4.10%	2616	4.36%
48	1326	6.88%	1311	6.05%
96	482	10.24%	469	9.19%
192	218	20.68%	210	18.71%
384	123	37.17%	107	31.20%
768	92	68.67%	109	71.09%

Table: Scaling results on the $200 \times 200 \times 200$ cells cavity flow test case

icoFoam: Cavity Flow Benchmark

Nothing but facts!

- On-core performance
  - around 500 MFlops
- Lot of time spent in
  - `MPI_Allreduce`
  - 8 bytes buffer size
- Fit very well to Tier-1
- . . . but not to Tier-0

$\Downarrow$

- Tuning needed
  - scaling behavior
  - exploit core architecture

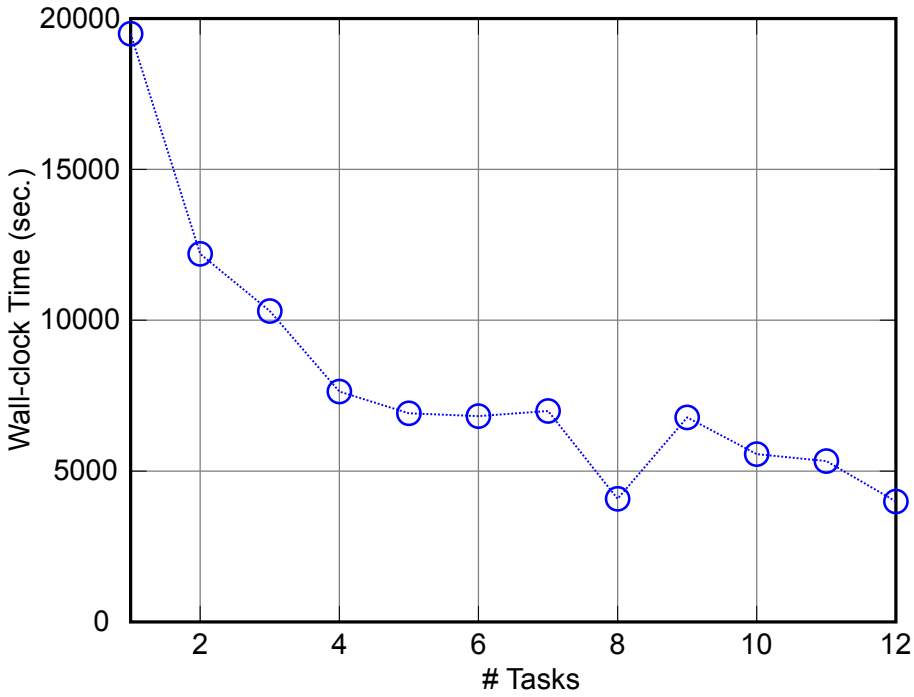
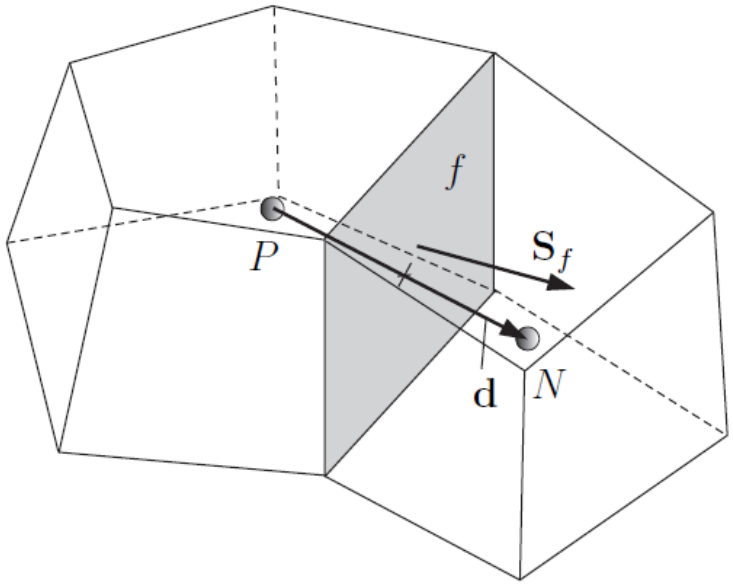


Figure: Intra-node scaling behavior of the $200 \times 200 \times 200$ cells cavity flow test case

A Glimpse on OpenFOAM Structure



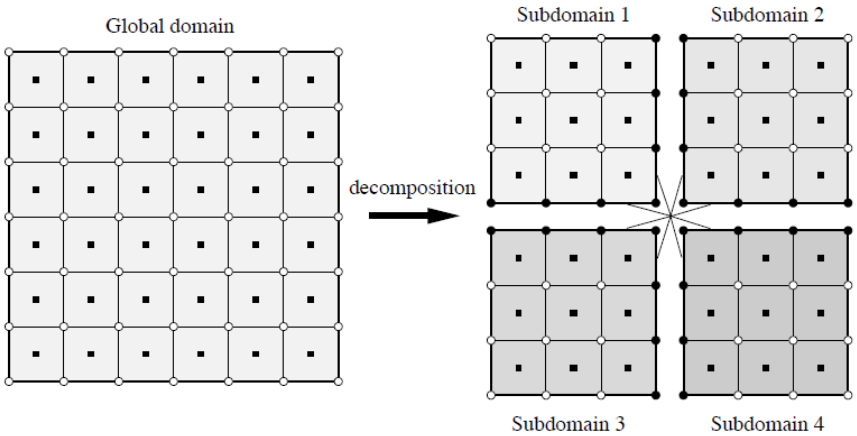
Finite Volume Primer

$$\int_{Cell} \mathrm{div}(A\nabla p)\ dV := \sum_{Faces} \int_{f} (A\nabla p) \cdot \mathbf{S}_f\ dS$$

Data Representation

- Variables may be associated with:
  1. cells
  2. faces
  3. points
- Time discretized independently

A Glimpse on OpenFOAM Structure



Solution Process: Overview

1. Different solvers, same kernel
   - sparse linear systems
   - iterative Krylov methods
   - SpMV multiplication

Task Decomposition

- Zero-Halo layer approach
- Internal Edges
  1. treated as BCs
- Usual pattern
  1. Interface initialization
  2. Local computation
  3. Interface update
- Local: Diagonal Block
- Interface: Off-Diagonal Blocks

Preconditioned Conjugate Gradient

Base Algorithm

1: $\mathbf{r}_0 \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}_0$
2: $\mathbf{z}_0 \leftarrow \mathbf{M}^{-1}\mathbf{r}_0$
3: $\mathbf{p}_0 \leftarrow \mathbf{z}_0$
4: $k \leftarrow 0$
5: **repeat**
6: $\alpha_k \leftarrow \frac{\mathbf{r}_k^{\mathrm{T}}\mathbf{z}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$
7: $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
8: $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$
9: $\mathbf{z}_{k+1} \leftarrow \mathbf{M}^{-1}\mathbf{r}_{k+1}$
10: $\beta_k \leftarrow \frac{\mathbf{z}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{z}_k^{\mathrm{T}}\mathbf{r}_k}$
11: $\mathbf{p}_{k+1} \leftarrow \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$
12: $k \leftarrow k + 1$
13: **until** $(\|\mathbf{r}_{k+1}\| < tol)$

Key Operations at Each Step

1. Sparse Matrix-Vector multiplication
1. Preconditioning
3. Scalar products

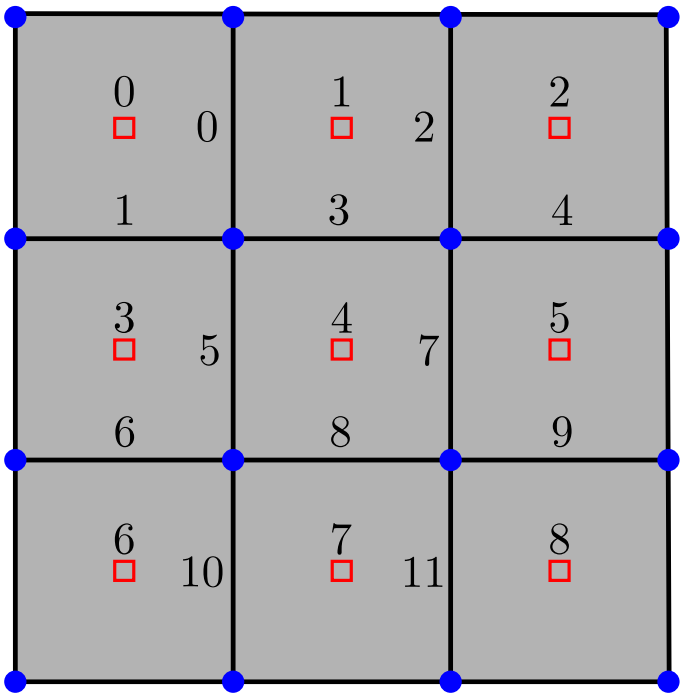
Zero-Halo Layer Scalar Product

```
scalar SumProd     = 0;
scalar partialSum  = 0;
// Local part of the product
for(label ii=0; ii<max; ii++)
   partialSum += f1p[ii] * f2p[ii];
// Gather other tasks contribution
MPI_Allreduce(&SumProd,&partialSum,1
MPI_SCALAR,MPI_SUM,MPI_COMM_WORLD);
```

Lessons learned so far...

1. The scalar products in the PCG algorithm act as barriers
2. The whole bunch of MPI_Allreduce
   - stems from an algorithmic constraint
   - is unavoidable, unless we venture on an algorithmic rewrite
3. How can we reduce communication and preserve the algorithm?
   - add multi-threading capabilities to sparse linear-algebra
4. To do that we have to dig into sparse matrix representation!

LDU Sparse Matrix Format



Storage Format Quick Guide

1. Matrix is considered
   - square
   - structurally symmetric
   - sum of three parts ($A = L + D + U$)
2. Off-diagonal positions mapping
   - `lPtr` (globally ordered)
   - `uPtr` (locally ordered)
3. Values stored in three double vectors
4. No fill-in introduced

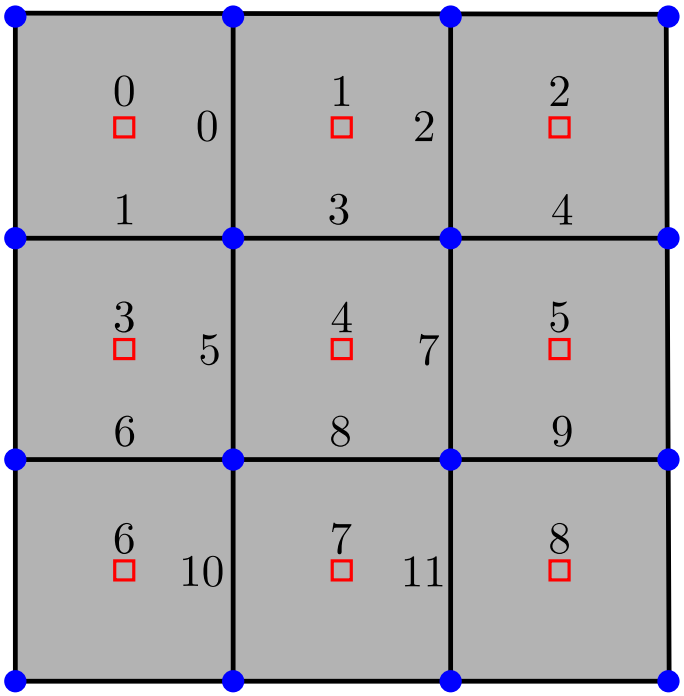
$$\texttt{lPtr} = \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 3 & 3 & 4 & 4 & 5 & 6 & 7 \end{bmatrix}$$

$$\texttt{uPtr} = \begin{bmatrix} 1 & 3 & 2 & 4 & 5 & 4 & 6 & 5 & 7 & 8 & 7 & 8 \end{bmatrix}$$

Sparse Matrix-Vector Multiplication

```
1  // Diagonal contributions
2  for (register label cell=0; cell<nCells; cell++)
3  {
4  ApsiPtr[cell] = diagPtr[cell]*psiPtr[cell];
5  }
6  // Off-diagonal contributions
7  for (register label face=0; face<nFaces; face++)
8  {
9  ApsiPtr[uPtr[face]] += lowerPtr[face]*psiPtr[lPtr[face]];
10 ApsiPtr[lPtr[face]] += upperPtr[face]*psiPtr[uPtr[face]];
11 }
```

Modifications to LDU Format



What prevents multi-threading?

1. Off-diagonal contributions
   - Concurrent write-access
   - Access by cells needed
2. Owner sort
   - `owPtr`
3. Losort
   - `reshape`
   - `loPtr`
4. Introduce doubly indirect access of rvalues

$$\texttt{owPtr} = \begin{bmatrix} 0 & 2 & 4 & 5 & 7 & 9 & 10 & 11 & 12 & 12 \end{bmatrix}$$

$$\texttt{loPtr} = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 & 5 & 7 & 8 & 10 & 12 \end{bmatrix}$$

$$\texttt{reshape} = \begin{bmatrix} 0 & 2 & 1 & 3 & 5 & 4 & 7 & 6 & 8 & 10 & 9 & 11 \end{bmatrix}$$

Sparse Matrix-Vector Multiplication

```
1  // Off-diagonal contributions
2  #pragma omp for
3  for (label cell=0; cell < nCells; ++cell)
4  {
5  for (label fidx = owPtr[cell]; fidx < owPtr[cell+1]; ++fidx)
6  {
7  AxPtr[cell] += upperPtr[fidx]*xPtr[uPtr[fidx]];
8  }
9  for (label fidx = loPtr[cell]; fidx < loPtr[cell+1]; ++fidx)
10 {
11 AxPtr[cell] += lowerPtr[reshape[fidx]]*xPtr[lPtr[reshape[fidx]]];
12 }
13 }
```

Can we go further?

1. Cache-Blocking Techniques
   - improvement of on-core performance
   - . . . and thus loss on scalability!
2. Change of Matrix Format to
   - hide latency to memory (S-CSR)
   - reduce size of data to be transfered (δ-CSR)
3. Ideas to be Developed in PRACE 2IP

THANKS, AND HAVE FUN IN THE EVENING!