Optimization of a Finite-Volume Method Application

José Alves, Rui Brito

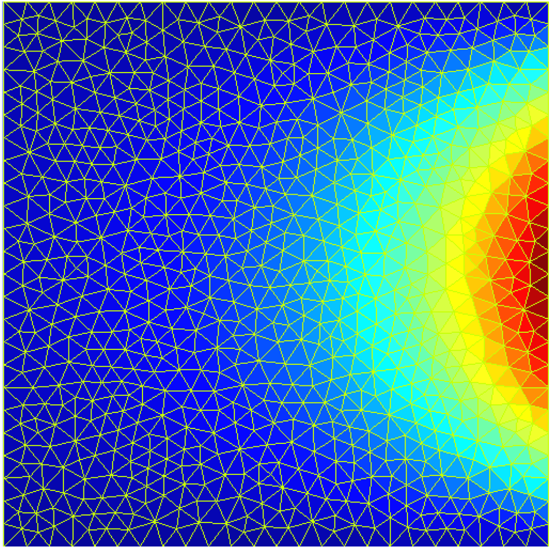
Universidade do Minho

Braga, July 2, 2013

Index

conv-diff (Recap)

- What? Computes the heat diffusion of a fluid spreading over an area;
- How? Uses a Finite-Volume method;
- Why? Represents surface as a mesh, making each cell only dependent of its neighbours;



The main objective is to compute a vector $\overline{\phi}$ such that

$\overline{\phi} \longrightarrow G(\overline{\phi}) = \begin{pmatrix} 0 \\ 0 \\ \vdots \end{pmatrix}$ This is accomplished in three different stages:

1. We begin with a candidate vector ϕ
2. For each edge, we compute the flux F_{ij}, with i and j being the indexes of the adjacent cells
3. For each cell, we compute $\sum |e_{ij}| F_{ij} - |c_i| f_i$

$$\text{Thus: } \phi = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_I \end{pmatrix} \longrightarrow G = \begin{pmatrix} G_1 \\ \vdots \\ G_I \end{pmatrix}$$

makeFlux Compute the contribution from each edge;

makeResidual Compute the ϕ vector, adding the flux for each cell from each contribution;

Original implementation

- ***Arrays-of-Pointers***;

makeFlux

For all **edges**:

1. Read adjacent cell data;
2. Compute edge velocity;
3. Compute flux through edge;

makeResidual

For all **edges**:

1. Subtract flux from right cell;
2. Add flux to left cell;

Test Machines (for most of the project)

	compute-511@search AMD Opt 6174	compute-601@search Xeon X5650	compute-101@search Xeon E7520	MacBookPro Intel Ivy-Bridge i7
# processors	2	2	2	1
# cores per processor	12	6	1	4
hyper-threading	-	yes	yes	yes
clock frequency(GHz)	2.2	2.66	3.2	2.3
L1 capacity	128KB	32KB	16KB	64KB
L2 capacity	512KB	256KB	2MB	256KB
L3 capacity	12MB	12MB	-	6MB
RAM capacity	64GB	48GB	2GB	16GB

Table: Test machines

Naive Optimizations

- Removed redundant loads and calculations;
- Changed some variable definitions to *const*;
- Usage of a recent compiler auto-optimizations(SLP);

Identified Problems

- High number of memory accesses;
- Low operational intensity;
- Deep memory indirection chain;
- Bad data locality;

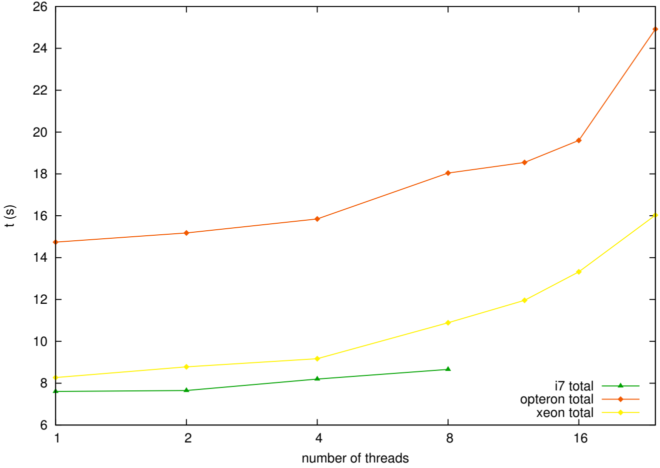


Figure: Total application runtime

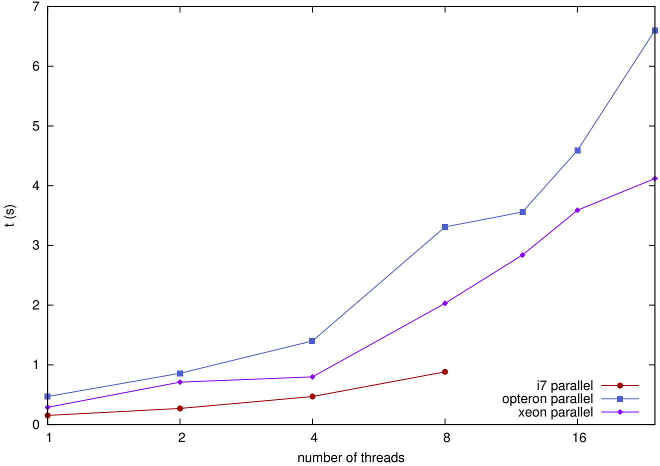
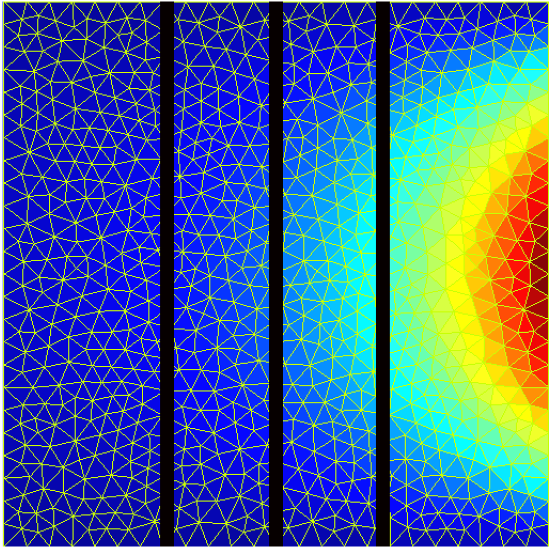


Figure: Parallel section runtime

Problems

- High level of communication between processes;
- High level of barrier synchronization;
- Some balancing problems;
- Computed error spikes;
- Some of FVLib's templates are hard to serialize (locality);
- Sequential portion is slow;



Optimizations

Array-Of-Structs

S_1	e_1
	e_2
	e_3
	...
S_2	e_1
	e_2
	e_3
	...
S_3	e_1
	e_2
	e_3
	...
...	

- Pointers $\Rightarrow$ Indexes;

Structs-Of-Arrays

e_1	e_2	e_3	...
S_1	S_1	S_1	
S_2	S_2	S_2	
S_3	S_3	S_3	
...	...	...	

- Pointers $\Rightarrow$ Indexes;
- Loads only what is needed;

OpenMP

Implementation

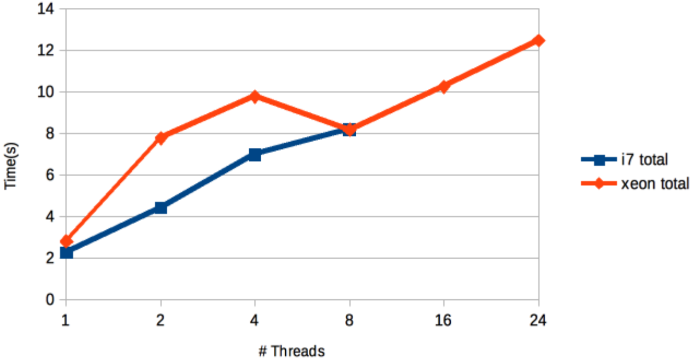
- SOA;
- Similar to sequential version:
  - `parallel for` added to both core functions;

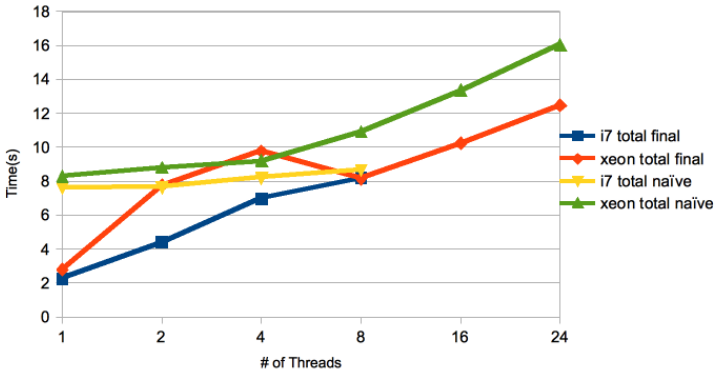
Load Balance

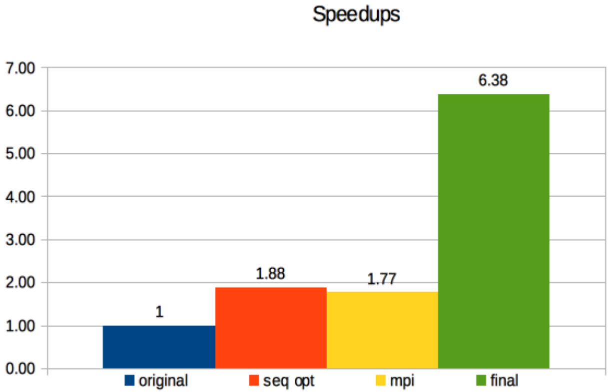
- Core functions are homogeneous;
- Static scheduling:
  - Round-robin;

Limitations

- Locality issues:
  - Softened with SOA;
  - Depends on the mesh structure;
    - `gmsh` does not optimize the mesh;
  - Specialized libraries:
    - `parmetis`;
    - Unknown complexity;


14
12
10
8
6
4
2
0
Time(s)
1
2
4
8
16
24
Threads
i7 total
xeon total


18
16
14
12
10
8
6
4
2
0
Time(s)
1
2
4
8
16
24
# of Threads
i7 total final
xeon total final
i7 total naïve
xeon total naïve


Speedups
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
1
1.88
1.77
6.38
original
seq opt
mpi
final

Conclusions

- SOA proved to be the best approach;
- Parallelization did not achieve any speedups;
- Bad locality of the mesh is a limitation;
    - Execution times spike with HyperThreading;
    - Does not scale well;

Optimization of a Finite-Volume Method Application

José Alves, Rui Brito

Universidade do Minho

Braga, July 2, 2013

– ? –