The effect of parallelization on a tetrahedral mesh optimization method

D. Benitez, E. Rodríguez, J.M. Escobar, R. Montenegro

SIANI Research Institute

University of Las Palmas de Gran Canaria, SPAIN

PPAM 2013

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

SIANI

Motivation

- Untangling and smoothing of tetrahedral meshes is an important step in the optimization of meshes in problems with moving boundaries because it provides simulations with elements of good quality
- Improving the speed of mesh generation helps users iterate problem setup faster
- There are currently no parallel mesh untangling algorithms in existence
- There are also no parallel simultaneous mesh untangling and smoothing techniques in the literature

Outline

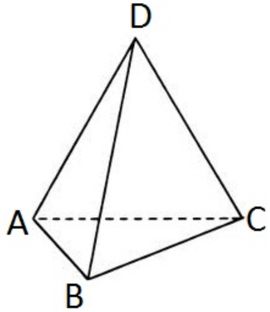
- **We propose a new parallel algorithm for simultaneous untangling and smoothing of tetrahedral meshes** (it is a tetrahedral mesh optimization method)
- **We also provide a detailed analysis of its parallelization on a many-core computer:**
  - **parallel scalability**
  - **load balancing**
  - **parallelism bottlenecks**
  - **influence of 3 graph coloring algorithms**

Summary

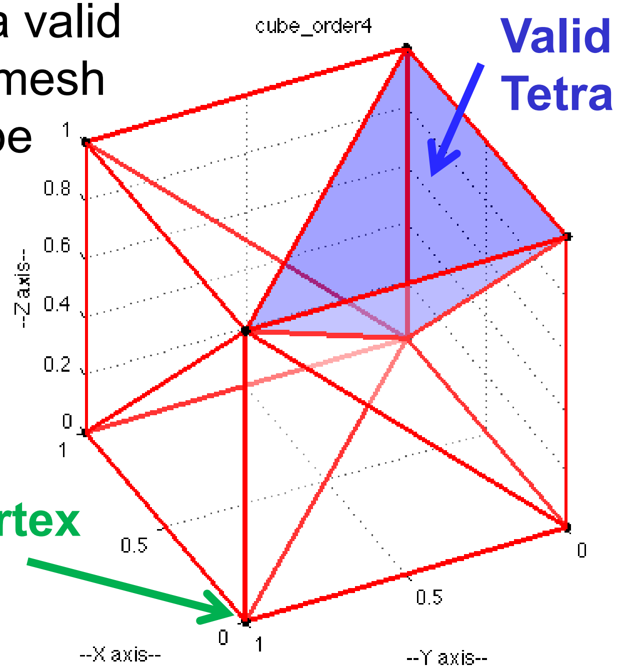
- Our mathematical approach to tetrahedral mesh optimization
- The novel parallel algorithm
- Experimental methodology
- Performance scalability
- Load balancing
- Parallelism bottlenecks
- Influence of coloring algorithms on parallel performance
- Conclusions and future work

Our mathematical approach to tetrahedral mesh optimization

Equilateral tetrahedron



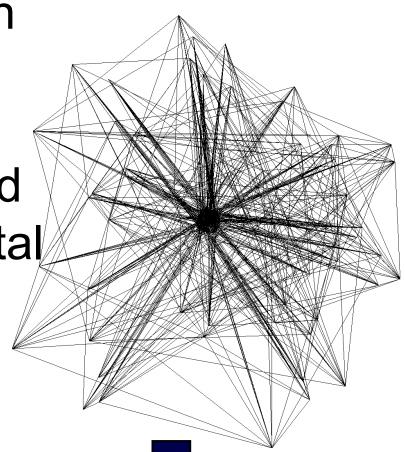
Example of a valid tetrahedral mesh for a cube



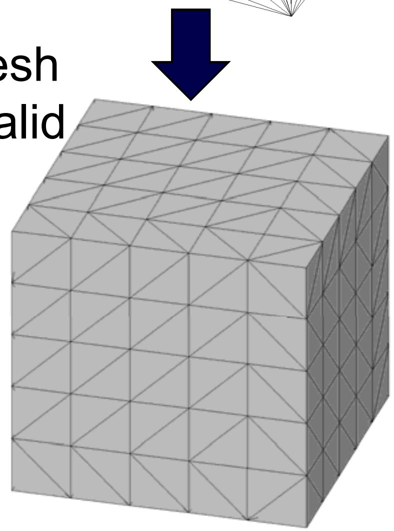
Valid Tetrahedron

A mesh vertex (node)

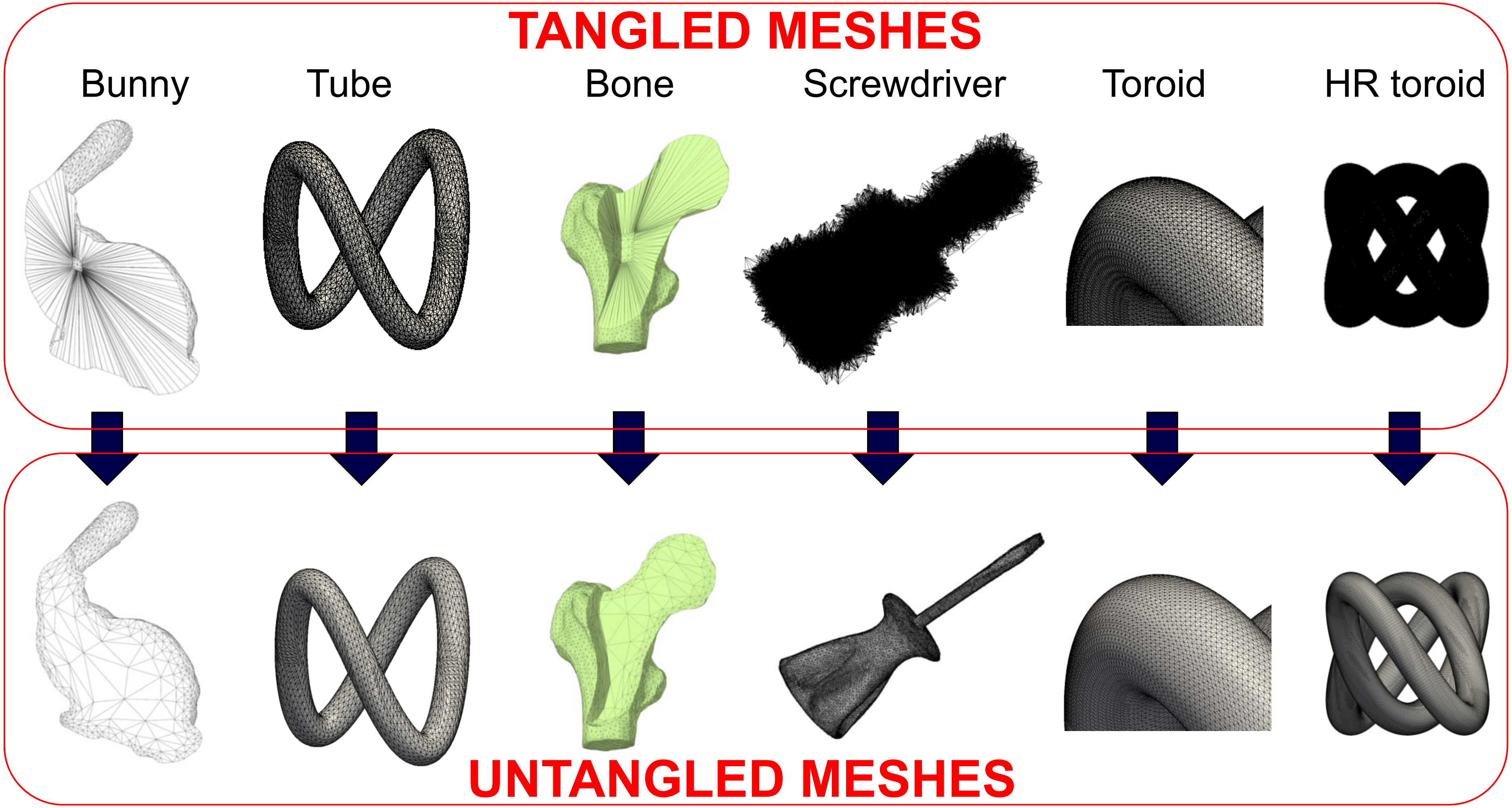
Tetrahedral mesh with non-valid tetrahedra (artificially tangled for our experimental tests)



Optimized tetrahedral mesh without non-valid tetrahedra

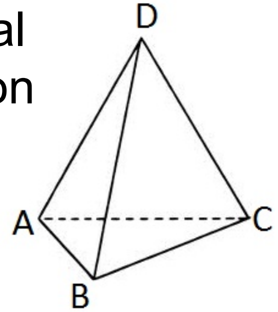


Our mathematical approach to tetrahedral mesh optimization



Our mathematical approach to tetrahedral mesh optimization

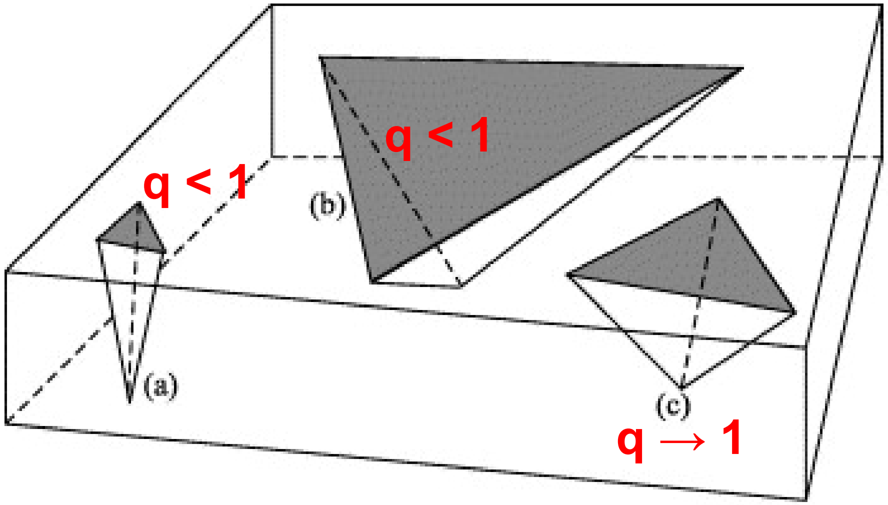
Equilateral tetrahedron



Its quality (q) specifies the degree to which regularity is achieved:

q=1: equilateral tetrahedron

q<1: non-equilateral tetrahedron



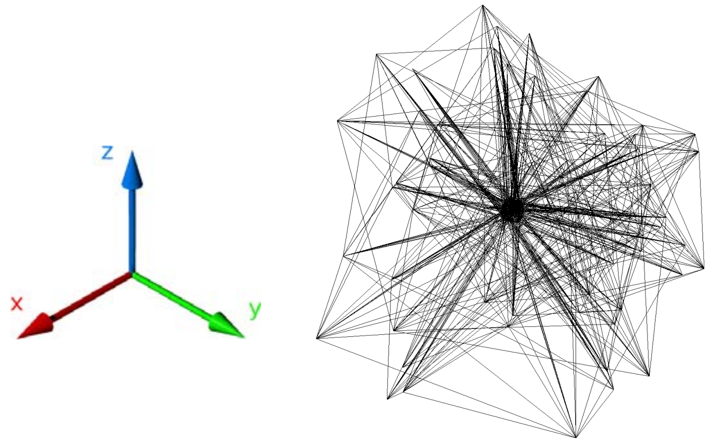
(a) A small triangle on the surface makes a tetrahedron too thin

(b) A large triangle on the surface makes a tetrahedron too flat

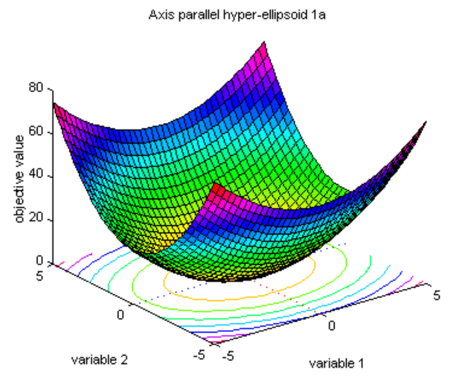
(c) An appropriately sized surface triangle makes a tetrahedron of desired shape

Our mathematical approach to tetrahedral mesh optimization

- **M : a tetrahedral mesh**
- $\boldsymbol{v}$ **: inner mesh node**
- $x_{\boldsymbol{v}}$ **: node position**
- $\mathbb{N}_{\boldsymbol{v}}$ **: the local submesh (set of tetrahedra connected to the node** $\boldsymbol{v}$**)**



- $K(x_{\boldsymbol{v}})$ **: objetive function that measures the quality of the local submesh**



Our mathematical approach to tetrahedral mesh optimization

- **Our untangling and smoothing technique finds the new position $x_{\boldsymbol{v}}$ that each inner mesh node $\boldsymbol{v}$ must hold, in such a way that $K(x_{\boldsymbol{v}})$ is optimized**
- **This process repeats several times for all the nodes of the mesh M**
- **Mathematical details:** J. M. Escobar , E. Rodrıguez , R. Montenegro , G. Montero , J. M. Gonzalez-Yuste (2003) Simultaneous untangling and smoothing of tetrahedral meshes. Computer Methods in Applied Mechanics and Engineering, 192: 2775-2787

The novel parallel algorithm

- **<u>Sequential</u> algorithm (SUS) for the simultaneous untangling and smoothing of a tetrahedral mesh M**

function OptimizeNode($x_{\boldsymbol{v}}$, $\mathbb{N}_{\boldsymbol{v}}$)
  Optimize objective function K($x_{\boldsymbol{v}}$)
end function
procedure SUS
  Q $\leftarrow$ 0
  k $\leftarrow$ 0
  while Q < λ **and** k < maxIter **do**
    for each vertex $\boldsymbol{v} \in M$ **do**
      $x'_{\boldsymbol{v}} \leftarrow$ OptimizeNode($x_{\boldsymbol{v}}$,N_v)
    end do
    Q $\leftarrow$ quality(M)
    k $\leftarrow$ k+1
  end do
end procedure

The novel parallel algorithm

- **<u>Sequential</u> algorithm (SUS) for the simultaneous untangling and smoothing of a tetrahedral mesh M**

```
function OptimizeNode(x_v, N_v)
  Optimize objective function K(x_v)
end function
procedure SUS
  Q ← 0
  k ← 0
  while Q < λ and k < maxIter do
    for each vertex v ∈ M do
      x'_v ← OptimizeNode(x_v, N_v)
    end do
    Q ← quality(M)
    k ← k+1
  end do
end procedure
```

INPUTS:

- *M* : tangled tetrahedral mesh,
- *maxIter* : maximum number of untangling and smoothing iterations
- N_v : set of tetrahedra connected to the free *node* v
- x_v :is the initial position of the free node
- x'_v : its position after optimization, which is implemented with procedure *OptimizeNode()*
- *Q* measures the lowest quality of a tetrahedron of *M*
- *quality()* : function that provides the minimum quality of mesh *M*

The novel parallel algorithm

- <u>Sequential</u> algorithm (SUS) for the simultaneous untangling and smoothing of a tetrahedral mesh M

function OptimizeNode($x_{\boldsymbol{v}}$, $\mathbb{N}_{\boldsymbol{v}}$)
    Optimize objective function K($x_{\boldsymbol{v}}$)
end function
procedure SUS
    Q $\leftarrow$ 0
    k $\leftarrow$ 0
    while Q < λ **and** k < maxIter **do**
        for each vertex $\boldsymbol{v} \in M$ **do**
            $x'_{\boldsymbol{v}} \leftarrow$ OptimizeNode($x_{\boldsymbol{v}}$,N_v)
        end do
        Q $\leftarrow$ quality(*M*)
        k $\leftarrow$ k+1
    end do
end procedure

OUTPUT:

•an untangled and smoothed mesh *M*, whose minimum quality must be larger than an user-specified threshold λ

The novel parallel algorithm

- **<u>Sequential</u> algorithm (SUS) for the simultaneous untangling and smoothing of a tetrahedral mesh M**

function OptimizeNode($x_{\boldsymbol{v}}$, $\mathbb{N}_{\boldsymbol{v}}$)
    Optimize objective function K($x_{\boldsymbol{v}}$)
end function
procedure SUS
    Q $\leftarrow$ 0
    k $\leftarrow$ 0
    while Q < λ **and** k < maxIter **do**
        for each vertex $\boldsymbol{v} \in M$ **do**
            $x'_{\boldsymbol{v}} \leftarrow$ OptimizeNode($x_{\boldsymbol{v}}$,N_v)
        end do
        Q $\leftarrow$ quality(M)
        k $\leftarrow$ k+1
    end do
end procedure

This algorithm iterates sequentially over all the mesh vertices in some order, at each step adjusting the spatial coordinates x'_v ***of*** a free node $\boldsymbol{v}$

The novel parallel algorithm

- **<u>Sequential</u> algorithm (SUS) for the simultaneous untangling and smoothing of a tetrahedral mesh M**

```
function OptimizeNode(x_v, ℕ_v)
  Optimize objective function K(x_v)
end function
procedure SUS
  Q ← 0
  k ← 0
  while Q < λ and k < maxIter do
    for each vertex v ∈ M do
      x'_v ← OptimizeNode(x_v, N_v)
    end do
    Q ← quality(M)
    k ← k+1
  end do
end procedure
```

This process repeats several times for all the nodes of the mesh M

The novel parallel algorithm

- **<u>Parallel</u> algorithm (pSUS) for the simultaneous untangling and smoothing of a tetrahedral mesh *M***

function OptimizeNode(x_v, $\mathbb{N}_v$)
  Optimize objective function K(x_v)
end function
procedure Coloring(G=(V,E))
G is partitioned into independent sets I={I_1, I_2, …} using C_1, C_2 or C_3 coloring algorithm
end procedure

Its inputs *are the same* as described for sequential algorithm

procedure pSUS
  I $\leftarrow$ Coloring(G=(V,E))
  k $\leftarrow$ 0
  Q $\leftarrow$ 0
  while Q < λ **and** k < maxIter **do**
    for each independent set $I_i \in$ I **do**
      for each vertex $v \in I_i$ **in parallel do**
        $x'_v \leftarrow$ OptimizeNode(x_v,N_v)
      end do
    end do
    Q $\leftarrow$ quality(*M*)
    k $\leftarrow$ k+1
  end do
end procedure

The novel parallel algorithm

- **Parallel** algorithm (**pSUS**) for the simultaneous untangling and smoothing of a tetrahedral mesh *M*

```
function OptimizeNode(x_v, N_v)
   Optimize objective function K(x_v)
end function
procedure Coloring(G=(V,E))
G is partitioned into independent
sets I={I_1, I_2, …} using C_1, C_2 or C_3
coloring algorithm
end procedure
```

We implemented graph coloring with procedure Coloring(), which partitions the mesh in a disjoint sequence of independent sets: I_1, I_2, …

```
procedure pSUS
   I ← Coloring(G=(V,E))
   k ← 0
   Q ← 0
   while Q < λ and k < maxIter do
     for each independent set I_i ∈ I do
       for each vertex v ∈ I_i in parallel do
         x'_v ← OptimizeNode(x_v,N_v)
       end do
     end do
     Q ← quality(M)
     k ← k+1
   end do
end procedure
```

The novel parallel algorithm

- **<u>Parallel</u> algorithm (pSUS) for the simultaneous untangling and smoothing of a tetrahedral mesh *M***

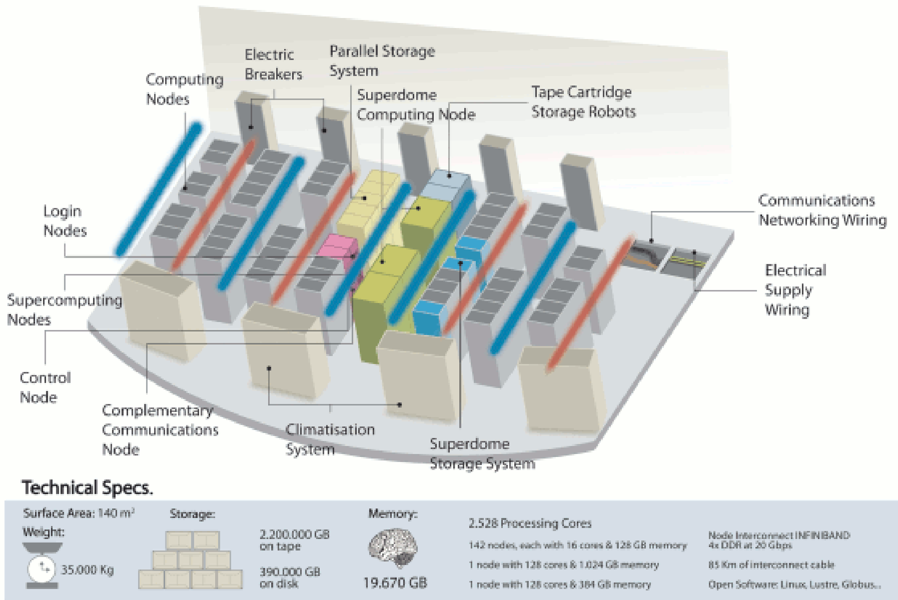
function OptimizeNode(x_v, $\mathbb{N}_v$)
  Optimize objective function K(x_v)
end function
procedure Coloring(G=(V,E))
G is partitioned into independent sets I={I_1, I_2, …} using C_1, C_2 or C_3 coloring algorithm
end procedure

This parallel algorithm optimize in parallel the nodes of each independent set

procedure pSUS
  I ← Coloring(G=(V,E))
  k ← 0
  Q ← 0
  while Q < λ **and** k < maxIter **do**
    for each independent set $I_i \in$ I **do**
      for each vertex $v \in I_i$ **in parallel do**
        x'_v ← OptimizeNode(x_v,N_v)
      end do
    end do
    Q ← quality(*M*)
    k ← k+1
  end do
end procedure

Experimental methodology

- **Finis Terrae supercomputer (www.cesga.es), the third largest supercomputer in Spain**

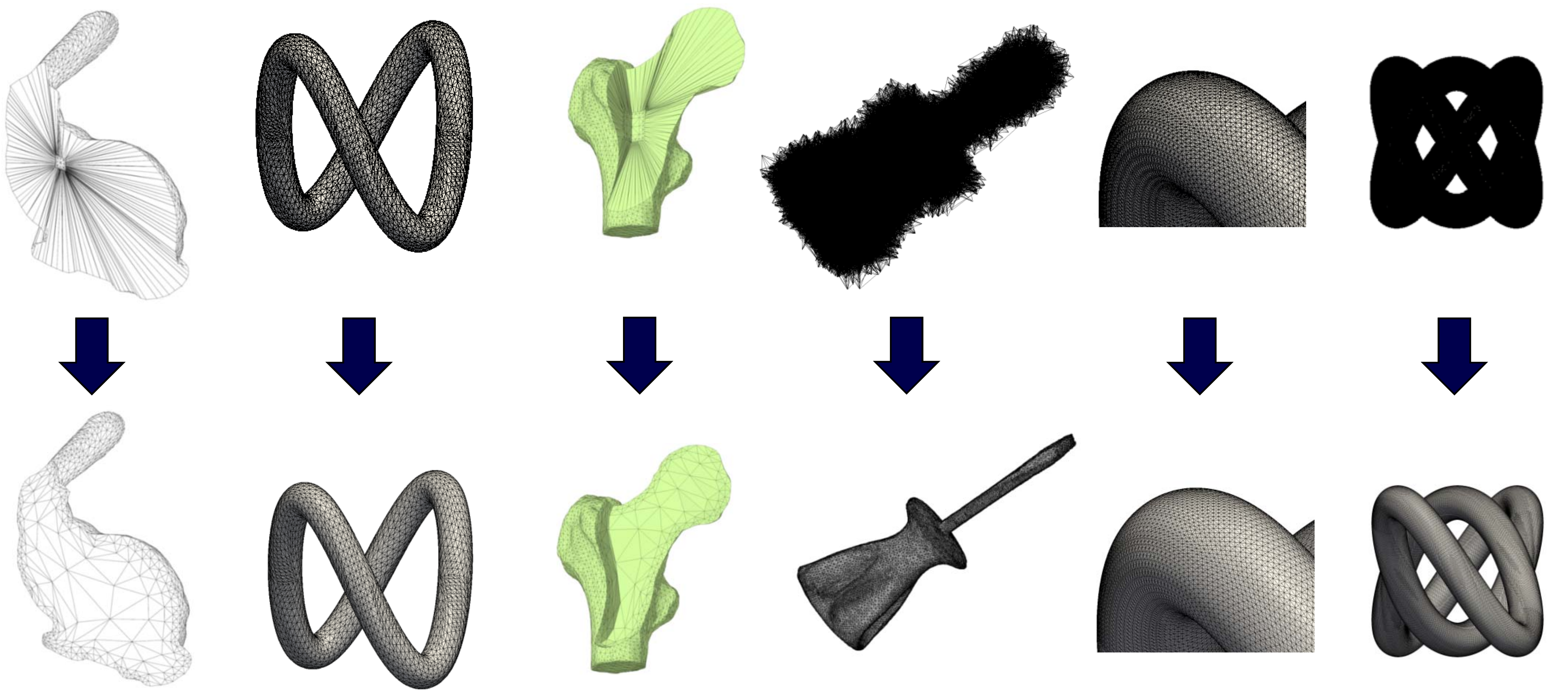


- **1 many-core node HP Integrity Superdome, with 128 cores Itanium Montvale and 1.024 GB (NUMA): shared memory architecture**

Experimental methodology

- **Six different tangled benchmark meshes**

“m=6358”	“m=9176”	“m=11525”	“m=39617”	“m=201530”	“m=520128”
Bunny	Tube	Bone	Screwdriver	Toroid	HR toroid



Experimental methodology

- **Description of the tangled benchmark meshes. The quality of non-valid tetrahedra is considered zero. So, the minimum quality is zero for all meshes**

Name	Number of vertices (m)	Number of tetrahedra	Average mesh quality	Number of inverted tetrahedra	Maximum vertex degree	Object
“m=6358”	6358	26446	0.2618	2215	26	Bunny
“m=9176”	9176	35920	0.1707	13706	26	Tube
“m=11525”	11525	47824	0.2660	1924	26	Bone
“m=39617”	39617	168834	0.1302	83417	26	Screwdriver
“m=201530”	201530	840800	0.2409	322255	26	Toroid
“m=520128”	520128	2201104	0.0657	1147390	26	HR toroid

Experimental methodology

- Intel C++ compiler 11.1 with “O2” flag
- Linux system kernel “2.6.16.53-0.8-smp”.
- The source code of the parallel version included OpenMP directives, which were disabled when the sequential version was compiled
- Both software versions were profiled with PAPI API, which uses performance counter hardware of Itanium 2 processors
- Hardware binding: processor and memory

Experimental methodology

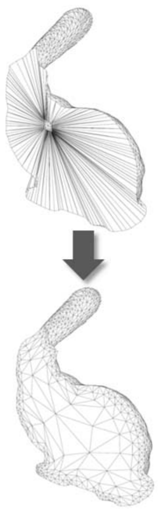
- For each benchmark mesh we run the parallel version multiple times using a given maximum number of active threads between 1 and 128
- Each run is divided into two phases
  - The first of them completely untangles a mesh. This phase loops over all mesh vertices repetitively
  - The second phase smoothes the mesh until successive iterations increases the minimum mesh quality less than 5%

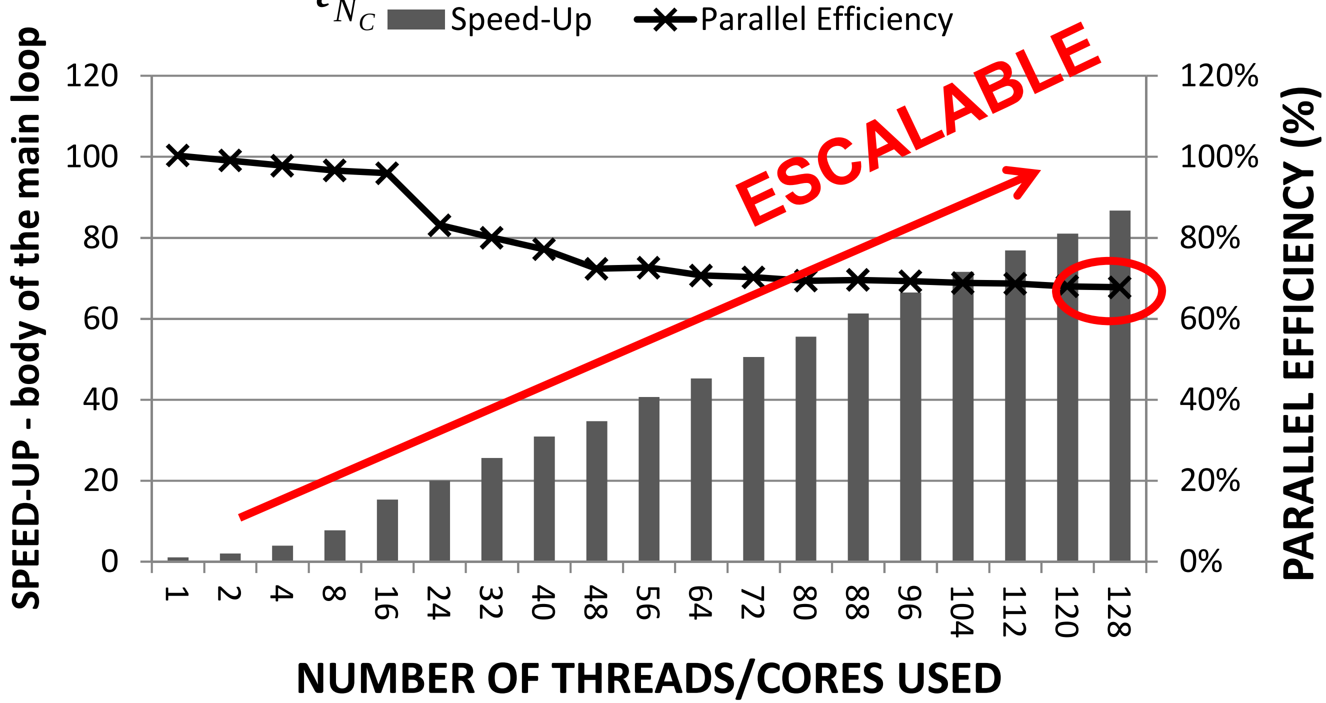
Performance scalability

- True speed-up and parallel efficiency of the body of the main loop

$$\underbrace{\text{the body of the main loop}}_{x'_v \leftarrow \text{OptimizeNode}(x_v, \text{N}_v)}$$

$$Speed-Up = \frac{t_S}{t_{N_C}} \qquad ParallelEfficiency = 100\% \times \frac{S_{N_C}}{N_C}$$

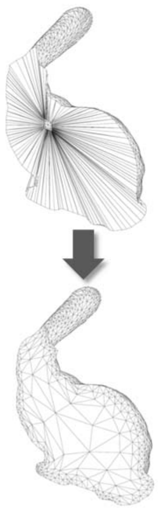




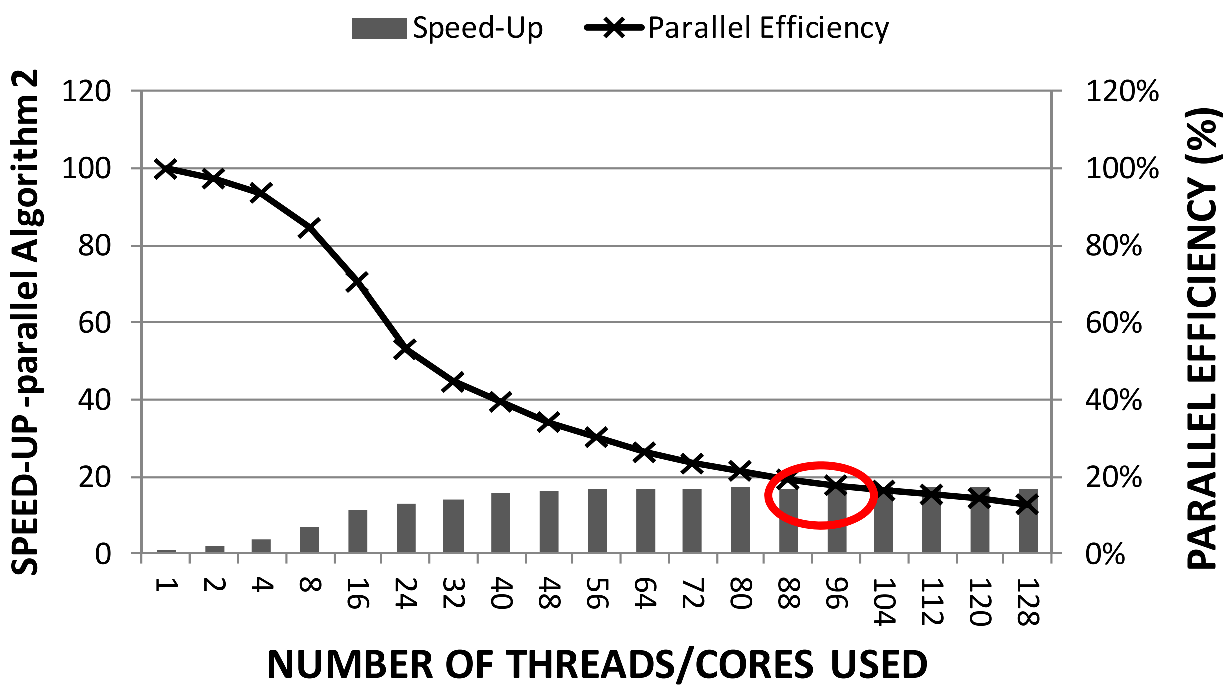
Performance scalability

- **True speed-up and parallel efficiency of the body of the complete parallel Algorithm**

procedure pSUS
…
while Q < λ **and** k < maxIter **do**
for each independent set $I_i \in I$ **do**
for each vertex $v \in I_i$ **in parallel do**
$x'_v \leftarrow \text{OptimizeNode}(x_v, N_v)$



m=6358



Performance scalability

- **Best runtime for the complete parallel algorithm (procedure pSUS)**

Name of tetrahedral mesh	Serial runtime (seconds)	Best parallel runtime (seconds)	Best number of cores	Best Speed-Up	Best parallel efficiency	Best coloring algorithm	Number of colors	Number of iterations (U&S)	Minimum mesh quality	Average mesh quality
m=6358	17.33	1.49	72	11.7X	16.2%	C_1	29	25	0.1319	0.6564
m=9176	37.25	1.17	88	31.9X	36.3%	C_3	29	26	0.2580	0.6823
m=11525	33.69	1.13	120	29.7X	24.8%	C_3	10	38	0.1109	0.6474
m=39617	87.40	1.59	128	54.9X	42.9%	C_1	31	11	0.1698	0.7329
m=201530	2505.37	81.28	128	30.8X	24.1%	C_2	21	143	0.2275	0.6687
m=520128	2259.72	41.86	120	54.0X	45.0%	C_3	34	36	0.2233	0.6750

Performance scalability

- **Performance model for our parallel algorithm based on Amdahl's law**

t_S	Sequential time
$t_{N_C}^{P}$	Parallel time without overhead
$t_{N_C} = t_{Nc}^{P} + t_{Nc}^{O}$	Parallel time with overhead

Speed-up

$$S_{N_C} = \frac{t_S}{t_{Nc}^{P} + t_{Nc}^{O}}$$

$$t = \frac{N_I}{IPC \times f}$$ CPU time

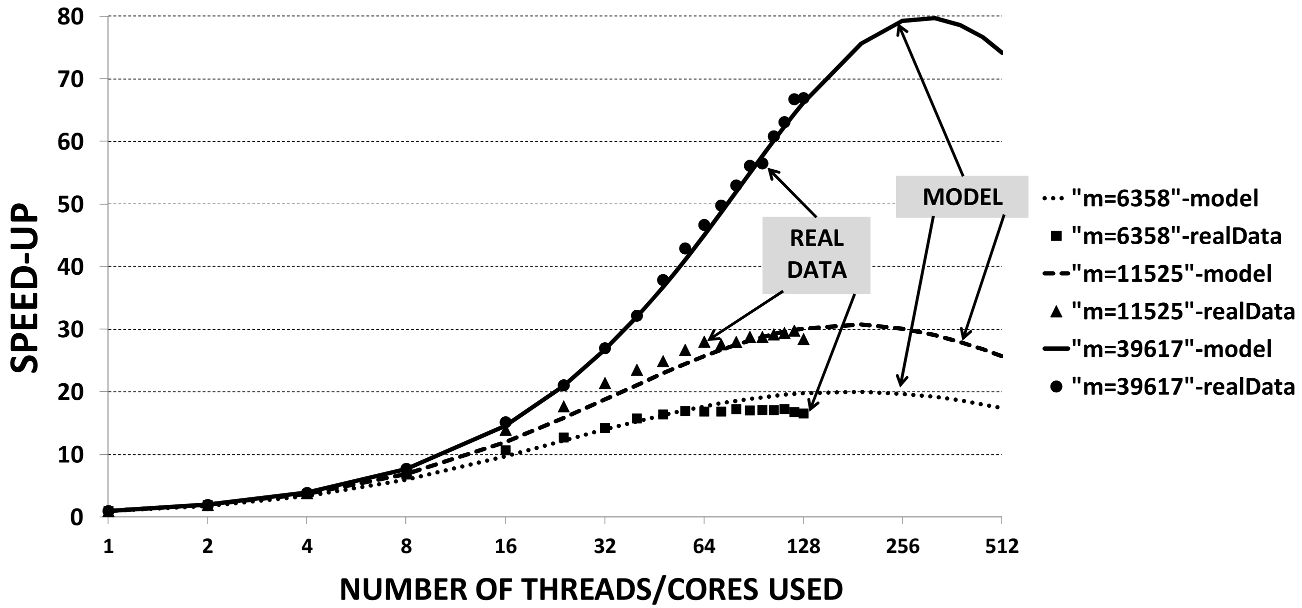
Performance scalability

- **Performance model for our parallel algorithm based on Amdahl's law**

$$S_{N_C} = \frac{t_S}{t_{Nc}^P + t_{Nc}^O} = \frac{\frac{t_S}{t_{Nc}^P}}{1 + \frac{t_{Nc}^O}{t_{Nc}^P}} = \frac{\frac{t_S}{t_{Nc}^P}}{1 + \theta_{N_C}}$$

$$t_{N_C}^P \approx \frac{t_S}{N_C}$$

$$\theta_{N_C} = \frac{t_{Nc}^O}{t_{Nc}^P} \underset{f=cons}{=} \frac{IPC_{Nc}^P \times N_I^O}{IPC_{Nc}^O \times N_I^P}$$

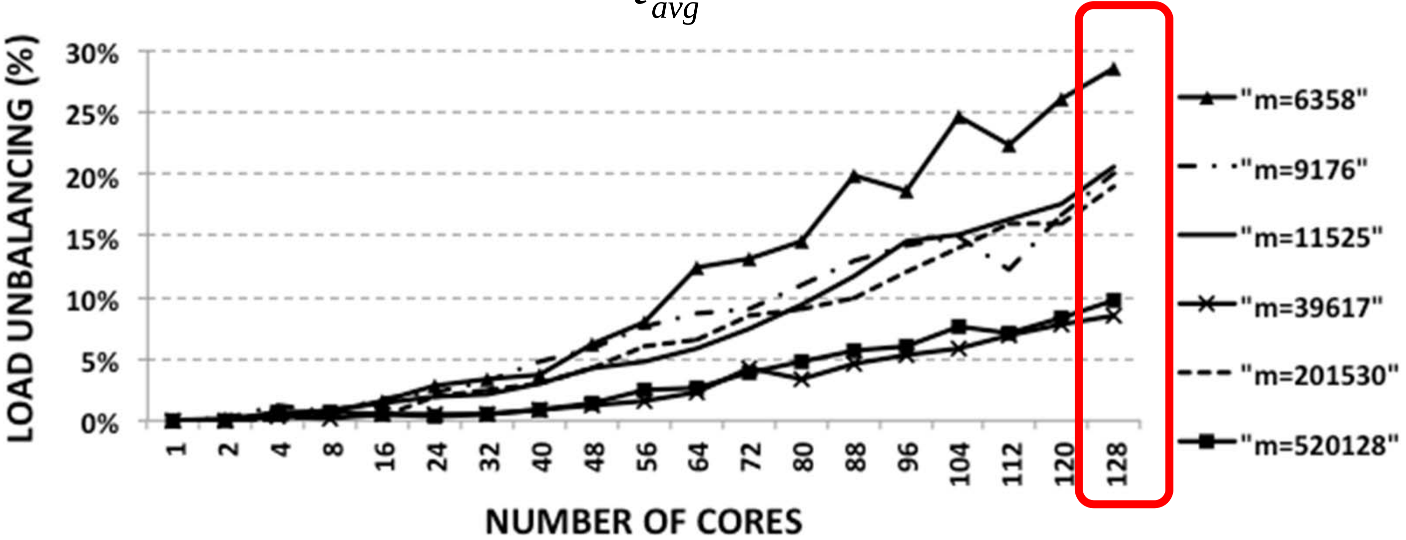


Conclusion: parallel efficiency deteriorate as the number of threads increases because they tend to be dominated by the thread scheduling overhead

Load balancing

- **Load unbalancing when meshes with different number of vertices (m) and up to 128 Itanium2 cores are used. C_3 coloring algorithm and dynamic OpenMP thread scheduling are used**

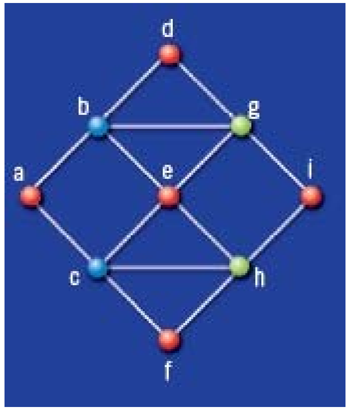
$$L_{N_C} = 100\,\% \times \frac{t_{\max} - t_{\min}}{t_{avg}} \qquad t_{max} > t_{avg} > t_{min} > 0$$



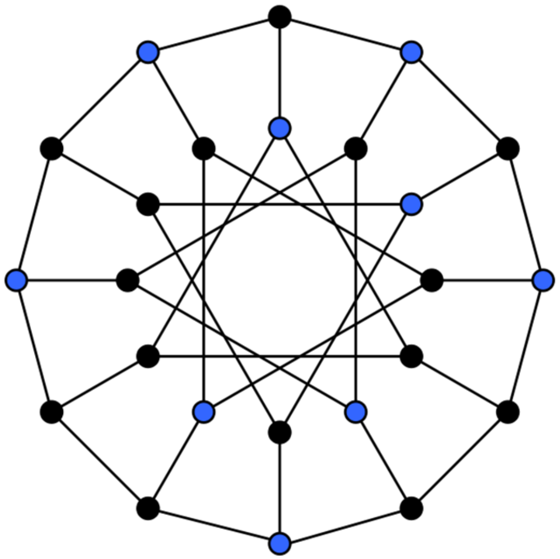
Parallelism bottlenecks

- During runtime of the main mesh optimization procedure, stall cycles of each parallel thread are in the range from 29%(1 core) to 58%(128 cores)
- These computation bottlenecks are located in:
  - double-precision floating-point units: from 70%(1c) to 27%(128c) of stall cycles
  - data loads: from 16%(1c) to 55%(128c)
    - cache memories: main source of data load stall cycles
    - NUMA (Non-Uniform Memory Access) memory: less than 1% of data load stall cycles
  - branch instructions: from 5%(1c) to 14%(128c)
  - “no-operation” instructions (40%): caused by the long instruction format and compiler inefficiency

Influence of coloring algorithms on parallel performance



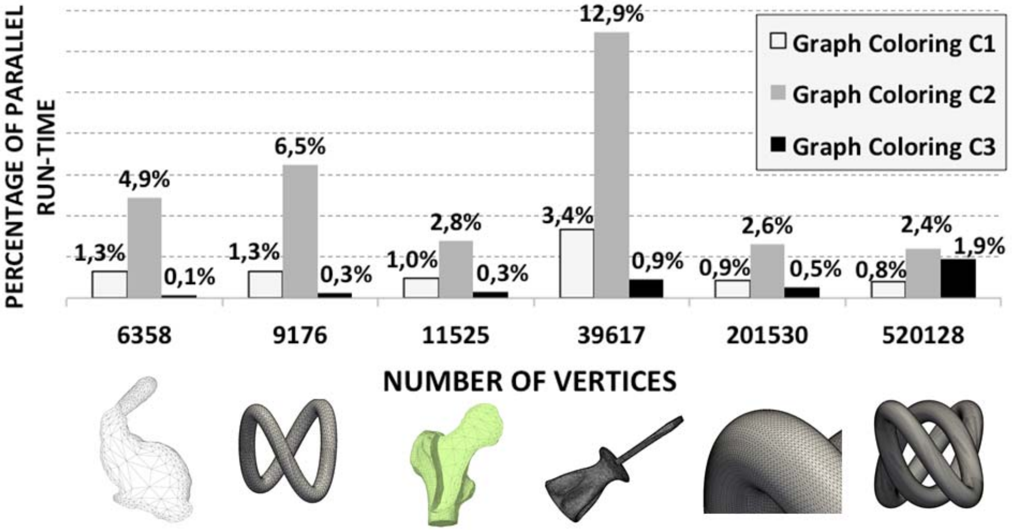
Distance-1 coloring : adjacent nodes do not have the same color



An independent set of a graph is a set of not adjacent vertices

Influence of coloring algorithms on parallel performance

- Percentage of total parallel runtime that is required by the three graph coloring algorithms C_1, C_2, and C_3 when the six benchmark meshes are untangled and smoothed and 128 Itanium2 processors are used

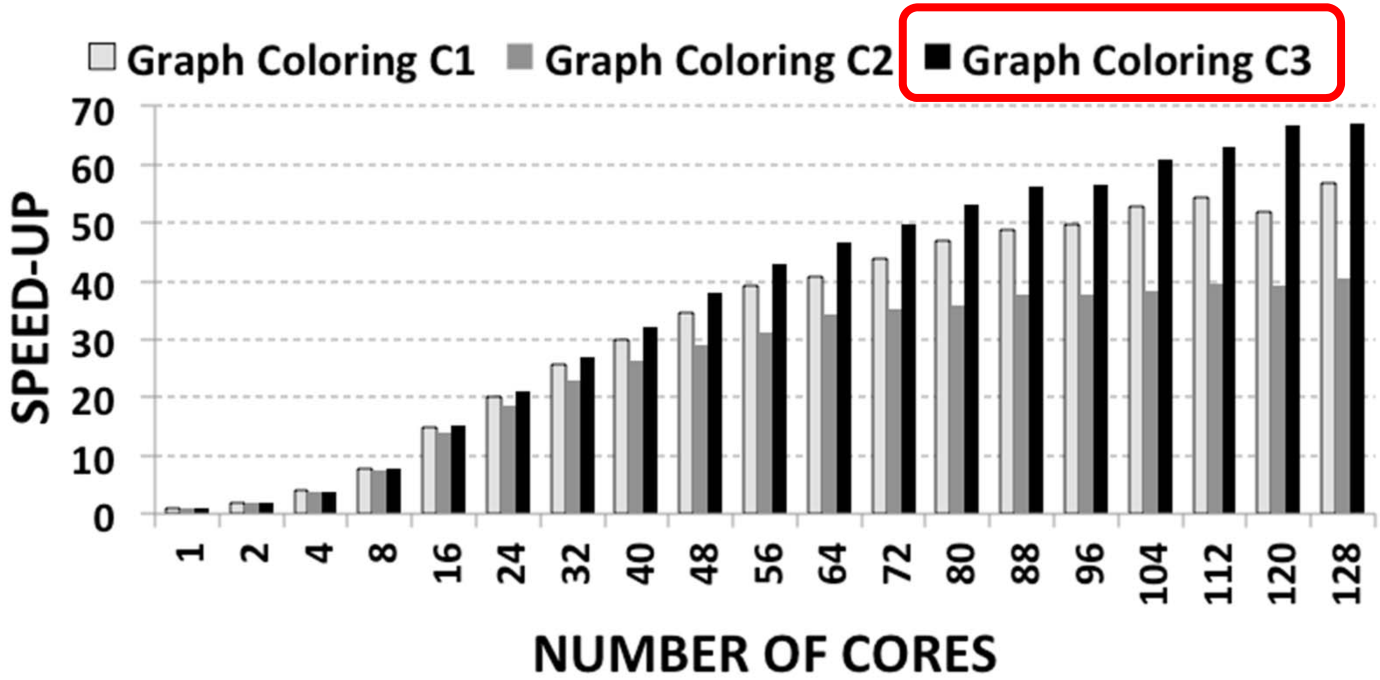


- This means that the computational load required by our parallel algorithm is much heavier than required by graph coloring algorithms.

Influence of coloring algorithms on parallel performance

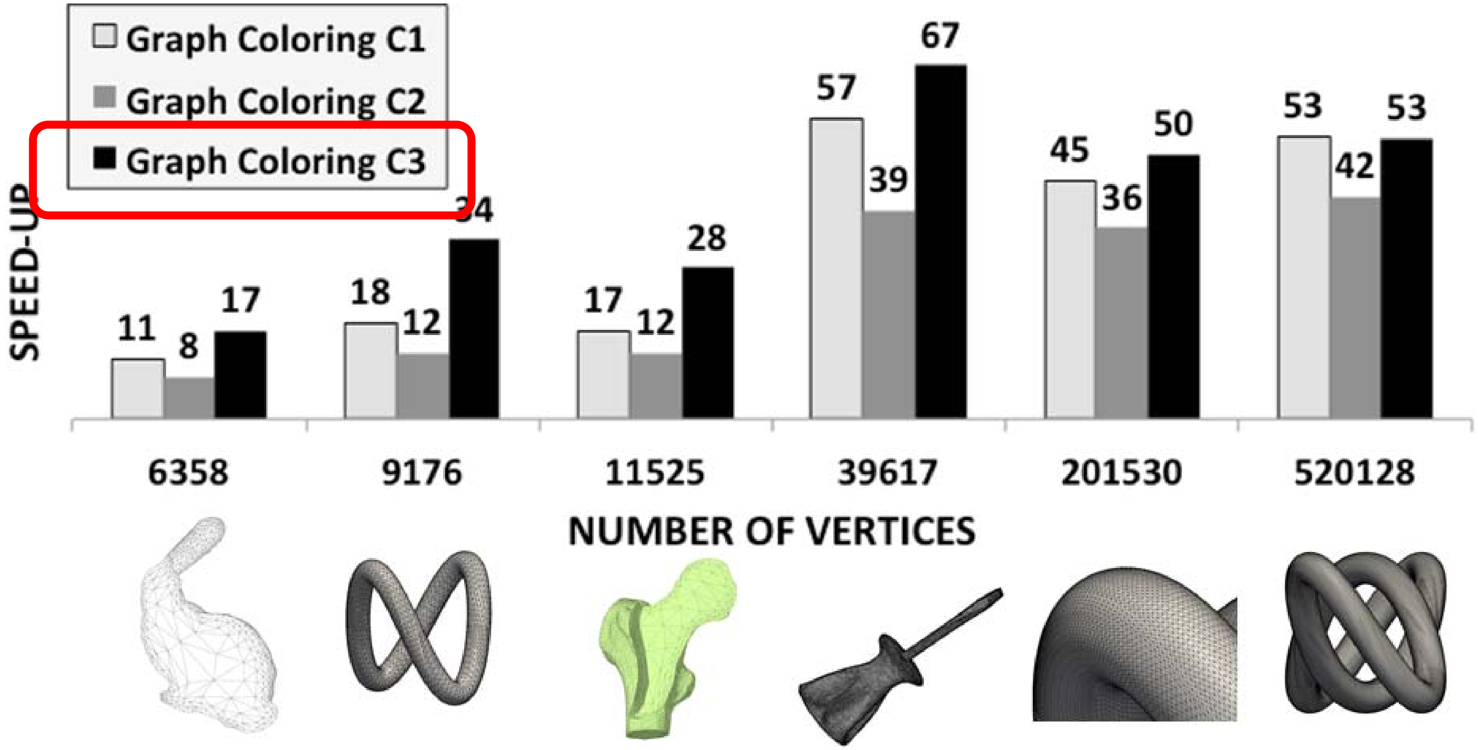
- **Speed-up achieved by the complete parallel algorithm (pSUS) depends on the mesh coloring algorithm**

m=39617



Influence of coloring algorithms on parallel performance

- **Speed-up achieved by the complete parallel algorithm (pSUS) for all six benchmark meshes when three graph coloring algorithms (C_1, C_2, and C_3) are used and all 128 shared-memory Itanium2 processors are active**



Conclusions

- **We demonstrate that this algorithm is highly scalable when run on a high-performance shared-memory many-core computer with up to 128 Itanium 2 processors.**
- **It is due to the graph coloring algorithm that is used to identify independent sets of vertices without computational dependency**

Conclusions

- We have analyzed the causes of its parallel deterioration on a 128-core shared-memory high performance computer using six benchmark meshes.
- It is mainly due to loop-scheduling overhead of the OpenMP programming methodology.
- The graph coloring algorithm has low impact on the total execution time. However, the total execution time of our parallel algorithm depends on the selected coloring algorithm.

Future work

- Our parallel algorithm is CPU bound and its demonstrated scalability potential for many-core architectures encourages us to extend our work to achieve higher performance improvements from massively parallel GPUs.
- The main problem will be to reduce the negative impact of global memory random accesses when the non-consecutive mesh vertices are optimized by the same streaming multiprocessor.