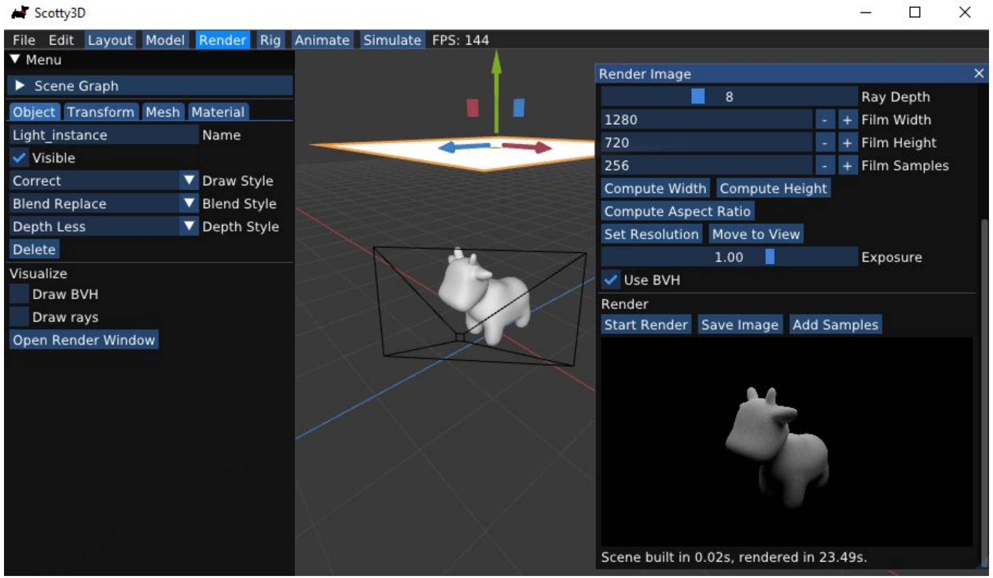
Scotty3D

- Custom graphics package used in 15462/662 Computer Graphics; C++
- Supports rasterization, raytracing, mesh editing, and particle simulation



FPS: 144

Light_instance Name

☑ Visible

Correct ▼ Draw Style

Blend Replace ▼ Blend Style

Depth Less ▼ Depth Style

8 Ray Depth

1280 - + Film Width

720 - + Film Height

256 - + Film Samples

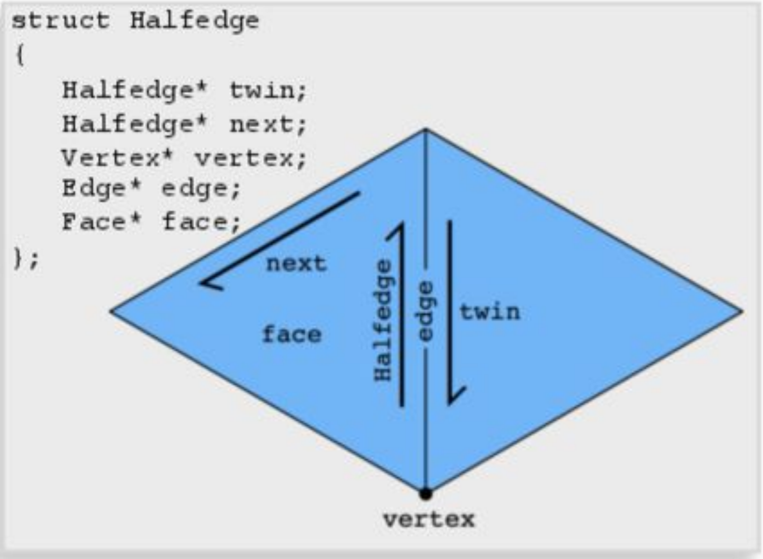
1.00 Exposure

☑ Use BVH

Scene built in 0.02s, rendered in 23.49s.

Halfedge Mesh Data Structure

- Local connectivity description with explicit lists of faces, edges, vertices, and ‘halfedges’

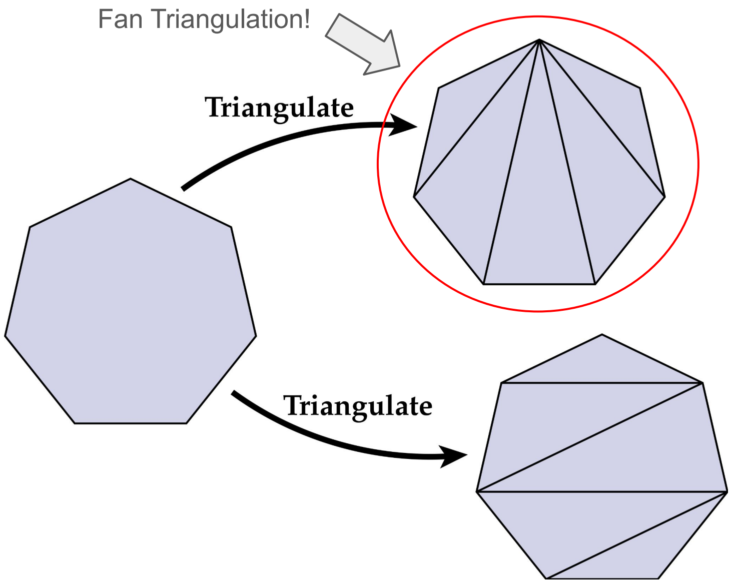


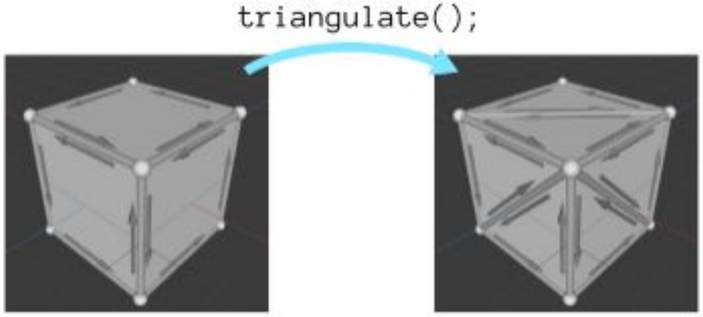
```
struct Halfedge
{
    Halfedge* twin;
    Halfedge* next;
    Vertex* vertex;
    Edge* edge;
    Face* face;
};
```

Polygon Triangulation

- Input : Any manifold* mesh
- Output : Same mesh with every face degree = 3

- Loop through every face input mesh has and subdivide it into triangles
- New face / edges / halfedges created in the process => Problem for parallelization!



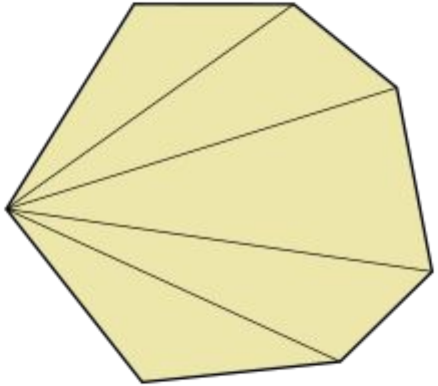


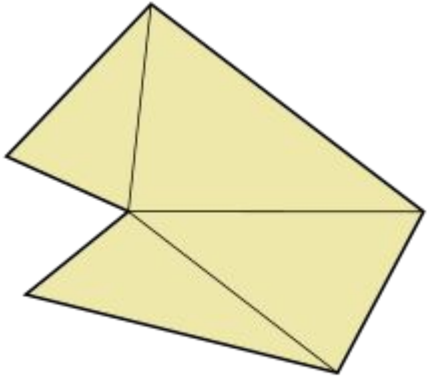
Manifold = “can exist in real world”

Serial Solution

- Loop through every face, perform fan triangulation
  - Fan triangulation : Choose a vertex and draw edges to all other vertices in the face
- Works regardless of concave / convex
  - Assume inverted faces are ok
- Let a face have *n* vertices, this generates *n-3* new faces, *n*-3 new edges, and *2*(n*-3*)* new halfedges

- **Problem** : we have temporal locality but no spatial locality => a lot of memory lookups & cache misses
  - Parallelize to hide latency of memory lookups!





Difficulties in Parallelization

1. Very hard to predict workload
    a. Unknown distribution of face degrees until we inspect each face

2. Cannot benefit from memory coherency
    a. No elements consisting a face is guaranteed to be contiguous in memory
    b. No two faces of adjacent indices are guaranteed to have contiguous/nearby elements

3. Halfedge Mesh in Scotty3D uses lists to store elements
    a. If we use shared memory (OpenMP), no built-in thread-safe way to emplace new elements
    b. If we use isolated memory (Open MPI), iterators used to store references to new elements will be invalid once merged with other processes

Parallel Solution 1 (OpenMP)

- Implement thread-safe custom emplace functions
  - Use C++’s std::mutex (synchronization primitive for exclusive, non-recursive ownership)
  - Each mesh has a unique pointer to a mutex instance

    => Lock mutex when a thread-safe emplace gets called, unlock when done

- #pragma omp parallel for
  - Experiment with various scheduling options (static, dynamic, guided)
  - Experiment with parallelization conditions (if faces.size() < threshold)

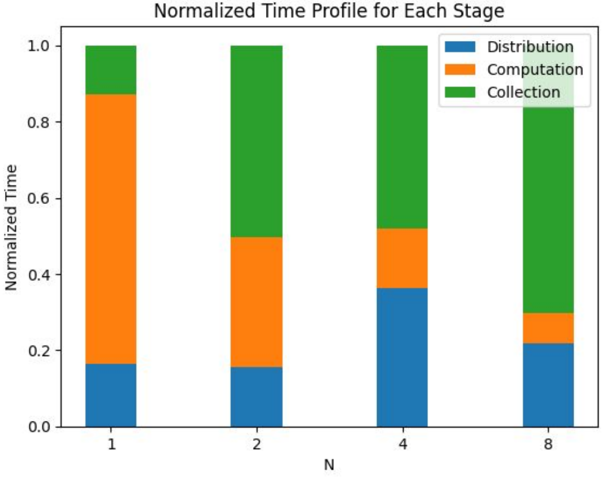
- Dynamically allocated objects (mesh elements) are in shared memory by default

Parallel Solution 2 (Open MPI)

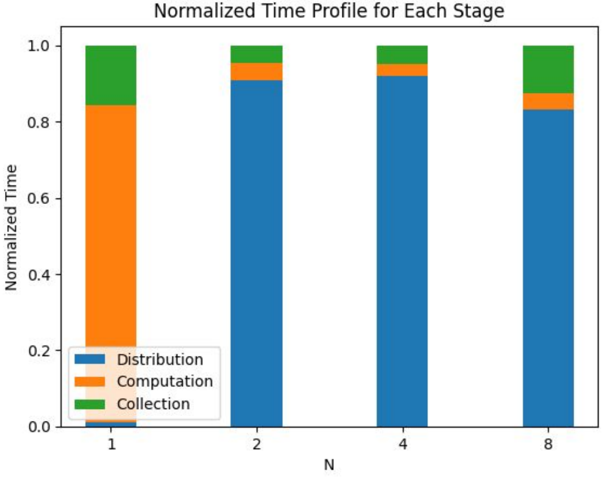
- Change lists to vectors
  - Instead of storing references to iterators within lists, store indices into vectors

- Three-stage implementation
  - Distribution : MPI_Bcst to send relevant mesh structure to each process
  - Computation : Perform fan triangulation, creating new faces, edges, and halfedges
  - Collection : New mesh elements are appended to the end of their respective element vectors
    - Ensure connectivity modifications are propagated when merged
    - Ensure any index changes due to previously merged elements is reflected to new connections

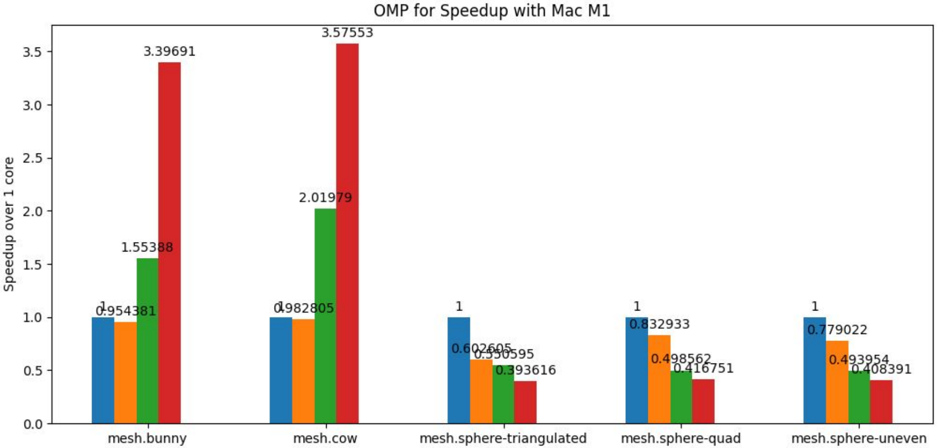
Broadcasting entire mesh

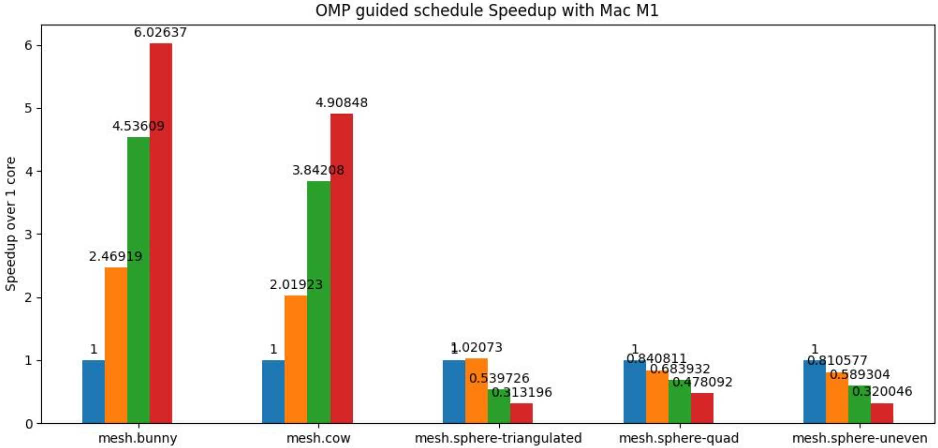


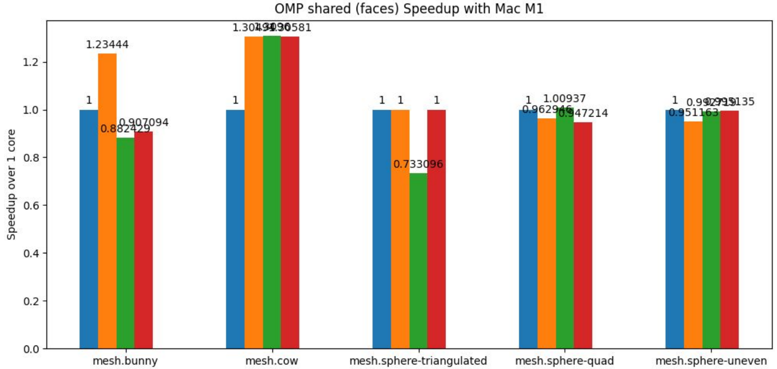
Broadcasting relevant parts only

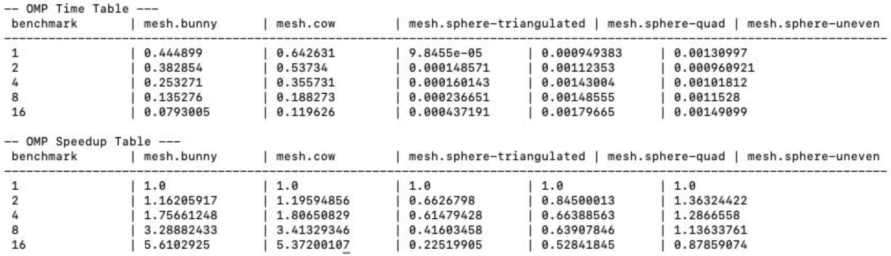


Results - OpenMP





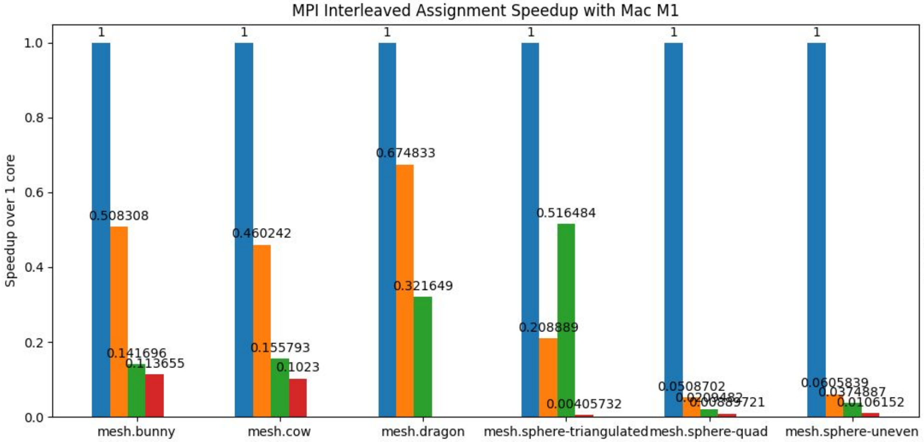


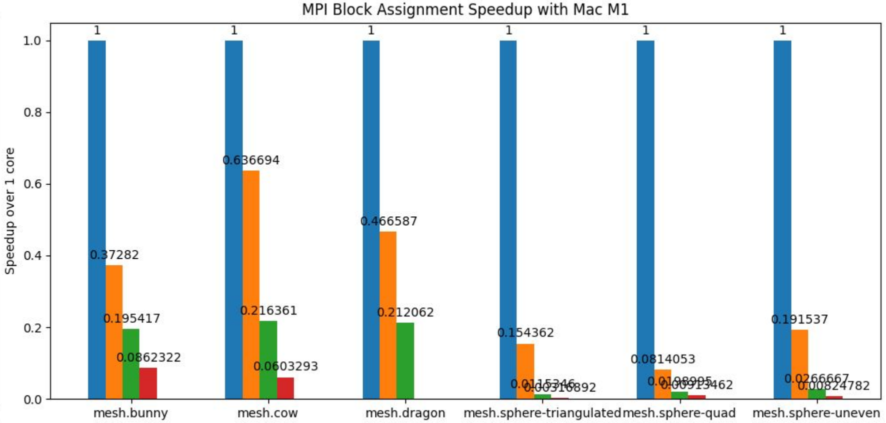


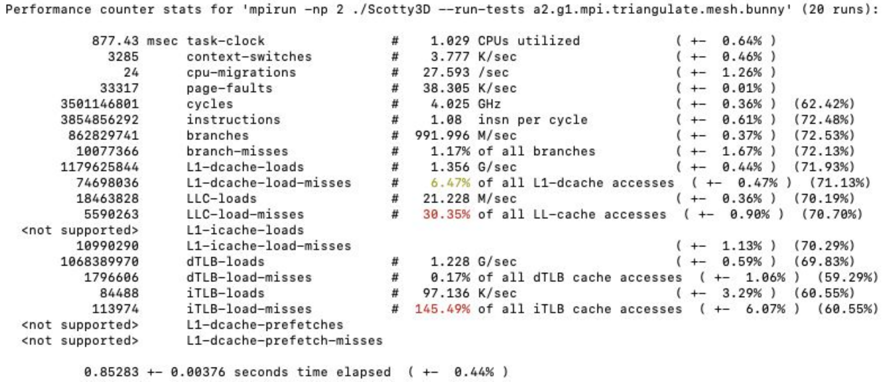
```
-- OMP Time Table ---
 benchmark         | mesh.bunny       | mesh.cow         | mesh.sphere-triangulated | mesh.sphere-quad | mesh.sphere-uneven
-----------------------------------------------------------------------------------------------------------------------------
 1                 | 0.444899         | 0.642631         | 9.8455e-05        | 0.000949383       | 0.00130997
 2                 | 0.382854         | 0.53734          | 0.000148571       | 0.00112353        | 0.000960921
 4                 | 0.253271         | 0.355731         | 0.000160143       | 0.00143004        | 0.00101812
 8                 | 0.135276         | 0.188273         | 0.000236651       | 0.00148555        | 0.0011528
 16                | 0.0793005        | 0.119626         | 0.000437191       | 0.00179665        | 0.00149099

-- OMP Speedup Table ---
 benchmark         | mesh.bunny       | mesh.cow         | mesh.sphere-triangulated | mesh.sphere-quad | mesh.sphere-uneven
-----------------------------------------------------------------------------------------------------------------------------
 1                 | 1.0              | 1.0              | 1.0               | 1.0               | 1.0
 2                 | 1.16205917       | 1.19594856       | 0.6626798         | 0.84500013        | 1.36324422
 4                 | 1.75661248       | 1.80650829       | 0.61479428        | 0.66388563        | 1.2866558
 8                 | 3.28882433       | 3.41329346       | 0.41603458        | 0.63907846        | 1.13633761
 16                | 5.6102925        | 5.37200107       | 0.22519905        | 0.52841845        | 0.87859074
```

Results - OpenMPI



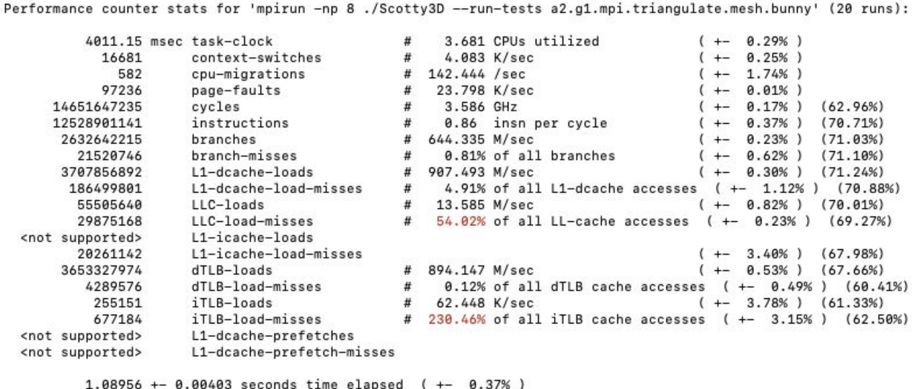




```
 Performance counter stats for 'mpirun -np 2 ./Scotty3D --run-tests a2.g1.mpi.triangulate.mesh.bunny' (20 runs):

            877.43 msec task-clock                #    1.029 CPUs utilized            ( +-  0.64% )
              3285      context-switches          #    3.777 K/sec                    ( +-  0.46% )
                24      cpu-migrations            #   27.593 /sec                     ( +-  1.26% )
             33317      page-faults               #   38.305 K/sec                    ( +-  0.01% )
        3501146801      cycles                    #    4.025 GHz                      ( +-  0.36% )  (62.42%)
        3854856292      instructions              #    1.08  insn per cycle           ( +-  0.61% )  (72.48%)
         862829741      branches                  #  991.996 M/sec                    ( +-  0.37% )  (72.53%)
          10077366      branch-misses             #    1.17% of all branches          ( +-  1.67% )  (72.13%)
        1179625844      L1-dcache-loads           #    1.356 G/sec                    ( +-  0.44% )  (71.93%)
          74698036      L1-dcache-load-misses     #    6.47% of all L1-dcache accesses  ( +-  0.47% )  (71.13%)
          18463828      LLC-loads                 #   21.228 M/sec                    ( +-  0.36% )  (70.19%)
           5590263      LLC-load-misses           #   30.35% of all LL-cache accesses  ( +-  0.90% )  (70.70%)
   <not supported>      L1-icache-loads
          10990290      L1-icache-load-misses                                         ( +-  1.13% )  (70.29%)
        1068389970      dTLB-loads                #    1.228 G/sec                    ( +-  0.59% )  (69.83%)
           1796606      dTLB-load-misses          #    0.17% of all dTLB cache accesses  ( +-  1.06% )  (59.29%)
             84488      iTLB-loads                #   97.136 K/sec                    ( +-  3.29% )  (60.55%)
            113974      iTLB-load-misses          #  145.49% of all iTLB cache accesses  ( +-  6.07% )  (60.55%)
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses

           0.85283 +- 0.00376 seconds time elapsed  ( +-  0.44% )
```



```
 Performance counter stats for 'mpirun -np 8 ./Scotty3D --run-tests a2.g1.mpi.triangulate.mesh.bunny' (20 runs):

           4011.15 msec task-clock                #    3.681 CPUs utilized            ( +-  0.29% )
             16681      context-switches          #    4.083 K/sec                    ( +-  0.25% )
               582      cpu-migrations            #  142.444 /sec                     ( +-  1.74% )
             97236      page-faults               #   23.798 K/sec                    ( +-  0.01% )
       14651647235      cycles                    #    3.586 GHz                      ( +-  0.17% )  (62.96%)
       12528901141      instructions              #    0.86  insn per cycle           ( +-  0.37% )  (70.71%)
        2632642215      branches                  #  644.335 M/sec                    ( +-  0.23% )  (71.03%)
          21520746      branch-misses             #    0.81% of all branches          ( +-  0.62% )  (71.10%)
        3707856892      L1-dcache-loads           #  907.493 M/sec                    ( +-  0.30% )  (71.24%)
         186499801      L1-dcache-load-misses     #    4.91% of all L1-dcache accesses  ( +-  1.12% )  (70.88%)
          55505640      LLC-loads                 #   13.585 M/sec                    ( +-  0.82% )  (70.01%)
          29875168      LLC-load-misses           #   54.02% of all LL-cache accesses  ( +-  0.23% )  (69.27%)
   <not supported>      L1-icache-loads
          20261142      L1-icache-load-misses                                         ( +-  3.40% )  (67.98%)
        3653327974      dTLB-loads                #  894.147 M/sec                    ( +-  0.53% )  (67.66%)
           4289576      dTLB-load-misses          #    0.12% of all dTLB cache accesses  ( +-  0.49% )  (60.41%)
            255151      iTLB-loads                #   62.448 K/sec                    ( +-  3.78% )  (61.33%)
            677184      iTLB-load-misses          #  230.46% of all iTLB cache accesses  ( +-  3.15% )  (62.50%)
   <not supported>      L1-dcache-prefetches
   <not supported>      L1-dcache-prefetch-misses

           1.08956 +- 0.00403 seconds time elapsed  ( +-  0.37% )
```