Development and performance of a PVM based parallel geometric modeler for MEMS

Craig Jorgensen, Darryl Melander, Rod Schmidt and Steve Plimpton

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM, USA, 87185
crjorge@sandia.gov, djmelan@sandia.gov, rcschmi@sandia.gov, sjplimp@sandia.gov

ABSTRACT

This paper describes a successful approach to improving the robustness and speed of Sandia National Laboratory's 3-D MEMS geometry modeler through a combination of mask subdivision and code parallelization. Symptoms of the robustness problems experienced with the original modeler that suggested the need for the subdivision strategy pursued here are explained. The basic elements of the subdivision approach and the subsequent parallel implementation are described. Results and timings for a number of different modeling problems are presented and discussed in order to illustrate the effectiveness of the current approach.

Keywords: MEMS, solid modeling, parallel processing, SUMMiT, PVM

1 INTRODUCTION

Driven by the ongoing need to create MEMS devices with greater complexity and improved functionality, fabrication processes are being developed that involve an increasing number of process steps and material layers. For example, the current standard at Sandia National Laboratory is the SUMMiT V^{TM} process [1] which contains 5 layers of polysilicon. However, as the number of process steps and material layers increase, the difficulty and time required to design new devices also increase. The use of accurate, robust, and fast design tools is becoming essential. Of particular importance is the ability to automatically generate and visualize an accurate 3-D solid-body representation of the resulting MEMS device defined by the set of 2-D masks. The resulting solid model can be visually evaluated and numerically analyzed for correctness. This improves the designer's ability to recognize problems and achieve early success with production.

Previous work at Sandia National Laboratory led to the development of a 3-D geometry modeler [2] for the SUMMiT V^{TM} MEMS designer. Based on the ACIS [3] kernel, this code creates a 3-D solid-body representation of a MEMS design by simulating the manufacturing process on a 2-D mask set.

The modeler performed well on many of Sandia's standard component designs and enabled a deeper understanding of the parts. However, for large complex designs and system level devices the code did not provide sufficient speed or robustness for effective use as an everyday design tool. Simple parts could be modeled in a few minutes but complicated designs could take many hours or even days to finish. In addition geometric processing errors that are related to the geometry kernel became more common as the complexity of the mask set increased causing unrecoverable problems with the process simulation.

These problems severely hampered the effectiveness of the modeler by discouraging analysis of the large and complex parts where understanding is needed most. Furthermore, since MEMS devices are only expected to become more complicated in the future, it was clear that without significant improvements the modeler would become increasingly less useful.

This paper describes a successful approach to improving the robustness and speed of the geometry modeler through a combination of mask subdivision and parallelization.

2 SUBDIVISION STRATEGY

Experience with the modeler demonstrated that the likelihood of failure increased with the complexity and size of the problem. Of particular note was that failure would often occur on system designs that were completely composed of MEMS devices that by themselves could be successfully modeled. This observation suggested that a subdivide-and-model strategy should be explored. The basic idea is to take a large complex design, break it up into smaller simpler pieces, model the pieces, and then combine the modeled pieces into the fully modeled complex design.

There are four steps to this strategy.

1. Read in the mask set.
2. Subdivide the mask set into simpler sub-mask sets.
3. Model each of the simple sub-mask sets.
4. Join the results of the modeling.

The second step is key to this approach and it is important enough to describe further. The subdivision psuedo code can be written as follows.

```
Subdivide (mask_set, simple_set_collection)
{
    Calculate mask_set complexity
    If complexity > complexity threshold then
        Divide mask_set into N simpler
            sub_mask_sets
        For I = 1 to N
            Subdivide (sub_mask_set(I),
                simple_set_collection)
    Else
        Add mask_set to simple_set_collection
}
```

The current subdivision algorithm estimates the complexity of the mask set based simply on the number of vertices. The threshold is a user specified input parameter. If the set contains more vertices than the threshold, the masks are spatially bisected to create a number of sub-mask sets. These sub-mask sets are then re-evaluated by the subdivision algorithm. Eventually the entire initial mask set is divided into pieces consider simple enough for modeling.

Figure 1 illustrates the subdivide-and-model strategy for a simple problem requiring two levels of subdivisions. In this example, the bottom left quadrant contains many more vertices than the other quadrants, leading to the need for a second subdivision.

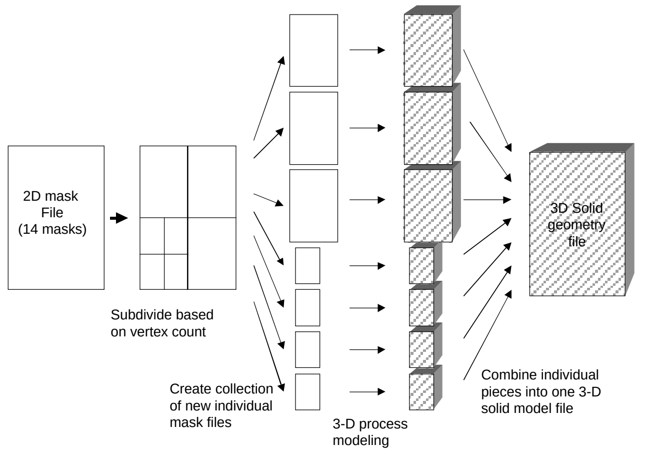
2D mask File (14 masks)
Subdivide based on vertex count
Create collection of new individual mask files
3-D process modeling
Combine individual pieces into one 3-D solid model file
3D Solid geometry file

Figure 1 Illustration of the subdivide-and-model strategy.

3 PARALLELIZATION

Figure 1 also helps to illustrate that many steps in the overall process could, in principle, be accomplished concurrently. The most obvious are the modeling tasks, each of which are independent from each other. However, much of the work in the subdivision and joining steps can also be accomplished through sets of independent tasks.

The parallelization approach taken is based on a master-slave strategy and was implemented by using the PVM software package (Parallel Virtual Machine) [4]. PVM is a software system that permits a heterogeneous collection of networked computers to be viewed by a user's program as a single parallel computer. In this case, a controlling "master" code was written which reads in the initial problem definition through input files, identifies and spawns tasks to available slave processors, monitors the status of each task, and stops when all tasks are finished and the overall process is completed. While the master code is run on the users workstation, the spawned tasks are assigned out to any of the available workstations within the PVM network.

We define three types of tasks, called subdivide, model, and join tasks. These tasks are linked through a set of dependencies that reflect the sequential aspects of the overall problem. For example, mask set subdivision must always occur first, and for each subdivision that occurs, an associated join task must later be performed to recombine the sub-masks that have been modeled.

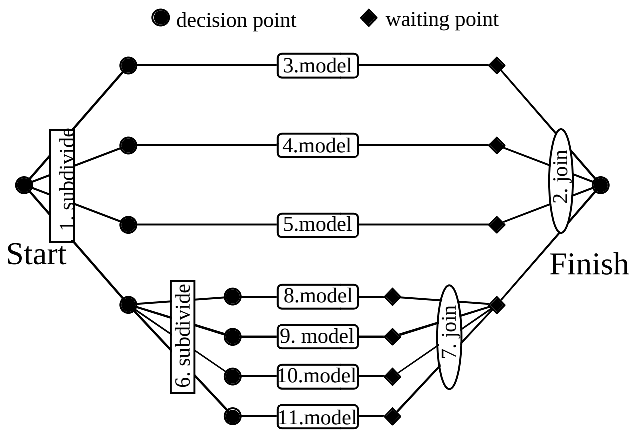

Figure 2 An example task flow chart for the problem illustrated in Figure 1.

Figure 2 shows the task flow chart for the simple problem illustrated in Figure 1. Each of the three task types are illustrated, together with what are denoted decision points and waiting points. Decision points correspond to when the complexity of the current mask set is evaluated to determine if further subdivision is necessary, or if a modeling task can begin (explained in Section 2). Waiting points correspond to the completion of modeling task, when the code must potentially wait until each of the parts required for a joining task has been modeled.

After the input files have been read in, the master code begins by determining if the problem is complex enough to require subdivision. In the example illustrated a subdivision is necessary, and a subdivide task (task 1) and an associated join task (task 2) are added to a task list maintained by the master code. Task 1 can begin immediately. However, task 2 must be put on hold until all the associated sub-mask regions are modeled and ready for joining. This process continues with appropriate tasks being identified at each decision point along the process path.

Multiple tasks can be accomplished simultaneously by allocating the tasks to different processors. For example, after the initial subdivide (task 1) is complete, three model tasks (tasks 3-5) and one subdivide task can be run concurrently on four separate processors. As any processor completes a task, it is then free to be assigned to the next task in the task list. As can be seen, the overall process in this case is accomplished as the sum of 11 separate tasks – two subdivide tasks, seven model tasks and two join tasks.

Figures 3 and 4 show some examples of the models that were successfully generated with the subdivide strategy described in this paper.

4 RESULTS

The parallel modeler was tested and benchmarked on a small network of SGI workstations. Nine machines were used connected by a standard 100 Mbit Ethernet LAN. The cluster was heterogeneous with each machine having one or two MIPS R10000 or R12000 processors ranging in speed from 175 to 400 MHz. The slowest machine in the cluster was used as the "master" and the other boxes (1 processor/machine) as "slaves"; timings are presented for 1 to 8 slaves. Geographically, the machines reside in two buildings a mile apart.

Procs	Split	Model	Join	Total	Speed Up
1	0	1156	0	1156	-
1	194	498	30	722	1.00
2	181	634	72	544	1.33
4	181	446	72	371	1.95
8	194	457	56	366	1.97

Table 1: CPU time (in seconds) to model a flexlink with 1289 mask vertices on different numbers of SGI processors. Total is the wall-clock time for the run; split/model/join times are sums across all participating processors.

Performance data for a modest-sized flexlink device with 1289 mask vertices is given in Table 1. The first entry is a run without subdivision. All other runs used a splitting threshold of 1000 vertices. Due to the geometrically asymmetric layout of the flexlink, four 4-way splits were performed, dividing the model into 13 pieces. By itself, the subdivision strategy reduced the time significantly. Furthermore, an additional speed-up of about 2x was achieved on 4 processors; adding more processors was not effective due to the small size of the problem.

Ideally the split/model/join times would be identical for any number of processors, since they are the sum of times required to perform the individual tasks. In practice, these numbers vary from run to run, chiefly depending on what processor they run on. This is due to the speed of the processor and the current load on the machine (which can be in interactive use on someone's desktop).

Table 2 gives timings for a larger microengine model with 34591 mask vertices. This design is interesting because the 3-D Modeler alone can not successfully model it (see entry 1 of Table 2). The splitting threshold was set at 500 vertices, which produced a total of 87 splits and 262 sub-models. This problem takes 14.5 hours to model on one processor and little over 4 hours with 8-processors. We again see some variability in the split/model/join timings due to processor differences. Another effect that limits our parallel speed-up is the cost of doing the initial few splits. The very first split is the most expensive, since it operates on the full model. It requires over 4000 seconds, during which time only 1 processor is active. Likewise, the second level split runs more quickly but only engages 4 processors. A similar effect occurs with joining operations at the end of the run (though joins are less expensive); the last join that creates the final model can also only run on one processor.

Procs	Split	Model	Join	Total	Speed Up
1	0	23053*	0	failed	failed
1	14030	29569	1544	45145	1.00
2	14266	35975	2202	30947	1.46
4	14447	33057	2105	19528	2.31
8	16087	29373	2169	15212	2.97

* *Modeler fails at this point*

Table 2: CPU time (in seconds) to model a microengine with 34591 mask vertices. Total is the wall-clock time for the run; split/model/join times are sums across all participating processors. *Modeler fails after

5 SUMMARY

In this paper we have presented a technique that has substantially improved the robustness and speed of Sandia Nationals Laboratory's 3-D MEMS geometry modeler through a combination of mask subdivision and code parallelization.

The strategy of mask subdivision has been shown to enable the modeler to in principle handle arbitrarily complex designs. It also improves, by itself, the speed of the modeling. The flexlink example in Table 1 requires 1156 seconds to model with the 3-D Modeler alone. The same flexlink design requires only 722 seconds on a single processor with the subdivide approach.

Additionally, mask subdivision leads to the definition of distinct tasks that can be accomplished concurrently – thus allowing for additional speed-up through the use of parallel processing. We took advantage of this feature by implementing a distributed system using PVM on a network of workstations. This resulted in further speed-ups for the modeling.

Subdivision is an easily implemented and useful technique for improving geometric modeling. While this paper demonstrates its effectiveness with a geometry modeler based on the ACIS kernel, we also expect that the strategy will prove useful with other modeling approaches.

6 ACKNOWLEDGEMENT

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

REFERENCES

[1] M. S. Rodgers, J. J. Sniegowski, "Designing Microelectromechanical Systems-on-a-Chip in a 5-Level Surface Micromachine Technology", 2nd International Conference on Engineering Design and Automation, 1998.

[2] C. R. Jorgensen, V. R. Yarberry, "A 3D Geometry Modeler for the SUMMiT V MEMS Designer", Modeling and Simulation of Microsystems, 594-597, 2001.

[3] ACIS by Spatial Corporation, http://www.spatial.com.

[4] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing, The MIT Press, 1994. Available on internet at http://www.netlib.org/pvm3/book/pvm-book.html.

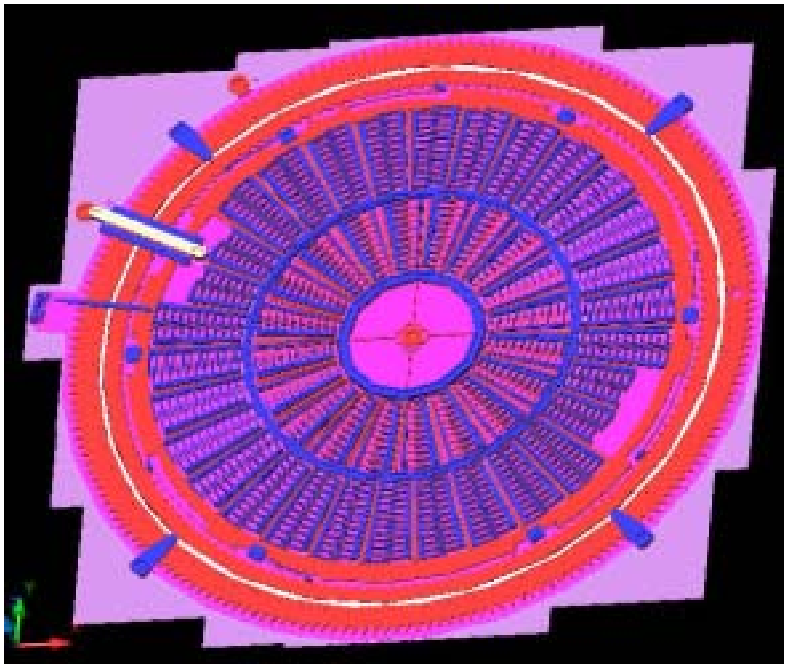

Figure 3: Solid model of Torsional Ratchet Actuator (TRA) with 13,833 vertices.

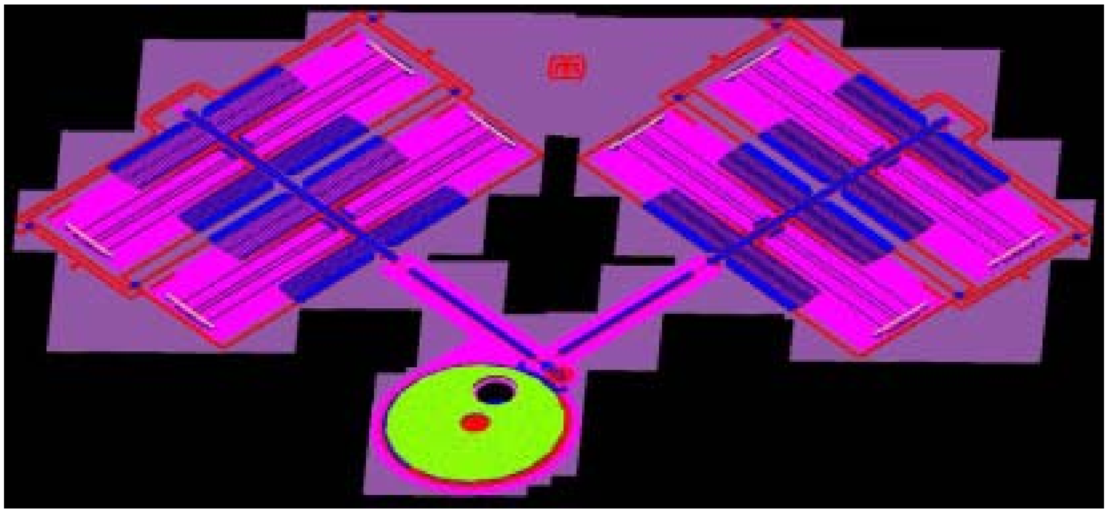

Figure 4: A solid model of a MEMS system consisting of two comb drives driving a gear which drives a second larger gear. It contains 20,188 vertices.