

## Lecture 9: Sources of Parallelism and Locality in Simulation - Part 2

*Lecturer: Horst Simon**Scribes: Isaac Liu*

**Disclaimer:** *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

## 9.1 Review

Common problems:

- Load Balancing - unstructured problems could be distraction from good performance if not done right. Easier if you have a structured grid or else need to come up with scheme to do load balancing.
- Locality - small surface to volume ratio increases locality.

ODEs and sparse matrix:

- In principle, sparse matrices are important kernel for solving ODEs.
  - explicit - don't really have to solve it, use spMV
  - implicit - use direct solvers or iterative solvers
- In conclusion, studying algorithms in detail is relatively easy, but still have problems of load balancing and locality.
- SpMV in compressed sparse row:
  - Stored only the non zeros (colored). Try to compute  $y = A * x$ . End up with the loop.
- Partitioning:
  - First start with row partitioning
  - More sophisticated. The ideal is focusing all the non zeros as diagonal blocks. So any ordering wants to come close to the ideal partitioning.
  - Graph partitioning

### 9.1.1 PDEs

Second half of Math 228

3 fundamental types:

- Elliptic problems
  - Favorite elliptic problem - a drum, if you push the membrane down, you get the membrane to displace according to force.

- Not time dependent. It describes a steady state. Any time instance of the steady state changes the solution of the problem.
- Global space dependency
- Hyperbolic problems
  - Example - A signal traveling. Dip your finger in water and watch the water traveling
  - Time dependent problem
- Parabolic problems
  - Heat flow - diffusion or heat flow. If you put a heat source in the middle of a stick and watch the temperature increase throughout the rod.
- Many systems and problems combine features above.

Heat flow example:

- A bar of uniform material, insulated except at the ends.
- Look in slides for equations (page 12)
- Explicit method (page 13)
  - Reordering the equations, we get the temperature at time+delta is a weighted sum of the temperature at time 3 points in the past.
  - If we map it to discrete grid, it looks like grid on page 14.
  - The matrix view is on page 15, and it becomes matrix multiply problem.
  - General way of solving explicitly is approximating some partial differential equation by some finite difference approach, come to a grid problem, and generate the matrix problem, and then solve the matrix problem, which is where you think about parallel computing.
  - Often used for hyperbolic and parabolic equations.
  - Problems with explicit approach is numerical instability, especially with parabolic equations. Basically, if you use too big of time step, the solution is not close to reality. This is the fundamental limitation of the explicit method.
- Implicit method (page 18)
  - You get the next time step by solving a linear equation.
  - Use backwards Euler to approximate time derivative.
  - From explicit to implicit, we shift the matrix from right to left, so the sign changes.
  - Another way is to use the trapezoid rule instead of backwards Euler to give better numerical properties.

Poisson's equation example (slide 20):

- Elliptic problem. Look at Equation in slide. Think of it as you have a particle in 3D space, and you look at the particle at location  $(x,y,z)$  due to particle at point 0. Then force is proportional to radius.
- As you discretize the problem, you will also arrive at the laplacian value.

- So to solve this, you have to solve a complex sparse matrix problem.
- Prototypical sparse matrix problem. So slide 22 gives an overview of algorithms to solve the sparse matrix problem (a big piece of math 221), slide 23 lists the complexity.
- All algorithms have different complexity characteristics.
- Mflops/s Verses Run time in practice - cited from Shadid and Tunimaro, SIAM parallel processing conference, March 1991
  - The paper implemented 3 solvers on a parallel machine - Jacobi, gauss-seidel (simpler of Red/Black SOR) and multigrid. The CPU time results weren't surprising corresponding to complexity of sequential processor. But the algorithmic complexity makes the implementation more difficult and you don't get much performance.
  - The overall message is that the algorithmic complexity (sequential complexity) still gives you the best speed even though performance isn't good. In general, the most efficient sequential algorithm will still get you speed.

### Summary

- There are either explicit or implicit approach possible
- Explicit - sparse matrix vector multiplication (iterative solver)
- Implicit -sparse matrix solve at each step
- Difference in algorithms and efficiency

## 9.2 Practical Meshes

- So far we just talked about 1D, 2D, or 3D regular meshes, in reality, meshes are typically irregular.
- Generally, there are these type of meshes:
  - Composite meshes - multiple bent regular meshes
  - Unstructured meshes - tetrahedral
  - Adaptive meshes - Add local refinement when you get to physically important phenomenon.
- Best way of portioning meshes is low surface to volume ratio
- Ghost regions - regions neighboring processors that contain information that you may need
- Composite mesh - To exploit parallelism, you keep many of the local data structures on one processor or one set of processors, and then subdivide further. Slide 29 shows what it looks like when you convert it to a sparse matrix
- Matrix reordering - If you apply matrix reordering, but you might get fill in, so there's a lot of work being done in reducing the fill in. But this just helps with sequential computation, not parallelism.
- Unstructured mesh example
  - NASA air foil
  - Tapered Tube

- Vertebra
- Adaptive mesh refinement (AMR) - Computing errors in physical simulation, if errors are large, then you refine the calculations. Each path is a structured grid, but they are overlapping in an unstructured way.
- Challenges of Irregular meshes
  - How to generate them
  - How to partition them
  - How to design iterative solvers
  - How to design direct solvers

## 9.3 Summary

Mapping what we've learned so far back to the 7 dwarfs

- Structured grids - Laplacian, stencil
- Unstructured grids - Last couple of slides
- Spectral methods - FFT
- Dense linear algebra - Wednesday, next lecture
- Sparse linear algebra - Sparse matrix multiply
- Particle methods - Last week
- Monte Carlo - Haven't covered yet, but easy

## 9.4 Final Projects

- Teams of 3 students, typically across departments
- Interesting parallel application or system
- Conference-quality type paper (10-page)
- High performance is key, understanding performance, tuning, scaling. It's even more important than the difficulty of the problem
- Leverage projects in other class (discuss with prof first) or research projects