2013 Spring CS 267 Assignment #0

Name: Jue Chen
Email: juechen (at) math (dot) Berkeley (dot) edu
Bio:

I am a third year Ph.D. student in math department. I also work in the Computational Research Division in Lawrence Berkeley National Lab. My research is focused on solving PDE by using a variety of numerical methods, including: finite element method, finite difference method and compact finite difference method, etc. My current research is searching for time periodic solutions for two-dimensional overturned wave by using boundary integral method and conformal mapping. I hope to get sufficient knowledge and experience to be able to implement scientific computations on a variety of parallel platforms from this class.

Research Application: Parallel multigrid algorithm

Searching for three-dimensional standing waves has always been offering a number of technical challenges, including the require of significantly more computational resources: simulating the wave itself requires one more dimension, and the number of degrees of freedom parameterizing the configuration space over which to search over is also larger.

One of the intermediate steps in [1] uses a finite element method with fourth order elements– this computationally expensive step is solved using a parallel multigrid algorithm. They write a parallel geometric multigrid algorithm in C++ using the OpenMPI library that can take advantage of the problem's structure. It is essentially equivalent to solving a linear system *Ax=b*, where *A* is a sparse matrix, *b* is a source term, and *x* is the unknown quantity.

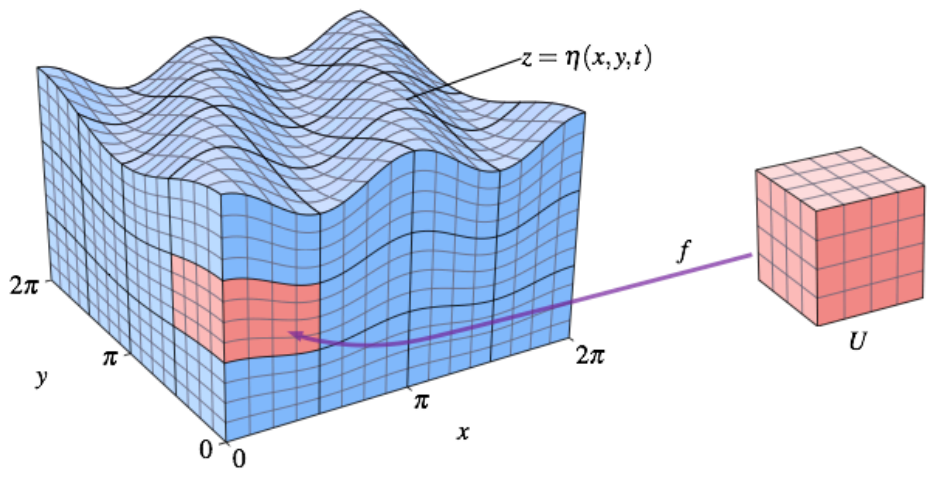


Figure 1: Illustration of the finite-element formulation, in which the bulk of the fluid is divided into $4 \times 4 \times 4$ patches that are scaled according to the height of the fluid $z = \eta(x,y)$. To compute the finite element matrix, a reference space $U = [0,4]^3$ is introduced, and a map f from U to one of these patches is considered.

In the algorithm, the computational grid is labeled as the zeroth grid, and a hierarchy of progressively coarser grids labeled from 1 to G are introduced. The multigrid algorithm is carried out in parallel by dividing the computational domain into a rectangular grid of 2^p columns. During the multigrid algorithm, each processor is responsible for storing the parts of the solution within its column, and the corresponding rows of the sparse matrix A. Each processor is assigned a unique index, and has a table of the eight processors that are either orthogonally or diagonally adjacent.

In certain parallel architectures (e.g. a cluster of multi-core processors), it is faster to communicate between processors with similar indices, and thus the assignment of the different columns in the multigrid algorithm is done in a way to maximize the proportion of communication between processors with similar indices. To do this, a processor's index is viewed as a binary number, and the odd-numbered bits are used to create the x position of the column, while the even-numbered bits are used to create the y position of the column. An example processor decomposition on sixteen processors into a 4×4 grid is shown in Fig.2.

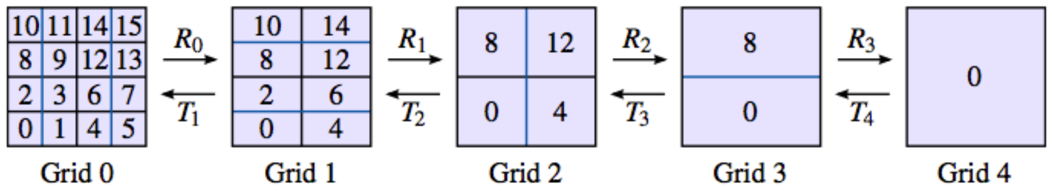
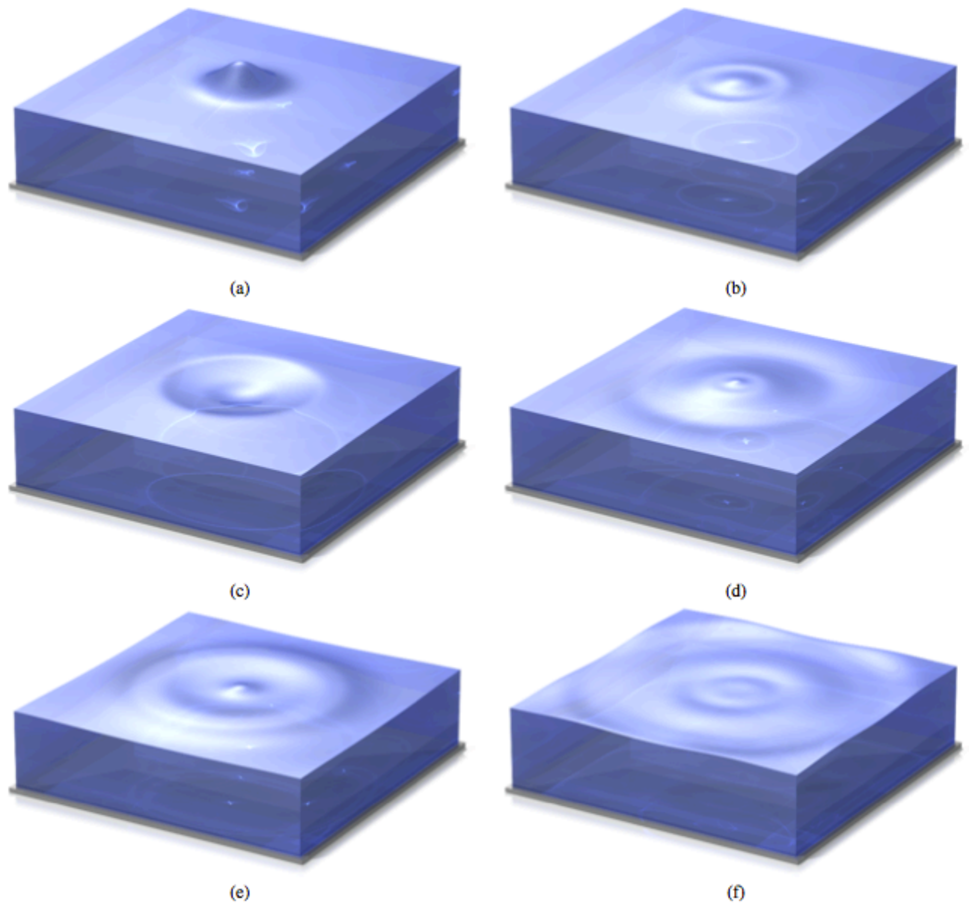


Figure 2: Typical multigrid hierarchy when using sixteen threads. On the finest grid 0, the domain is divided into a 4×4 grid, that is assigned to the threads via a binary numbering scheme as described in the text. At each coarser level, the number of threads involved is halved, and the domains are amalgamated between the remaining threads.

To initially set up the linear system, the following recursive procedure is carried out to compute the system at level $g+1$ using the information at level g:

1. Processors at level g communicate the bound information $(i_\pm, j_\pm, k_\pm)$ in edge strips to their neighbors. Using this, and their own bound information, they calculate the bound information at level $g+1$.
2. Processors at level g with indices of the form $2^g(2k+1)$ communicate their bound information to those with the form $2^g(2k)$.
3. Processors at level $g+1$ use the bound information to calculate the precise amount of memory required for their representation of the linear system at level $g+1$, and allocate it.
4. Processors at level g communicate edge strips of the linear system to their neighbors. Using this, and their own information they calculate the linear system at level $g+1$.
5. Processors at level g with indices of the form $2^g(2k+1)$ communicate the calculated linear system to those with the form $2^g(2k)$.

Figure 3: Snapshots of an example computation, in which the fluid is initially at rest, and the surface is flat apart from a small Gaussian peak in the center of the simulation domain. Snapshots of the system at $t = 0, 1, 2.5, 4.5, 6.5, 9.5$ are given in (a) to (f) respectively, showing how the peak collapses and a ripple moves radially outwards.



Given the difficulties of computation, whereby calculating a single time periodic solution can take several days using 16 threads, the numerical results are of relatively low resolution when compared to two-dimensional studies. And the results cover only very small part of the possible range of three-dimensional time periodic solutions that may exist, but serve to highlight some interesting questions for further study.

Reference:

1. Chris H. Rycroft, Jon Wilkening, Computation of three-dimensional standing water waves