11th. World Congress on Computational Mechanics (WCCM XI)
5th. European Conference on Computational Mechanics (ECCM V)
6th. European Conference on Computational Fluid Dynamics (ECFD VI)
July 20 - 25, 2014, Barcelona, Spain

LOAD BALANCING FOR MULTIPHYSICS

Rainald Löhner[1] and Joseph D. Baum[2]

[1] CFD Center, MS 6A2, SPACS, George Mason University
Fairfax, VA 22030-4444, USA, rlohner@gmu.edu, http://www.scs.gmu.edu/ rlohner
[2]Advanced Technology Group, Leidos,
McLean, VA 22020, USA, joseph.d.baum@leidos.com

Key words: *Load Balancing, Multiphysics Problems, Parallel Computing.*

Over the last decade it has become clear that the only way to higher CPU performance (loosely speaking: more floating point operations per second [FLOPS]) is via massive parallelism. This can be achieved (and is pursued) at the level of the chip (either via many cores or via specialized hardware, e.g. GPUs), via a network of chips, or via a combination of both approaches. In fact, most of the Top 500 supercomputers at present use a combination of this kind to achieve outstanding performance.
For field solvers, which are commonly used for computational fluid and solid mechanics as well as electromagnetics, the classic way to distribute work among many distributed memory processors is via domain decomposition. Given that the work requirements are proportional to the number of elements/points in a domain, the aim is to achieve subdomains of equal size while minimizing communication. As the communication between processors is proportional to the surface points of each subdomain, the aim is to minimize surface-to-volume ratios, which is achieved by keeping the domains as contiguous (non-split) and 'spherical' as possible. Techniques commonly used for domain splitting include the advancing front methods, coordinate- and moment- recursive bisection, and space-filling curve subdivisions. With the possibility of computing larger problems, the desire to compute physics of ever increasing complexity also emerged. Some current flow applications include a traditional (i.e. unreacting) flow solver, chemical reactions, moving embedded or immersed bodies, particles, etc. A timestep for such an application may proceed as follows: a) Identify the (new) position of embedded/immersed bodies, obtaining the new boundary conditions/geometric parameters required; b) Advance the chemical reactions one timestep, obtaining the source-terms for the flow solver; c) Update the particles one timestep, obtaining the source-terms for the flow solver; d) Advance the flowfield one timestep. The order of these operations is not mandatory and will vary among field solvers. What is important, though, is that at the end of each of these steps a synchronization among processors is required: the calculation can not proceed until all processors have completed each step in turn. Therefore, for optimal performance **the load should be balanced in each step**. This inherent requirement of all multiphysics

solvers implies that, compared to simple field solvers (e.g. just flow), the potential for load imbalances and suboptimal performance increases substantially.

An obvious way out of this potential dilemma is to load balance each of the components separately. This approach has been pursued repeatedly but implies a large amount of message passing, as all the different partitions required for the individual physics modules have to communicate with each other.

The way pursued in the present effort is to partition the load into into $m \cdot N_p$ subdomains, where N_p is the number of available (MPI) processors and m is proportional to the number of different multiphysics models (flow, chemical reactions, embedded surfaces, particles, etc.) used in each timestep to advance the solution. These $m \cdot N_p$ subdomains are then agglomerated into N_p larger domains using a heap-based greedy algorithm, attempting to equidistribute the work for each of the different multiphysics models as much as possible.

As an example, we consider a relatively long tube where a detonation occurs. As the blast wave reaches the walls of the tube, particles are introduced into the flowfield. The number of elements is of $O(5 \cdot 10^7)$, while the number of particles eventually reaches $O(2 \cdot 10^6)$. The problem was run with 16 distributed memory (mpi) processes/domains, and 8 shared memory cores (OpenMP) per domain, i.e. a total of 256 cores. For the present purpose, the problem was run for 1000 steps, with a re-split using the method described above every 100 steps. The splitting obtained at the end of the 1000 steps may be discerned in Figures 2a-c. Note that there are many more domains than mpi-processes/domains, but that, as expected, the basic moment-based recursive bisection has still produced 'slices' along the tube. The breakdown of times is approximately as follows: flow solver 60%, particle update 30%, repartitioning and renumbering 10%. This implies that the parallel repartioning does not lead to an excessive increase in CPU requirements while allowing for a much better load balance.

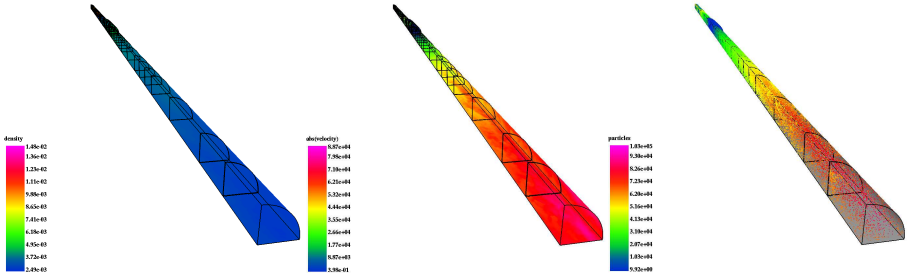

Figure 1 Blast In Tube With Particles