Computer Physics Communications 184 (2013) 1297–1309

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Parallel grid library for rapid and flexible simulation development☆

I. Honkonen [a,b,*], S. von Alfthan [a], A. Sandroos [a], P. Janhunen [a], M. Palmroth [a]

[a] Finnish Meteorological Institute, Helsinki, Finland
[b] Department of Physics, University of Helsinki, Helsinki, Finland

ARTICLE INFO

Article history:
Received 24 May 2012
Received in revised form
5 October 2012
Accepted 14 December 2012
Available online 29 December 2012

Keywords:
Parallel grid
Adaptive mesh refinement
Free open source software

ABSTRACT

We present an easy to use and flexible grid library for developing highly scalable parallel simulations. The distributed cartesian cell-refinable grid (dccrg) supports adaptive mesh refinement and allows an arbitrary C++ class to be used as cell data. The amount of data in grid cells can vary both in space and time allowing dccrg to be used in very different types of simulations, for example in fluid and particle codes. Dccrg transfers the data between neighboring cells on different processes transparently and asynchronously allowing one to overlap computation and communication. This enables excellent scalability at least up to 32 k cores in magnetohydrodynamic tests depending on the problem and hardware. In the version of dccrg presented here part of the mesh metadata is replicated between MPI processes reducing the scalability of adaptive mesh refinement (AMR) to between 200 and 600 processes. Dccrg is free software that anyone can use, study and modify and is available at https://gitorious.org/dccrg. Users are also kindly requested to cite this work when publishing results obtained with dccrg.

Program summary

Program title: DCCRG
Catalogue identifier: AEOM_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEOM_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU Lesser General Public License version 3
No. of lines in distributed program, including test data, etc.: 54975
No. of bytes in distributed program, including test data, etc.: 974015
Distribution format: tar.gz
Programming language: C++.
Computer: PC, cluster, supercomputer.
Operating system: POSIX.
The code has been parallelized using MPI and tested with 1–32768 processes
RAM: 10 MB–10 GB per process
Classification: 4.12, 4.14, 6.5, 19.3, 19.10, 20.
External routines: MPI-2 [1], boost [2], Zoltan [3], sfc++ [4]
Nature of problem:
Grid library supporting arbitrary data in grid cells, parallel adaptive mesh refinement, transparent remote neighbor data updates and load balancing.
Solution method:
The simulation grid is represented by an adjacency list (graph) with vertices stored into a hash table and edges into contiguous arrays. Message Passing Interface standard is used for parallelization. Cell data is given as a template parameter when instantiating the grid.
Restrictions:
Logically cartesian grid.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Correspondence to: P.O. Box 503, 00101 Helsinki, Finland. Tel.: +35 8503803147; fax: +35 8295394603.
E-mail addresses: ilja.honkonen@fmi.fi, ilja.honkonen@helsinki.fi (I. Honkonen).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2012.12.017

Running time:
Running time depends on the hardware, problem and the solution method. Small problems can be solved in under a minute and very large problems can take weeks. The examples and tests provided with the package take less than about one minute using default options.

In the version of dccrg presented here the speed of adaptive mesh refinement is at most of the order of 10^6 total created cells per second.

References:

[1] http://www.mpi-forum.org/.
[2] http://www.boost.org/.
[3] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, C. Vaughan, Zoltan data management services for parallel dynamic applications, Comput. Sci. Eng. 4 (2002) 90–97. http://dx.doi.org/10.1109/5992.988653.
[4] https://gitorious.org/sfc++.

1. Introduction

During the rising phase of the solar cycle, it is becoming more important to understand the physics of the near-Earth space. The dynamical phenomena caused by the constant flow of magnetized collisionless plasma from the Sun creates space weather that may have harmful effects on space-borne or ground-based technological systems or on humans in space. While the physics of space weather is being studied with in situ instruments (e.g. NASA's Radiation Belt Storm Probes launched in 2012-08-30[1]) and by means of remote sensing, it is also important to model the near-Earth space with numerical simulations. The simulations can be used both as a context to the one-dimensional data sets from observations, as well as a source to discover new physical mechanisms behind observed variations. Present large scale (global) simulations are based on computationally light-weight simplified descriptions of plasma, such as magnetohydrodynamics (MHD, [1–4]). On the other hand the complexity and range of spatial scales (from less than 10^1 to over 10^6 km) in space weather physics signifies the need to incorporate particle kinetic effects in the modeled equation set in order to better model, for example, magnetic reconnection, wave–particle interactions, shock acceleration of particles, ring current, radiation belt dynamics and charge exchange (see e.g. [5] for an overview). However, as one goes from MHD towards the full kinetic description of plasma (from hybrid PIC [6] and Vlasov [7] to full PIC [8,9]), the computational demands increase rapidly, indicating that the latest high performance computing techniques need to be incorporated in the design of new simulation architectures.

As the number of cores in the fastest supercomputers increases exponentially the parallel performance of simulations on distributed memory machines is becoming crucial. On the other hand, utilizing a large number of cores efficiently in parallel is challenging especially in simulations using run-time adaptive mesh refinement (AMR). This is largely a data structure and an algorithm problem albeit specific to massively parallel physical simulations running on distributed memory machines.

In computer simulations dealing with, for example, continuous matter (a fluid) the simulated domain is discretized into a set of points or finite volumes which we will refer to as cells. At any given cell the numerical solution of a differential equation describing the problem often depends only on data within a (small) part of the simulated volume. This is true for a single time step in a solver for a hyperbolic problem or a single iteration in a solver for an elliptic problem. This spatial data dependency can be implemented implicitly in the solver function(s) or explicitly as a separate grid library used by the application.

In a simple case the number of cells in the simulation stays constant and the data dependency of each cell is identical allowing cell data to be stored in an array whose size is determined at grid creation and the spatial neighbors to be represented as indices into this array. A straightforward AMR extension of this concept is to create additional nested grids in specific parts of the simulation domain with higher resolution. By solving each grid separately and interpolating the results from finer grids into coarser grids one does not have to modify the solver functions. This technique is used extensively for example by Berger (see [10] for some of the earliest work) and by [11,12]. In the rest of this work however we will concentrate on AMR implementations in which additional overlapping grids are not created but instead cells of the initial grid are refined, i.e. replaced with multiple smaller cells.

A generic unstructured grid (as provided for example by libMESH [13]) does not admit as simple a description as above and is generally described by a directed graph in which vertices represent simulation cells and directed edges represent the data dependencies between cells. Unfortunately the nomenclature of graph theory and geometry overlap to some extent and discussing both topics simultaneously can lead to confusion. Fig. 1 shows the nomenclature we use from this point forward, the standard graph theoretical terms are given in parentheses for reference. A cell is a natural unit in simulations using the finite volume method (FVM) and hereinafter we will use the term cell instead of vertex when discussing graphs. Also an edge in FVM simulations usually refers to the edges of a cube representing the physical volume of a cell, and hence we will use the term arrow to refer to a directed edge in a graph. Furthermore we note that each cell in the grid can also represent, for example, a block of cells similarly to [3], but for the purposes of this work the actual data stored in grid cells is largely irrelevant.

Since a graph can also be used to represent the cells and arrows of grids simpler than an unstructured mesh, the question arises how does a particular program implement its graph representation of the simulated system, e.g. what simplifying assumptions have been made and how is the graph represented in memory? A popular representation in (M)HD AMR simulations is to have a fixed number of arrows directed away from each source cell and to store the arrows as native pointers to the destination cells. In case a cell does not exist all arrows pointing to it are invalidated in neighboring cells. This technique has been used with different variations by [14–17], for example.

There are several possibilities for representing the cells and arrows of a graph, for example an adjacency list or an adjacency matrix [18]. In physical simulations the number of arrows in the graph is usually of the same order as the number of cells in which

[1] http://www.nasa.gov/mission_pages/rbsp/main/index.html