Geophysical Research Abstracts
Vol. 17, EGU2015-6469, 2015
EGU General Assembly 2015
© Author(s) 2015. CC Attribution 3.0 License.

Implementing the PM Programming Language using MPI and OpenMP – a New Tool for Programming Geophysical Models on Parallel Systems

Tim Bellerby

University of Hull, Geography Environment and Earth Sciences, Geography Environment and Earth Sciences, Hull, United Kingdom (t.j.bellerby@hull.ac.uk)

PM (Parallel Models) is a new parallel programming language specifically designed for writing environmental and geophysical models. The language is intended to enable implementers to concentrate on the science behind the model rather than the details of running on parallel hardware. At the same time PM leaves the programmer in control – all parallelisation is explicit and the parallel structure of any given program may be deduced directly from the code.

This paper describes a PM implementation based on the Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) standards, looking at issues involved with translating the PM parallelisation model to MPI/OpenMP protocols and considering performance in terms of the competing factors of finer-grained parallelisation and increased communication overhead. In order to maximise portability, the implementation stays within the MPI 1.3 standard as much as possible, with MPI-2 MPI-IO file handling the only significant exception. Moreover, it does not assume a thread-safe implementation of MPI.

PM adopts a two-tier abstract representation of parallel hardware. A PM *processor* is a conceptual unit capable of efficiently executing a set of language tasks, with a complete parallel system consisting of an abstract N-dimensional array of such processors. PM processors may map to single cores executing tasks using cooperative multi-tasking, to multiple cores or even to separate processing nodes, efficiently sharing tasks using algorithms such as work stealing. While tasks may move between hardware elements within a PM processor, they may not move between processors without specific programmer intervention. Tasks are assigned to processors using a nested parallelism approach, building on ideas from Reyes *et al.* (2009). The main program owns all available processors. When the program enters a parallel statement then either processors are divided out among the newly generated tasks (number of new tasks $<$ number of processors) or tasks are divided out among the available processors (number of tasks $>$ number of processors). Nested parallel statements may further subdivide the processor set owned by a given task. Tasks or processors are distributed evenly by default, but uneven distributions are possible under programmer control. It is also possible to explicitly enable child tasks to migrate within the processor set owned by their parent task, reducing load unbalancing at the potential cost of increased inter-processor message traffic.

PM incorporates some programming structures from the earlier MIST language presented at a previous EGU General Assembly, while adopting a significantly different underlying parallelisation model and type system. PM code is available at www.pm-lang.org under an unrestrictive MIT license.

Reference

Ruymán Reyes, Antonio J. Dorta, Francisco Almeida, Francisco de Sande, 2009. Automatic Hybrid MPI+OpenMP Code Generation with llc, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science Volume 5759*, 185-195