TACC Technical Report IMP-00

The IMP Elevator Pitch

Victor Eijkhout*

February 22, 2016

* `eijkhout@tacc.utexas.edu`, Texas Advanced Computing Center, The University of Texas at Austin

Abstract

A one-page statement of ‘why IMP’.

The following IMP reports are available or under construction:

IMP-00 The IMP Elevator Pitch
IMP-01 IMP Distribution Theory
IMP-02 The deep theory of the Integrative Model
IMP-03 The type system of the Integrative Model
IMP-04 Task execution in the Integrative Model
IMP-05 Processors in the Integrative Model
IMP-06 Definition of a ‘communication avoiding’ compiler in the Integrative Model
IMP-07 Associative messsaging in the Integrative Model (under construction)
IMP-08 Resilience in the Integrative Model (under construction)
IMP-09 Tree codes in the Integrative Model
IMP-10 Thoughts on models for parallelism
IMP-11 A gentle introduction to the Integrative Model for Parallelism
IMP-12 K-means clustering in the Integrative Model
IMP-13 Sparse Operations in the Integrative Model for Parallelism
IMP-14 1.5D All-pairs Methods in the Integrative Model for Parallelism (under construction)
IMP-15 Collectives in the Integrative Model for Parallelism
IMP-16 Processor-local code generation (under construction)
IMP-17 The CG method in the Integrative Model for Parallelism (under construction)
IMP-18 A tutorial introduction to IMP software (under construction)
IMP-19 Report on NSF EAGER 1451204.
IMP-20 A mathematical formalization of data parallel operations
IMP-21 Adaptive mesh refinement (under construction)
IMP-22 Implementing LULESH in IMP (under construction)

Parallel programming is hard and getting harder. On future (exascale, manycore) architectures, most likely some combination of distributed and shared memory programming will be needed. Some people maintain that current tools can and will suffice, and that programmability can be managed, for instance, by using Domain-Specific Languages (DSLs). We argue that there is opportunity for a new, general, parallel programming system.

Let's take a step back. Many scientific applications do not use the full generality of our parallel programming systems. Typical operations, such as in linear algebra, graph operations, finite elements, can be formulated as a sequential program of operations on distributed data. A system that allows for expression of such 'sequential semantics', without spelling out the coordination of parallel processes underneath, would clearly be a Good Thing.

Now, such programming systems have been tried, see for instance High Performance Fortran (HPF), but they have largely failed because the system software can not be smart enough in deriving messages and task dependencies and such. The Integrative Model for Parallelism (IMP) is built on a new theoretical formalism that lets the programmer specify the necessesary minimum to let the system generate an efficient execution. And this minimum is far less than what is currently required in MPI or various Directed Acyclic Graph (DAG) models. What's more, it is an expression that is independent of the target mode of parallelism, so the same source code works for MPI and DAG models and hybrid modes.

We claim at least that IMP allows for easier programmability with essentially identical execution efficiency. However, IMP has the potential for going beyond that.

1. IMP uses the 'inspector-executor' paradigm, where first a task graph is built, which is subsequently executed. This means that the inspector can already re-arrange tasks, for instance to hide latency.
2. Since task can be migrated and duplicated, more sophisticated 'communication avoiding' strategies can be realized with minimal hints from the programmer.
3. Load balancing and resilience through task replication can also be realized.

We have built a demonstration prototype that shows that identical IMP source code can be translated to efficient execution in MPI and OpenMP task-based implementation layers. Other execution models can most likely also be supported, implying that IMP is 'future-proof'.