NestStepModelica – Mathematical Modeling and Bulk-Synchronous Parallel Simulation

Christoph Kessler, Peter Fritzson
PELAB Programming Environments Lab, Department of Computer Science
Linköping University, S-581 83 Linköping, Sweden
{chrke, petfr}@ida.liu.se

Abstract

The majority of parallel computing applications are used for simulation of complex engineering applications and/or for visualization. To handle their complexity, there is a need for raising the level of abstraction is specifying such applications using high level mathematical modeling techniques, such as the Modelica language and technology.

However, with the increased complexity of modeled systems, it becomes increasingly important to use today's and tomorrow's parallel hardware efficiently. Automatic parallelization is convenient, but may need to be combined with easy-to-use methods for parallel programming.

In this context, we propose to combine the abstraction power of Modelica with support for shared memory bulk-synchronous parallel programming including nested parallelism (NestStepModelica), which is both flexible (can be mapped to many different parallel architectures) and simple (offers a shared address space, structured parallelism, deterministic computation, and is deadlock-free). A prototype version of NestStepModelica is being developed, and preliminary results from using its run-time system are already available.

1 Introduction to Mathematical Modeling and Modelica

Modelica is a modern language for equation-based object-oriented mathematical modeling which is being developed through an international effort [4] [8]. It allows defining simulation models in a declarative manner, modularly and hierarchically and combining various formalisms expressible in the more general Modelica formalism. The multidomain capability of Modelica allows combining electrical, mechanical, hydraulic, thermodynamic, etc., model components within the same application model.

To summarize, Modelica has improvements in several important areas:

- *Object-oriented mathematical modeling*. This technique makes it possible to create model components, which are employed to support hierarchical structuring, reuse, and evolution of large and complex models covering multiple technology domains.
- *Acausal modeling*. Modeling is based on *equations* instead of assignment statements as in traditional input/output block abstractions. Direct use of equations significantly increases re-usability of model components, since components adapt to

the data flow context in which they are used. However, for interfacing with traditional software, algorithm sections with assignments as well as external functions/procedures are also available in Modelica.

- *Physical modeling of multiple application domains*. Model components can correspond to physical objects in the real world, in contrast to established techniques that require conversion to "signal" blocks with fixed input/output causality. In Modelica the structure of the model becomes more natural in contrast to block-oriented modeling tools. For application engineers, such "physical" components are particularly easy to combine into simulation models using a *graphical* editor.

2 Integrating Parallelism and Mathematical Models

There are several approaches to exploit parallelism in mathematical models:

2.1 Automatic Parallelization of Mathematical Models

Current approaches to extracting parallelism from high level mathematical models can be categorized into three groups:

- Parallelism over the method.
- Parallelism over time.
- Parallelism of the system.

A thorough investigation of the third approach, automatic parallelization over the model equation system, is done in recent work on automatic parallelization (using fine-grained task-scheduling) of mathematical models [1].

2.2 Coarse-Grained Explicit Parallelization Using Components

As automatic parallelization methods have their limits, a natural idea is to let the programmer structure the application into computational components using strongly-typed communication interfaces. Ongoing work using this approach is reported in [9].

2.3 Explicit Parallel Programming

The third approach is providing general easy-to-use explicit parallel programming constructs within the algorithmic part of the modeling language. This is the approach we explore in this paper, with the NestStepModelica language embedded into the algorithmic part of the Modelica language.

3 NestStepModelica

NestStep [7] is a parallel programming language based on the BSP (Bulk-Synchronous Parallel) model [11] which is an abstraction of a restricted message passing architecture and charges a cost for communication. It is defined as a set of language extensions that may be added, with minor modifications, to any imperative programming language, be it procedural or object oriented. The sequential aspect of computation is inherited from the base language, which here is the algorithmic (non-equational) part of Modelica.

The new NestStepModelica language constructs provide shared variables and process coordination. NestStepModelica processes run, in general, on different machines that are coupled by the NestStepModelica language extensions and runtime system to a virtual parallel computer. Each processor executes one process with the same imperative fragment of a simulation program (SPMD), and the number of processes remains constant throughout program fragment execution. Below is an example, `parPrefix`, computing prefix sums in parallel.

```
function parPrefix  "Compute prefix sums in parallel"
  input  Integer[:] arr1(mem="shared", distr="block");
  output Integer[size(arr,1)] arr(mem="shared", distr="block"):=arr1;
protected
  parameter Integer p=noProcessors();
  Integer Ndp = N div p;   // Assume p divides N for simplicity
  Integer[Ndp] prefix;     // Local prefix array
  Integer myPrefixSum;     // Prefix offset for this processor
  Integer sum(mem="shared");// Shared consistent at superstep boundary
  Integer i,j;
algorithm
  j := 1; // In Modelica lowest index for arrays is 1
  step(); // Start of BSP superstep
    forall i in arr loop  // Iterate over local elements in one block
      prefix[j] := sum;
      sum := sum+arr[i];
      j := j+1;
    end for;
    endstepReduce(result=sum, op=Operators.plus,prefixVar=myPrefixSum);
  endstep(); // End of BSP superstep
  j := 1;
  step();
    forall i in arr loop // Put prefix sum results into arr
      arr[i] := prefix[j]+ myPrefixSum;
      j := j+1;
    end for;
  endstep();
end parPrefix;
```

The following is a short overview of some NestStepModelica language constructs:

- `noProcessors()` – gives the total number of processors allocated to this fragment.
- `noProcessorsThisGroup()` – the number of processors in the current group.
- `mem="shared"` – shared memory allocation specifier for a variable or array to be shared between the processors of a group.
- `distr="block"` – block distribution of a distributed shared array.
- `distr="cyclic"` – cyclic distribution of a distributed shared array.
- `step()` – start of a superstep.
- `endstep()` – end of a superstep.

- `neststep(nsubgroups=`*intexpr*`[, subgrouptojoin=`*intexpr*`])` – start of a nested superstep.
- `endneststep()` – end of a nested superstep
- `forall` – a loop where iterations are independent and may be executed in parallel, iterating over the indices of the local partition of a distributed shared array.
- `endstepReduce(result, op [, prefixVar])` – specifies for individual shared variables or array sections how concurrent write conflicts should be resolved by use of a reduction operator at the end of this superstep.

4 Conclusion and Future Work

At the time of writing, the run-time system of NestStepModelica, implemented on top of MPI, is operational. In measurements on a Linux cluster, the NestStepModelica run-time system outperformed the OpenMP implementation (running on top of a distributed shared memory emulation layer) by a factor of up to 30 [10]. A NestStepModelica front end (compiler) is under development as an OpenModelica [3] extension.

This work was supported by Vinnova in the GRIDModelica project.

References

[1] Peter Aronsson and Peter Fritzson. Automatic Parallelization in OpenModelica. In *Proc. of 5th EUROSIM Congress on Modeling and Simulation*, Paris. ISBN 3-901608-28-1, Sept 2004.

[2] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The Paderborn University BSP (PUB) Library. *Parallel Computing*, 29:187–207, 2003.

[3] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In *Sim. News Europe*, 44/45, Dec 2005. www.ida.liu.se/projects/OpenModelica.

[4] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004. See also: www.mathcore.com/drModelica

[5] Jonathan M. D. Hill, Bill McColl, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, and Rob Bisseling. BSPlib: the BSP Programming Library. *Parallel Computing*, 24(14):1947–1980, 1998.

[6] Christoph Kessler. Managing Distributed Shared Arrays in a Bulk-Synchronous Parallel Environment. *Concurrency – Pract. Exp.*, 16:133–153, 2004.

[7] Christoph W. Keßler. NestStep: Nested Parallelism and Virtual Shared Memory for the BSP model. *The Journal of Supercomputing*, 17:245–262, 2000.

[8] The Modelica Association. *The Modelica Language Specification Version 2.2*, March 2005. http://www.modelica.org.

[9] Kaj Nyström, Peter Aronsson and Peter Fritzson. GridModelica - A Modeling and Simulation Framework for the Grid. In *Proc. of the 45th Conference on Simulation and Modelling of the Scandinavian Simulation Society* (SIMS2004), 23-24 September 2004, Copenhagen, Denmark.

[10] Joar Sohl. *A Scalable Run-time System for NestStep on Cluster Supercomputers*. Master thesis LITH-IDA-EX-06/011-SE, IDA, Linköpings universitet, Linköping, Sweden, March 2006.

[11] Leslie Valiant. A Bridging Model for Parallel Computation. *Comm. ACM*, 33(8), August 1990.