Parallel Computing 45 (2015) 18–48

Contents lists available at ScienceDirect

Parallel Computing

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

DaSH: A benchmark suite for hybrid dataflow and shared memory programming models

Vladimir Gajinov [a,b,*], Srdjan Stipić [a,b], Igor Erić [c], Osman S. Unsal [a], Eduard Ayguadé [a,b], Adrian Cristal [d,e]

[a] Barcelona Supercomputing Center, Spain
[b] Universitat Politecnica de Catalunya, Spain
[c] University of Belgrade, Serbia
[d] Artificial Intelligence Research Institute, Spain
[e] Spanish National Research Council, Spain

ARTICLE INFO

Article history:
Available online 28 March 2015

Keywords:
Benchmark suite
Dataflow
Shared memory
Transactional memory
Programming model

ABSTRACT

The current trend in development of parallel programming models is to combine different well established models into a single programming model in order to support efficient implementation of a wide range of real world applications. The dataflow model has particularly managed to recapture the interest of the research community due to its ability to express parallelism efficiently. Thus, a number of recently proposed hybrid parallel programming models combine dataflow and traditional shared memory models. Their findings have influenced the introduction of task dependency in the OpenMP 4.0 standard.

This article presents DaSH – the first comprehensive benchmark suite for hybrid dataflow and shared memory programming models. DaSH features 11 benchmarks, each representing one of the Berkeley dwarfs that capture patterns of communication and computation common to a wide range of emerging applications. DaSH also includes sequential and shared-memory implementations based on OpenMP and Intel TBB to facilitate easy comparison between hybrid dataflow implementations and traditional shared memory implementations based on work-sharing and/or tasks. Finally, we use DaSH to evaluate three different hybrid dataflow models, identify their advantages and shortcomings, and motivate further research on their characteristics.

1. Introduction

Current multicore processors are, for the most part, built on top of the shared memory architecture. However, despite the constant evolution of shared memory models and the rich support for programming shared memory systems, some of the problems remain. One of the major issues with this architecture is the cost of synchronization that often prevents shared-memory programs from scaling to high core counts. Solutions to this problem usually require significant programmer effort. This has caused a renewed interest into dataflow due to its ability to express parallelism efficiently.

The main characteristic of the dataflow model [1,2] is that execution of an operation is constrained only by the availability of its input data. Following a single assignment rule, the dataflow model is able to extract all the parallelism inherent in a

* Corresponding author at: Barcelona Supercomputing Center, Spain.
E-mail address: vladimir.gajinov@gmail.com (V. Gajinov).

http://dx.doi.org/10.1016/j.parco.2015.03.005
0167-8191/

program. Yet, there are many applications, such as a reservation system or chess, for which the state is a fundamental part of the problem. Expressing these problems in a dataflow model is possible, but often inefficient.

Due to diversity of algorithms and applications, it is therefore unlikely that a single programming model can support efficient implementation of the whole application spectrum [3]. Thus, a number of recent proposals aim to combine different well established models into a single programming model that can support a wider range of computation and communication patterns. OmpSs [4], Atomic Dataflow (ADF) [5] and Intel Threading Building Blocks (TBB) [6] are some of the models that have recognized the potential of extending the shared memory model with support for dataflow. While each of these models uses its custom set of applications to prove the concept and demonstrate its potential, there is a necessity for an extensive benchmarks suite that will feature not only applications that lend themselves well to dataflow, but also those for which some other parallel programming paradigm, such as work-sharing or tasking, may be more suitable. To the best of our knowledge, no such benchmark suite exists at the moment.

As its main contribution, this article presents DaSH the first comprehensive benchmark suite for hybrid dataflow and shared memory programming models. While developing DaSH, we have followed the approach from Berkeley that has identified a set of 13 dwarfs that capture patterns of communication and computation common to a wide range of emerging applications [3]. Accordingly, each DaSH benchmark represents an application from a single Berkeley dwarf. Currently, DaSH features 11 benchmarks. For each benchmark, we provide sequential implementation, two shared memory implementations based on work-sharing and/or tasking (one that uses OpenMP and the other that uses TBB) and three dataflow implementations realized using the OmpSs, ADF and Intel TBB Flow Graph hybrid dataflow models.

Our second contribution is the evaluation of these three hybrid dataflow models that demonstrates the usefulness of DaSH for the research of these models. In particular, we show that:

- Hybrid dataflow models support a straightforward implementation of certain types of irregular algorithms by allowing the developer to naturally represent the algorithm, which results in better programmability and/or performance. For example, the dataflow implementation of the sparse LU factorization performs 27% better than the corresponding shared memory implementations.
- The main strength of these models is the ability to eliminate unnecessary barriers and thus expose more parallelism. Most of the performance gain when using these models is a result of the increased parallelism and decreased thread idle time. In addition, we have identified applications for which dataflow provides barrier-free implementation even when the algorithm inherently depends on barriers.
- There are important differences between the three hybrid dataflow models that we evaluate, both in terms of programmability and performance. We identify some of their shortcomings and suggest possible improvements.

2. Hybrid dataflow models

This section provides an overview of hybrid parallel programming models that combine dataflow and shared memory programming. In general, these models are based on execution of tasks that are scheduled, according to the dataflow principles, when their input dependencies are satisfied. Typically, a programmer explicitly defines input and/or output dependencies for each task. This information is then used by the runtime system to construct a task dependency graph that governs the execution of a program. As tasks execute, they produce data on which other program tasks depend on. Once all input dependencies for a given task are satisfied, the task becomes enabled and can be scheduled for execution. Tasks are executed by worker threads and scheduling is typically based on work-stealing.

Contrary to a pure dataflow model, which assumes side-effect free execution of dataflow tasks, a hybrid dataflow model can benefit from the underlying shared memory architecture by allowing dataflow tasks to share data. Particularly, in a pure dataflow model, each time the data is updated a new copy is produced. Inherently, the problem occurs when the updated data is some complex structure, such as an array, which effectively makes updates expensive. Hybrid dataflow model can avoid this problem by treating such complex data as a shared state and updating it in place. Naturally, accessing shared state may require synchronization. In this work we employ transactional memory (TM) [7] for the shared state synchronization because it integrates seamlessly into the dataflow model. In particular, both abstractions exhibit isolation property: dataflow in terms of execution of data dependent tasks and transactional memory in terms of concurrent accesses to the shared state by different sharers.

In this work, we are use OmpSs [4], the ADF model [5] and the Flow Graph extension of the Intel TBB framework [6] to implement the DaSH benchmark suite and establish its practicality for evaluation of hybrid dataflow models. Hence, we continue this section with an overview of these three models.

2.1. OmpSs

OmpSs extends OpenMP with a support for asynchronous parallelism that is based on execution of data-dependent tasks. Specifically, OpenMP task directive is extended with three additional clauses – `in`, `out` and `inout` – that a programmer may use to explicitly declare data dependencies for a task. When a new task is created, the runtime system matches its `in` and `out` dependencies against dependencies of all existing tasks. If the match is found, the new task becomes a successor of the corresponding tasks. This process creates a task dependency graph at runtime. Tasks are scheduled for execution as soon as