User Experiences with a Chapel Implementation of UTS

Claudia Fohry
FG Programmiersprachen/-methodik
FB Elektrotechnik/Informatik
Universität Kassel, Germany
Email: fohry@uni-kassel.de

Jens Breitbart
Lehrstuhl für Rechnertechnik und
Rechnerorganisation/Parallelrechnerarchitektur
Technische Universität München, Germany
Email: j.breitbart@tum.de

***Abstract*—Chapel strives to combine programmability and performance in an architecture-independent way. We focus on programmability, and report on our experiences in implementing the Unbalanced Tree Search (UTS) benchmark with user-level task pools. Our contributions include a discussion on how to code objects that internally contain distributed arrays, as well as suggestions for language design such as support for locale-specific and task-specific data, scalar variable-based reduction, and the omission of const.**

The ambition of high-productivity parallel programming has led to the PGAS programming model and its concretization in a number of implementations. PGAS exposes a shared memory that is split into disjoint partitions, each comprising distinct computing resources that have faster access to local than to remote memory. This talk considers Chapel, which is mostly being developed at Cray.

To assess the expressiveness of Chapel, we implemented the Unbalanced Tree Search (UTS) benchmark [1]. This benchmark allows us to concentrate on task parallelism, remote memory access and related issues. It consists in extracting a highly-imbalanced tree and counting the number of nodes. For given tree shape parameters, a node holds all information about the subtree rooted in it. The information is cryptographically encoded in a node descriptor, which naturally describes a task.

We implemented UTS with a conventional, user-level task pool, which deploys a fixed number of long-running Chapel tasks per locale as workers, and a distributed data structure for mutual work stealing. This setting allowed us to study communication, synchronization, and the interplay between object-orientation and parallelism / distribution in a simple framework. It is different from that of a previous Chapel implementation of UTS [2], but we reused some of its sequential code.

The talk describes our experiences, thereby concentrating on language features that we felt missing or difficult to use, as well as on patterns of language usage. The code can be obtained from the first author's homepage. The talk uses material from a previous paper [3]. The main contributions of the talk are on the usage side

- a pattern for coding objects that internally contain distributed arrays,
- a comparison between use of class vs. record constructs for tree nodes,
- remarks on the use of synchronization variables, and
- performance numbers;

as well as on the language side exposure of issues

- support for locale-specific and task-specific data,
- possibility to spawn tasks on groups of locales,
- scalar variable-based (as opposed to array-based) reduction, and
- omission of `const`.

References

[1] S. Olivier *et al.*, "UTS: An Unbalanced Tree Search benchmark," in *Proc. Workshop on Languages and Compilers for High-Performance Computing*. Springer LNCS 4382, 2006, pp. 235–250.

[2] J. Dinan *et al.*, "Unbalanced Tree Search (UTS) benchmark in Chapel," July 2007, Program source available at https://chapel.svn.sourceforge.net/svnroot/chapel/ trunk-/test/studies/uts/.

[3] C. Fohry and J. Breitbart, "Experiences with implementing task pools in Chapel and X10," in *Proc. 10th Int. Conf. on Parallel Processing and Applied Mathematics*. Springer LNCS 8385, to appear, 2013.