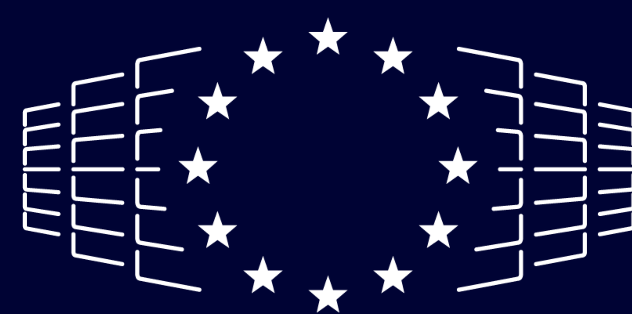


EuroHPC
Joint Undertaking

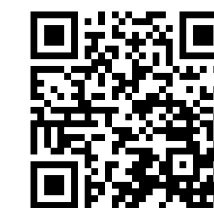
EuroHPC Summit Week

#PRACEdays

Locality-Flexible and Cancelable Tasks for the APGAS Library

Jonas Posner
Research Group Programming Languages / Methodologies
University of Kassel, Germany
jonas.posner@uni-kassel.de

UNIKASSEL VERSITÄT



https://www.uni-kassel.de/go/EuroHPC20

Motivation

- Parallel programs should deal with both shared and distributed memory.
- Many applications deploy locality-flexible tasks that can be run properly on any resource.
- Programmers do not want to worry about load balancing.
- Some applications, such as search problems, benefit from a cancellation if a specific result is reached.
- **Needed: Dynamic load balancing over all resources and an easy to use cancellation option provided by the runtime.**

Background

- The APGAS library is part of IBM's X10 project and implements an asynchronous PGAS (Partitioned Global Address Space) variant in Java.
- A *place* comprises a partition of the global address space and a subset of the computational resources.
- Tasks can be created *asynchronously* on user-specified places.

Design

- For **intra**-place load balancing, each place maintains Java's Fork/Join-Pool with multiple workers.
- For **inter**-place load balancing, each place maintains one *manager* thread, which follows the lifeline scheme.
- A manager monitors its local pool and tries to steal tasks from remote places if the local pool gets empty (coordinated work stealing).
- Stolen tasks are inserted into the local pool and scheduled to all local workers by Java's Fork/Join-Pool.
- Each manager logs its incoming and outgoing steals.
- When a place runs out of tasks, the manager sends its steal log to place 0 and goes inactive.
- An inactive manager is reactivated by a new task in its pool.
- The manager on place 0 accumulates all steal logs and thus performs the global termination detection.
- Cancellation dequeues all system-wide unprocessed tasks.

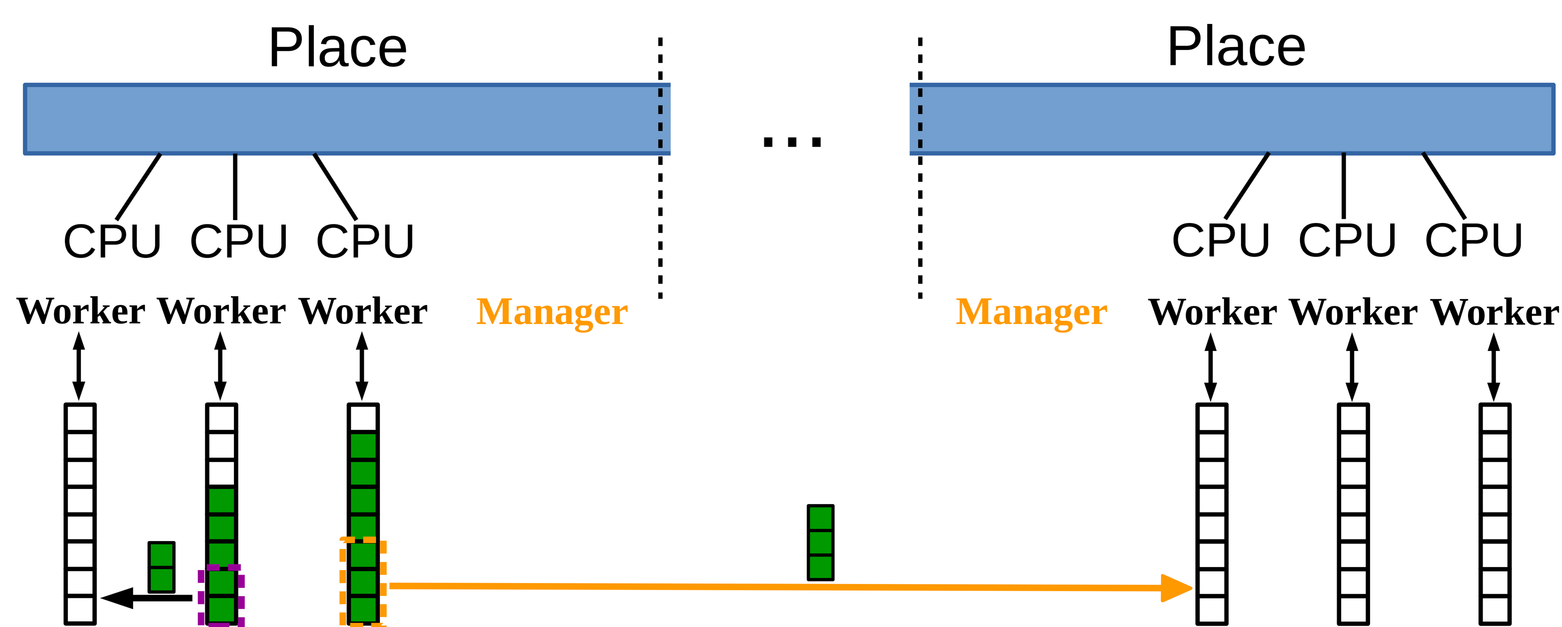


Figure 1: Hybrid Work Stealing for Intra- and Inter-Place Load Balancing

Related Work

- *HabaneroUPC++*: Provides an `asyncAny` construct. In contrast to our implementation, it uses RDMA for victim selection, a dedicated CPU core for work stealing, binds to C++, tasks cannot be canceled.
- *Charm++*: Distributed objects of iterative applications can be migrated automatically between processors using various load balancing strategies.

References

[1] J. Posner and C. Fohry. "A Combination of Intra- and Inter-place Work Stealing for the APGAS Library". In: *Parallel Processing and Applied Mathematics*. Springer, 2018, pp. 234–243.

[2] J. Posner and C. Fohry. "Hybrid Work Stealing of Locality-Flexible and Cancelable Tasks for the APGAS Library". In: *The Journal of Supercomputing* (2018).

[3] Jonas Posner. *Extended APGAS library repository*. 2019. URL: https://github.com/posnerj/PLM-APGAS.

Solution

- We have extended APGAS by dynamic hybrid work stealing, working on shared and distributed memory simultaneously.
- APGAS programmers use independent locality-flexible sequential tasks and do not need to worry about load balancing.
- Results can be easily stored and reduced.
- All system-wide unprocessed tasks can be canceled.
- Usable through some novel constructs, mainly `asyncAny`.

Example: Approximation of π

```
 1 // Blocks until all tasks within have been processed
 2 finishAsyncAny(() -> {
 3  for (long i = 0; i < numTasks; ++i) {
 4   // Submits locality-flexible and cancelable tasks
 5   asyncAny(() -> {
 6    long tmp = 0;
 7    for (long j = 0; j < pointsPerTask; ++j) {
 8     double x = 2 * random() - 1.0;
 9     double y = 2 * random() - 1.0;
10     tmp += (x * x + y * y <= 1) ? 1 : 0;
11    }
12    // Adds the task result to the worker result
13    mergeAsyncAny(new PiRes(tmp, pointsPerTask));
14    // Reduces all worker result and cancels the
15    // calculation when the threshold is reached
16    if (reduceAsyncAny().getResult() > threshold) {
17     cancelAllAsyncAny();
18    }
19   });
20  }
21 });
22 // Reduces all worker results
23 PiRes piRes = (PiRes) reduceAsyncAny();
24 long result = piRes.getResult();
25 long points = piRes.getPoints();
26 System.out.println("Pi=" + 4.0 * result/points);
```

Performance

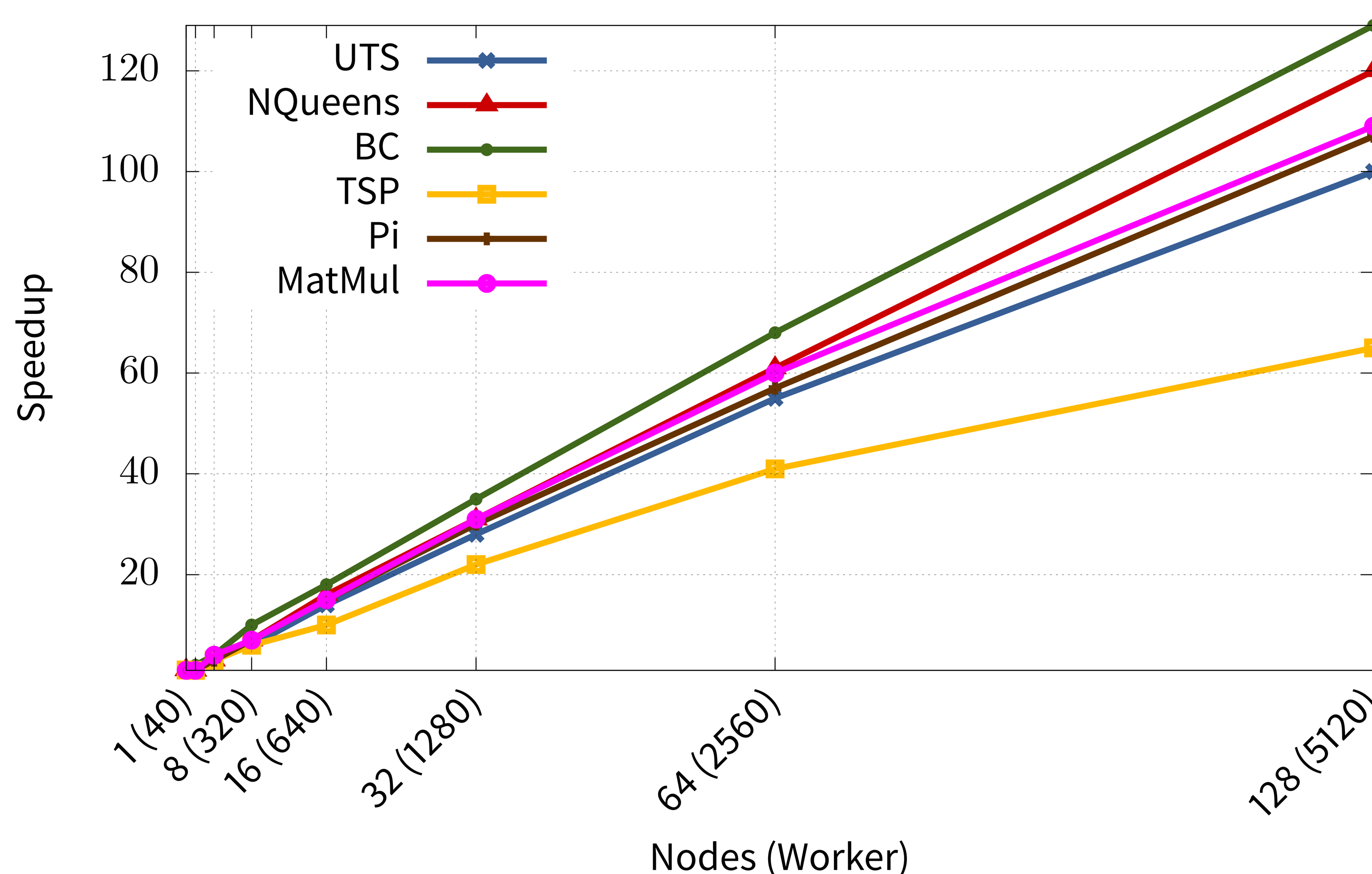


Figure 2: Speedups of Unbalanced Tree Search (UTS), NQueens, Betweenness Centrality (BC), Travel Salesmen Problem (TSP), Pi and Matrix Multiplication (MatMul) with up to 128 Nodes, each with 40 Workers (in total 5120), over execution time of one node (40 Workers). Parallel implementations use `asyncAny` and strong scaling.

Future Work

- Fault-tolerance for `asyncAny` to tolerate permanent place crashes. Crashes are automatically detected and lost tasks are restored from task backups at runtime.
- Malleability for `asyncAny`. Places can be added and removed to running applications and tasks are always distributed accordingly.

The EuroHPC Summit Week conference series receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 800957 (EXDCI-2), 823767 (PRACE-6IP), 824151 (HPC-GIG).

#EHPCSW