Dependency-Aware Parallel Tasks

Afshin Zafari, Martin Tillenius, Elisabeth Larsson

UPPSALA UNIVERSITET

Programming Model

Motivation

- Fine-grained parallelism is required for performance
- Need simpler abstractions
- Abstractions must support expressiveness

Programming Model

- The program is divided up into tasks
- Tasks are annotated with which data they need
- Dependencies are deduced from these annotations

Tasks depend on data,
not on other tasks

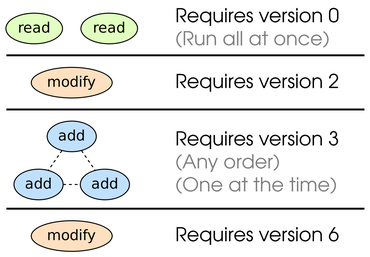
Data Versions

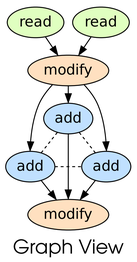
- Versions of task parameters specify dependencies
- More expressive than a DAG
- A task requires a certain version to run

Example

8 tasks accessing the same handle x:

read x, read x, modify x, add x, add x, add x, modify x





Graph View

Distributed Memory Implementation

- All shared data is distributed over the nodes
- Remote data is requested by *listeners*
- When the requested versions become ready, data is sent to all requesting nodes
- Communication is non-blocking
- Viewed as state machine to handle async events

Experiment: Dependencies

Experiment: Cholesky factorization, as an example of non-trivial dependencies.

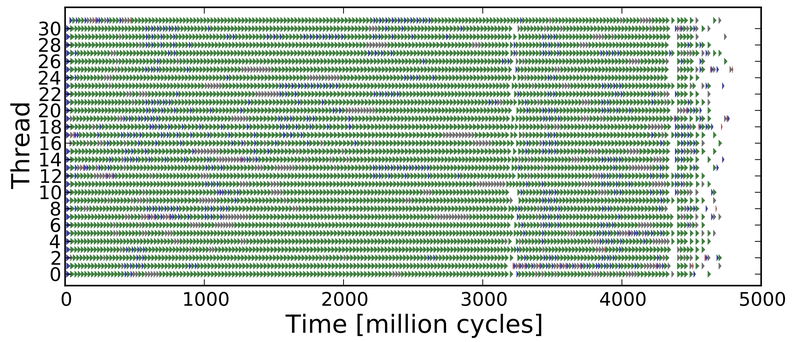


Figure 1: Execution trace of a Cholesky factorization of 32×32 blocks, executed on 32 cores. Each triangle represents a task working on blocks of 256×256 elements. Each task is a call to Intel's Math Kernel Library (MKL).

Speedup: 28.2x on 32 cores, compared to a single MKL call. MKL also has parallel Cholesky, but only achieved 21.1x. MKL might not be optimized for the AMD Bulldozer system this experiment was run on.

Experiment: Scalability

Experiment: Find what task granularity is required for scaling. We avoid effects from shared resources by executing tasks that only delay for a specific time.

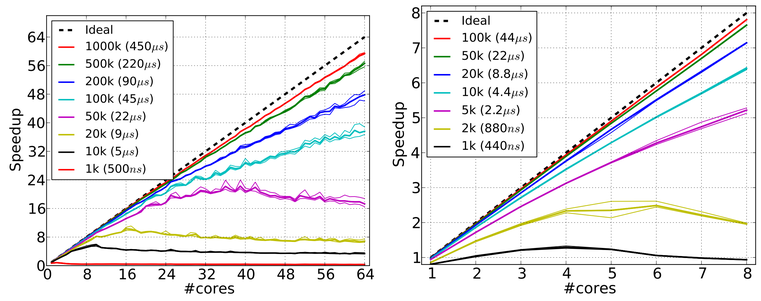


Figure 2: Scaling for different task granularities on (Left) 4 × 16 core AMD and (Right) 2 × 4 core Intel.

On 64 cores; tasks need to take ~500,000 cycles.
On 8 cores; tasks need to take ~20,000 cycles.

Experiment: Distributed Memory

Experiment: Assembly of a matrix, to be used in a partial differential equations solver.

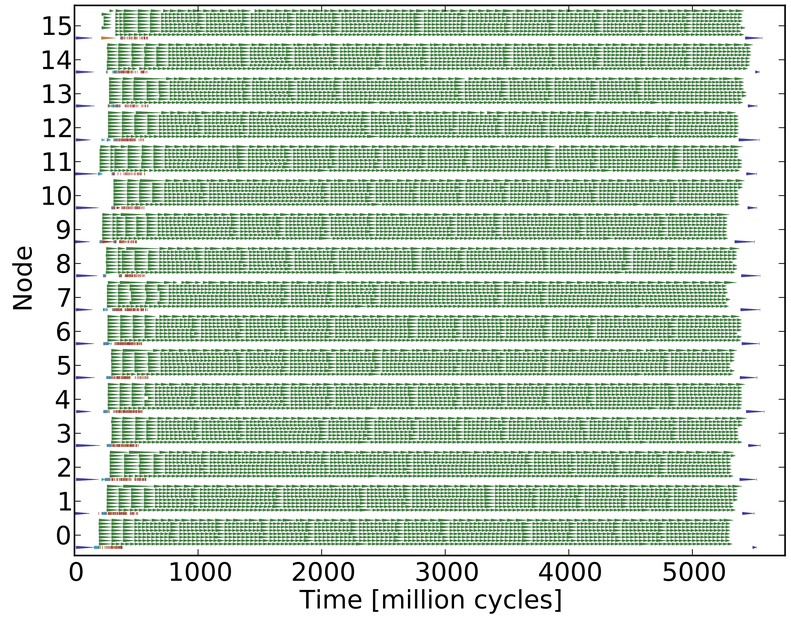


Figure 3: Execution trace from running on 16 nodes with 8 cores each. The matrix is divided into 16,384 blocks of 512×512 elements. Each task writes one block.

Task color meaning			
Green:	Calculation	Blue:	MPI barrier
Cyan:	Create tasks	Red:	MPI receive
Grey:	Other	Orange:	MPI initiate start

Speedup: 97.7x on 128 cores (over best serial speed)

Conclusions

Tasks give easy access to performance

Acknowledgements

Thanks to Kostis Sagonas for providing access the 64 core machine used in parts of the experiments. The other computations were performed on the Tintin cluster provided by SNIC through UPPMAX under project p2009014.