UPPSALA
UNIVERSITET

Chunks and Tasks: A programming model for parallelization of dynamic algorithms

Emanuel H. Rubensson

Division of Scientific Computing
Department of Information Technology
Uppsala University

Oct 16, 2014

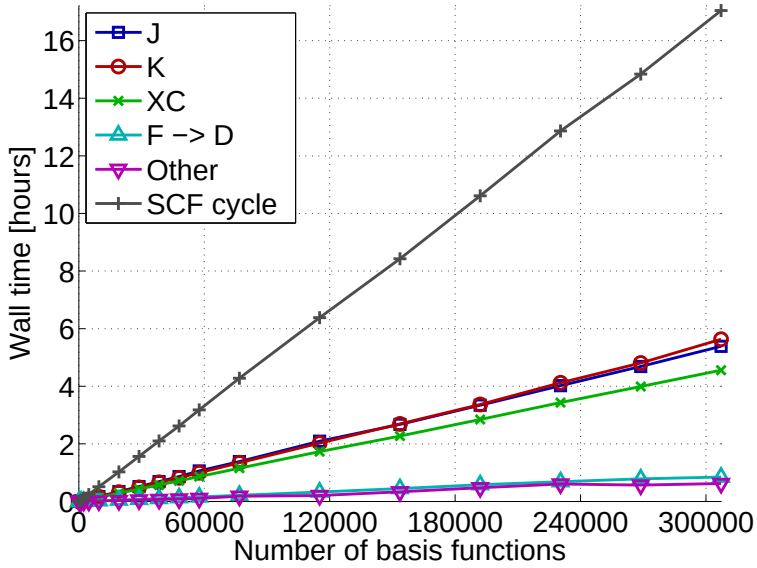
UPPSALA
UNIVERSITET

Motivation

Large-scale electronic structure calculations

The Ergo program (`www.ergoscf.org`)

- Developed by Elias Rudberg, Emanuel H. Rubensson, and Paweł Sałek
- Hartree–Fock and density functional theory
- Linear scaling time and memory usage
- C++ implementation
- Publicly available (GPL license)



UPPSALA
UNIVERSITET

Linear scaling algorithms in Ergo

Dynamic hierarchical algorithms:

- Recursive polynomial expansions for the density matrix
  – Hierarchical block-sparse matrix-matrix multiplication
- Fast multipole method
- Hierarchical computation of the Hartree–Fock exchange matrix

But only parallelized for shared memory

Distributed memory parallelization desirable but challenging

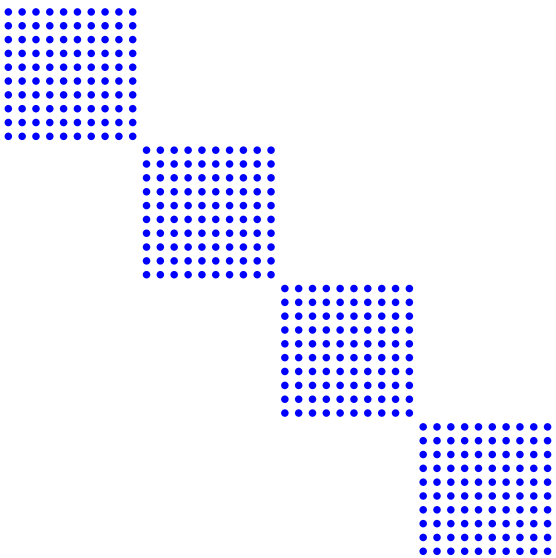
UPPSALA UNIVERSITET

Sparse matrix-matrix multiplication

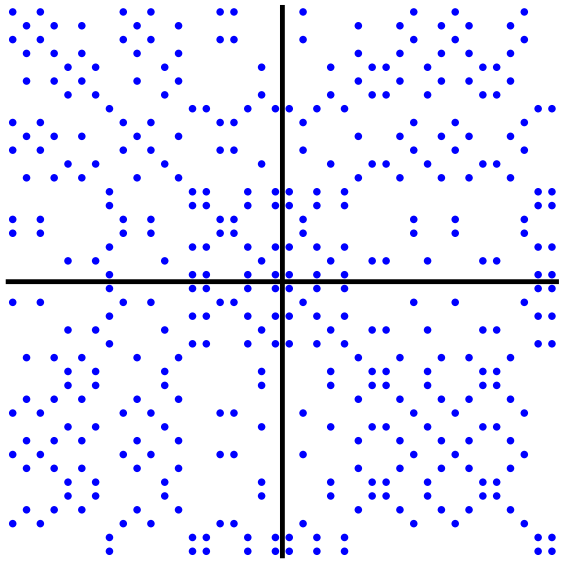
common approach

- Random permutation
- Static data distribution
- Algorithm for dense matrices (Cannon's alg. or SUMMA)

Example matrix



Random permutation



See e.g. A. Buluç, J. R. Gilbert, SIAM J. Sci. Comput. **34**, C170 (2012) and U. Borštnik et al., Parallel Comput. **40**, 47 (2014)

Conventional parallelization (MPI)

Application programmer responsibilities

Conventional parallelization (MPI)

Distribute work

Distribute data

Send and receive messages

UPPSALA
UNIVERSITET

MPI vs Chunks and Tasks

Application programmer responsibilities

Conventional parallelization (MPI)	Chunks and Tasks
Distribute work	Divide work into smaller parts
Distribute data	Divide data into smaller pieces
Send and receive messages	Register “tasks” and “chunks”

Chunks and Tasks

The programming model

- Developed for dynamic hierarchical algorithms
- Abstractions for both data and work (chunks and tasks)
- No explicit communication calls in user code
- Determinism, freedom from race conditions and deadlocks
- Feasible to implement efficient backends
- Fail safety: local recovery possible

How does it work?

Write your program in terms of chunks and tasks.

- Chunk: defined by data members, serialization functions, may contain identifiers to other chunks giving rise to chunk hierarchies.
- Chunk registration: ID = registerChunk(myChunk)
- Task: defined by input chunk types, output chunk type, work to be performed
- Task registration: ID = registerTask(ID, ID, ...)

UPPSALA
UNIVERSITET

Design decisions

compared to related programming models

Computation driven by registration of tasks

Like Cilk, OmpSs, Scioto, StarPU, SuperGlue, XKaapi but unlike Charm++ (where the computation is driven by messages)

$\Longrightarrow$ No message passing

Recursive nesting of tasks allowed

Like Cilk, OmpSs, Scioto, SuperGlue, XKaapi but unlike SmpSs and StarPU

$\Longrightarrow$ Scalable, good for hierarchic dynamic algorithms

Abstractions for both work and data

Previous task-based approaches mostly focused on shared memory. Distributed memory: either user provides data distribution or all data is managed by a "master" node

$\Longrightarrow$ User does not have to provide data distribution

UPPSALA
UNIVERSITET

Design decisions

compared to related programming models

Identifiers to chunks provided by the library

Unlike Linda and Concurrent Collections

$\Longrightarrow$ makes it feasible to make data available efficiently

$\Longrightarrow$ leads to restrictions in how data can be accessed

Chunks are readonly once they are registered

Unlike Linda but similar to Concurrent Collections

$\Longrightarrow$ Chunk cache coherence not an issue

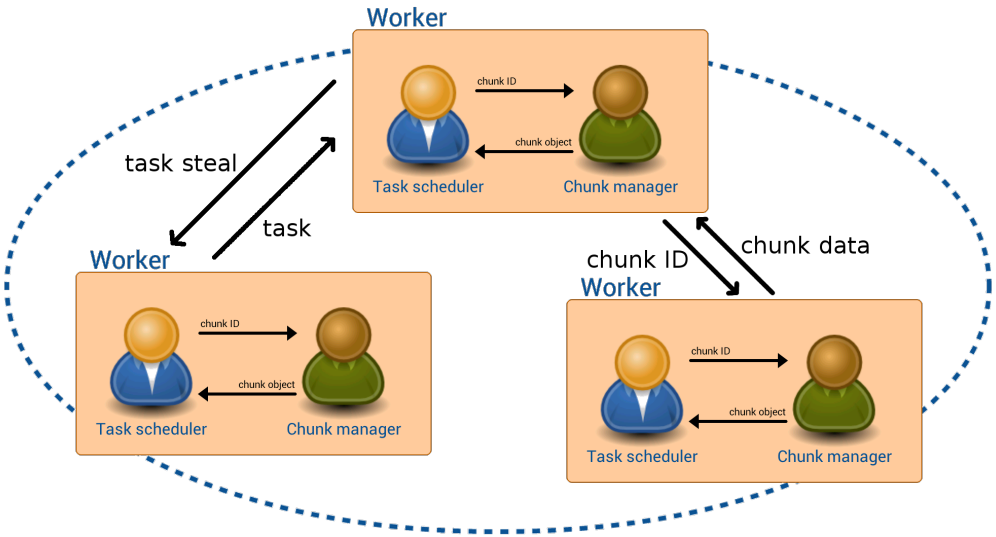
$\Longrightarrow$ Determinism (easier for the user to understand her code)

UPPSALA UNIVERSITET

CHT-MPI

A Chunks and Tasks implementation

- Publicly available since May 2014 (modified BSD license)
- Task scheduler based on work stealing
- Scalable: No "master node" managing all work or data



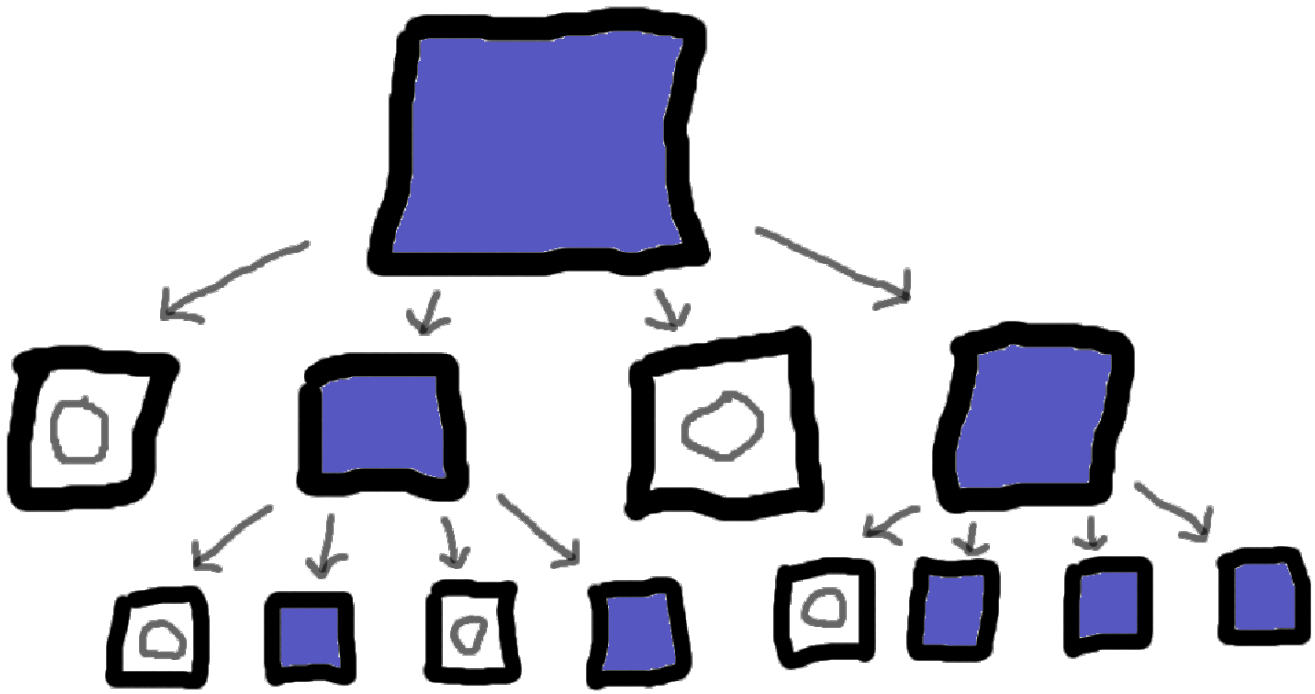
Webpage: `www.chunks-and-tasks.org`

Ref: *Chunks and Tasks: A programming model for parallelization of dynamic algorithms*, Parallel computing **40**, 328 (2014)

UPPSALA
UNIVERSITET

A Chunks and Tasks matrix library

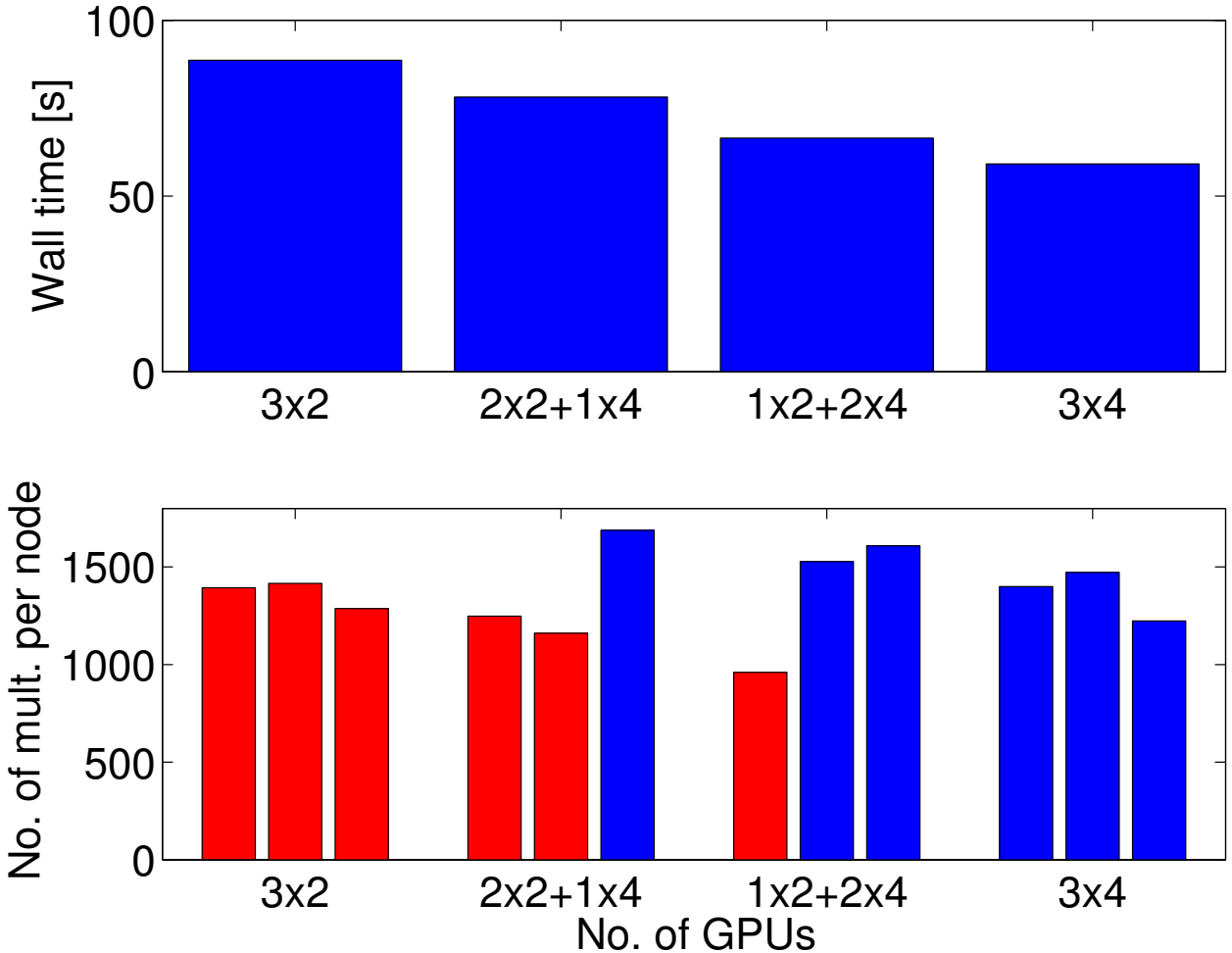
Sparse quad-tree representation



UPPSALA UNIVERSITET

Calculation on the Erik cluster at LUNARC

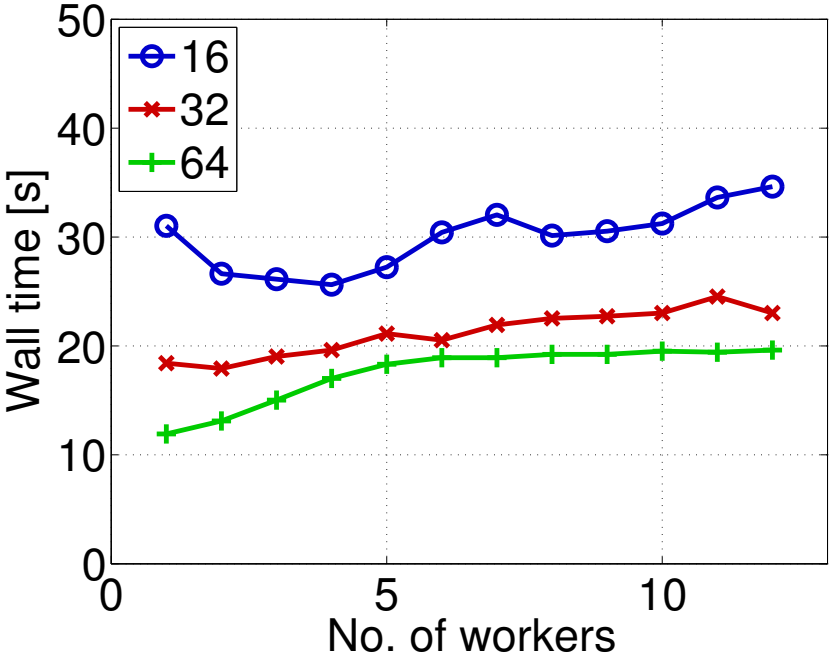
Three nodes with varying number of Nvidia K20 GPUs
(and dual 64-bit, 8-core Intel Xeon E5-2650 2.00 GHz)
Multiplication of two 32768×32768 matrices

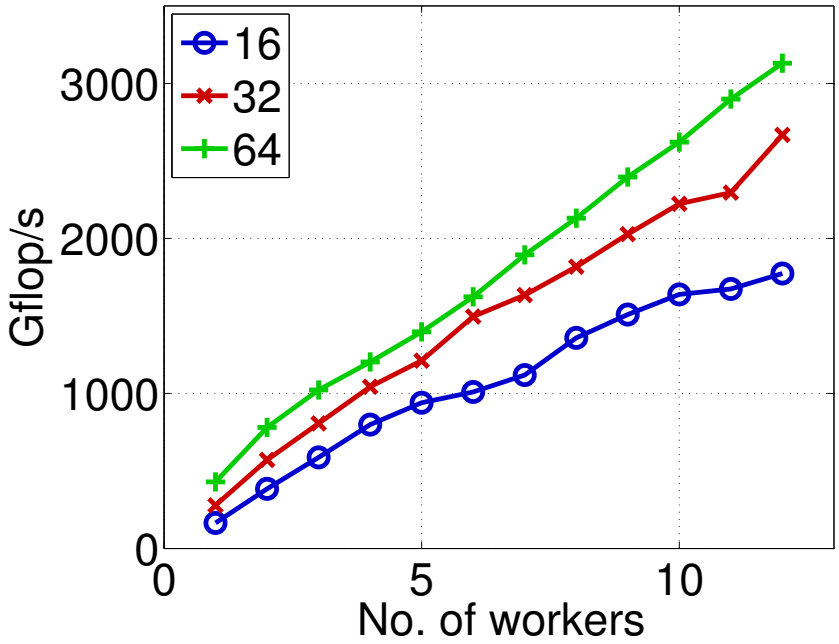


UPPSALA
UNIVERSITET

Weak scaling, banded matrices

Erik cluster, 16 CPU cores + 2 GPUs per worker (node)
Matrix dimension from 40000 on 1 node to 480000 on 12 nodes
Fixed bandwidth $2b + 1$, $b = 4000$





UPPSALA
UNIVERSITET

Thank you!

- Thanks to Elias Rudberg!

- Chunks and Tasks: `www.chunks-and-tasks.org`
  Parallel computing **40**, 328 (2014)
- The Ergo program: `www.ergoscf.org`

Support: The Swedish research council. The Göran Gustafsson foundation. The Lisa and Carl-Gustav Esseen foundation. The Swedish strategic e-science research program eSSENCE. Computational resources provided by SNIC at LUNARC and UPPMAX.