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
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)
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
Sparse matrix-matrix multiplication

**common approach**

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

Conventional parallelization (MPI)

Application programmer responsibilities

Conventional parallelization (MPI)

Distribute work
Distribute data
Send and receive messages
### MPI vs Chunks and Tasks

**Application programmer responsibilities**

<table>
<thead>
<tr>
<th>Conventional parallelization (MPI)</th>
<th>Chunks and Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribute work</td>
<td>Divide work into smaller parts</td>
</tr>
<tr>
<td>Distribute data</td>
<td>Divide data into smaller pieces</td>
</tr>
<tr>
<td>Send and receive messages</td>
<td>Register “tasks” and “chunks”</td>
</tr>
</tbody>
</table>
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
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**: \( ID = \text{registerChunk}(\text{myChunk}) \)
- **Task**: defined by input chunk types, output chunk type, work to be performed
- **Task registration**: \( ID = \text{registerTask}(\text{ID}, \text{ID}, ...) \)
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)*

⇒ No message passing

**Recursive nesting of tasks allowed**

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

⇒ 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

⇒ User does not have to provide data distribution
Design decisions

compared to related programming models

Identifiers to chunks provided by the library

*Unlike Linda and Concurrent Collections*

⇒ makes it feasible to make data available efficiently

⇒ leads to restrictions in how data can be accessed

Chunks are readonly once they are registered

*Unlike Linda but similar to Concurrent Collections*

⇒ Chunk cache coherence not an issue

⇒ Determinism (easier for the user to understand her code)
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)
A Chunks and Tasks matrix library

Sparse quad-tree representation
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 $32768 \times 32768$ matrices

![Bar chart showing wall time in seconds for different matrix multiplication setups.]

![Bar chart showing number of multiplications per node for different matrix multiplication setups.]
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 $2b + 1$, $b = 4000$
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.