Lithe: Enabling Efficient Composition of Parallel Libraries

Heidi Pan, Benjamin Hindman, Krste Asanović

xoxo@mit.edu {benh, krste}@eecs.berkeley.edu
Massachusetts Institute of Technology UC Berkeley

HotPar Berkeley, CA March 31, 2009
How to Build Parallel Apps?

Need both programmer productivity and performance!
Composability is Key to Productivity

Functional Composability

- **App 1**: sort
  - code reuse
  - same library implementation, different apps

- **App 2**: sort

- **App**: bubble sort, quick sort
  - modularity
  - same app, different library implementations

**Functional Composability**
Composability is Key to Productivity
Talk Roadmap

- **Problem**: Efficient parallel composability is hard!
- **Solution**:
  - Harts
  - Lithe
- **Evaluation**
Motivational Example

Sparse QR Factorization
(Tim Davis, Univ of Florida)

System Stack

Software Architecture

SPQR

TBB
MKL
OpenMP
OS
Hardware

Column Elimination Tree

Frontal Matrix Factorization
Out-of-the-Box Performance

Performance of SPQR on 16-core Machine

Time (sec)

Out-of-the-Box

Input Matrix

landmark

deltaX

ESOC

Rucci

sequential

0

0.5

1

1.5

2

2.5

3

3.5

0

5

10

15

20

25

80

75

70

65

60

85

1200

1000

800

600

400

0
Out-of-the-Box Libraries
Oversubscribe the Resources
MKL Quick Fix

Using Intel MKL with Threaded Applications

http://www.intel.com/support/performancetools/libraries/mkl/sb/CS-017177.htm

If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set `OMP_NUM_THREADS=1` in the environment.
Sequential MKL in SPQR
Performance of SPQR on 16-core Machine

- **Out-of-the-Box**
- **Sequential MKL**

**Input Matrix**

- **landmark**
- **deltaX**
- **ESOC**
- **Rucci**

**Time (sec)**

- 0
- 0.5
- 1
- 1.5
- 2
- 2.5
- 3
- 3.5
SPQR Wants to Use Parallel MKL

No task-level parallelism!

Want to exploit matrix-level parallelism.
Tim Davis manually tunes libraries to effectively partition the resources.
Performance of SPQR on 16-core Machine

- Out-of-the-Box
- Sequential MKL
- Manually Tuned

Input Matrix:
- landmark
- deltaX
- ESOC
- Rucci

Time (sec):
- landmark: 3.5, 3.0, 1.5, 0.5
- deltaX: 25, 20, 15, 10
- ESOC: 85, 80, 75, 70
- Rucci: 1200, 1000, 800, 600
Manual Tuning Cannot Share Resources Effectively

Give resources to OpenMP

Give resources to TBB
Manual Tuning Destroys Functional Composability

Tim Davis

OMP_NUM_THREADS = 4

MKL
OpenMP
Manual Tuning Destroys Performance Composability

SPQR

MKL v1
MKL v2
MKL v3

App

0 1 2 3
Talk Roadmap

- **Problem**: Efficient parallel composability is hard!
- **Solution**:
  - **Harts**: better resource abstraction
  - **Lithe**: framework for sharing resources
- Evaluation
Virtualized Threads are Bad

Different codes compete unproductively for resources.
Space-time partitions aren’t enough

What to do within an app?
Harts: Hardware Thread Contexts

- Represent real hw resources.
- Requested, not created.
- OS doesn’t manage harts for app.
Sharing Harts

Hart 0

Hart 1

Hart 2

Hart 3

time

TBB

OpenMP

OS

Hardware

Partition
Cooperative Hierarchical Schedulers

- Modular: Each piece of the app scheduled independently.
- Hierarchical: Caller gives resources to callee to execute on its behalf.
- Cooperative: Callee gives resources back to caller when done.
A Day in the Life of a Hart

TBB Sched: next?
execute TBB task

TBB Sched: next?
execute TBB task

TBB Sched: next?
nothing left to do, give hart back to parent

Cilk Sched: next?
don’t start new task, finish existing one first

Ct Sched: next?
Standard Lithe ABI

- **TBB\textsubscript{Lithe} Scheduler**
  - enter
  - yield
  - request
  - register
  - unregister

- **OpenMP\textsubscript{Lithe} Scheduler**

- **Caller**
  - call
  - return

- **Callee**
  - call
  - return

- Analogous to function call ABI for enabling interoperable codes.
- Mechanism for sharing harts, not policy.
Given the current scheduler, Lithe Runtime is illustrated with a scheduler hierarchy. This hierarchy comprises multiple levels, starting from the Hardware layer at the bottom, followed by the OS layer, and then the Lithe layer. Above Lithe, there are two layers: TBB\textsubscript{Lithe} and OpenMP\textsubscript{Lithe}.

- **TBB\textsubscript{Lithe}** has an enter, yield, request, register, and unregister feature.
- **OpenMP\textsubscript{Lithe}** has an enter, yield, request, register, and unregister feature as well.

The current scheduler is depicted with a blue square, and the harts are shown in green squares.
Register / Unregister

Register dynamically adds the new scheduler to the hierarchy.

```
matmult()
    register(OpenMP_Lithe);
    ...
    unregister(OpenMP_Lithe);
```

Request asks for more harts from the parent scheduler.

```
matmult(){
  register(OpenMP_Lithe);
  request(n);
  :
  unregister(OpenMP_Lithe);
}
```
Enter / Yield transfers additional harts between the parent and child.

```c
enter(OpenMP_Lithe);
...;
yield();
```
SPQR with Lithe
SPQR with Lithe

Diagram showing the integration of SPQR with Lithe, showcasing the use of TBB, MKL, and OpenMP libraries for parallel execution. The diagram illustrates the progression of time with `matmult`, `reg`, and `unreg` operations.
Talk Roadmap

- **Problem**: *Efficient* parallel composability is hard!
- **Solution**:
  - Harts
  - Lithe
- **Evaluation**
Implementation

- **Harts**: simulated using pinned Pthreads on x86-Linux
  ~600 lines of C & assembly

- **Lithe**: user-level library (register, unregister, request, enter, yield, ...)
  ~2000 lines of C, C++, assembly

- **TBB_{Lithe}**: 
  ~1500 / ~8000 relevant lines added/removed/modified

- **OpenMP_{Lithe} (GCC4.4)**: 
  ~1000 / ~6000 relevant lines added/removed/modified
All results on Linux 2.6.18, 8-core Intel Clovertown.

- **TBB\textsubscript{Lithe}** Performance (µbench included with release)

<table>
<thead>
<tr>
<th></th>
<th>tree sum</th>
<th>preorder</th>
<th>fibonacci</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBB\textsubscript{Lithe}</td>
<td>54.80ms</td>
<td>228.20ms</td>
<td>8.42ms</td>
</tr>
<tr>
<td>TBB</td>
<td>54.80ms</td>
<td>242.51ms</td>
<td>8.72ms</td>
</tr>
</tbody>
</table>

- **OpenMP\textsubscript{Lithe}** Performance (NAS parallel benchmarks)

<table>
<thead>
<tr>
<th></th>
<th>conjugate gradient (cg)</th>
<th>LU solver (lu)</th>
<th>multigrid (mg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP\textsubscript{Lithe}</td>
<td>57.06s</td>
<td>122.15s</td>
<td>9.23s</td>
</tr>
<tr>
<td>OpenMP</td>
<td>57.00s</td>
<td>123.68s</td>
<td>9.54s</td>
</tr>
</tbody>
</table>
Performance Characteristics of SPQR (Input = ESOC)
Performance Characteristics of SPQR (Input = ESOC)

Sequential TBB=1, OMP=1
172.1 sec
Performance Characteristics of SPQR (Input = ESOC)

Out-of-the-Box TBB=8, OMP=8
111.8 sec
## Performance Characteristics of SPQR (Input = ESOC)

<table>
<thead>
<tr>
<th>NUM_OMP_THREADS</th>
<th>Out-of-the-Box</th>
<th>Manually Tuned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>70.8 sec</td>
<td></td>
</tr>
</tbody>
</table>

3D diagram showing the comparison of `Out-of-the-Box` and `Manually Tuned` performance characteristics. The diagram includes time in seconds on the y-axis, and different thread counts on the x-axis and z-axis.
Performance of SPQR with Lithe

- **Input Matrix**
  - landmark
  - deltaX
  - ESOC
  - Rucci

- **Time (sec)**
  - Out-of-the-Box
  - Manually Tuned
  - Lithe
Future Work

SPQR

TBB\textsubscript{Lithe}

OpenMP\textsubscript{Lithe}

Ct\textsubscript{Lithe}

Cilk\textsubscript{Lithe}
Conclusion

- Composability essential for parallel programming to become widely adopted.
- Lithe project contributions
  - Harts: better resource model for parallel programming
  - Lithe: enables parallel codes to interoperate by standardizing the sharing of harts

- Parallel libraries need to share resources cooperatively.
We would like to thank George Necula and the rest of Berkeley Par Lab for their feedback on this work.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). This work has also been in part supported by a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors also acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.