Threading Building Block (TBB) by Intel

- C++ library
  - Represents higher-level task-based parallelism that abstracts platform details and threading from the user
  - Doesn’t require special language/compiler
  - Can be used on any processor with any OS
- Task scheduler implements task stealing for dynamic workload balancing
- Implements common parallel performance pattern such as parallel loops, flow graphs etc.

TBB

- Implements concurrent containers – scalable alternative to serial data containers such as C++ STL containers
- Provides a comprehensive set of synchronization primitives – mutexes, condition variables, atomic operations
- Provides the ability to specify priorities for tasks/task groups
- Implements parallel data structures such as concurrent priority queues, concurrent unordered sets, etc.

Grand Central Dispatch (GCD) by Apple

- Also implements task queues using a thread pool paradigm
  - Uses threads at the lower level
- Applications create tasks that can be expressed either as a function or as a “block” (akin to a closure)
- Mac OS X, Apache HTTP Server

GCD

- Application framework
  - Dispatch queues – queue of tasks; concurrent or serial. Queues with different priority levels are created by the library that can execute tasks concurrently
  - Dispatch source – allow client to register blocks/functions to execute asynchronously on system events or POSIX signal
  - Dispatch groups – allow several tasks to be grouped for later joining
  - Dispatch semaphores – allow a client to permit a certain number of tasks to execute concurrently
Lithe: Composing Parallel Software Efficiently

Pan, Hindman, Asanovic
MIT/Berkeley
PLDI 2010

Software Trends

• Parallel applications using parallel libraries in turn using other parallel libraries
• E.g. SuiteSparseQR
  • Incorporated into MATLAB
  • Uses Intel's TBB
  • Calls BLAS libraries
    – Which use OpenMP
• State of the art
  – User required to make decisions on degree of parallelism at each level
  – Thread virtualization implies user has no control over influencing resource allocation

Possible Solutions

• Require all parallelism to be expressed using a universal high-level abstraction
• Provide a low-level substrate for parallel libraries to interface with

Lithe

• Cooperative hierarchical resource management
  – Hardware threads
  – Hierarchy of schedulers
  – Cooperative request/release/allocation of threads up and down the hierarchy
Lithe Implementation

- Harts: hardware threads, one per physical core/hardware context
- Execution contexts

Conventional System Vs Lithe

Lithe Interface

- Scheduler -> implements Lithe callback interface for a library
- Runtime -> invokes the appropriate library’s callback

Image source: Pan et al., PLDI 2010

Image creator: Shantonu Hossain
**Example**

Figure 3. An example of how multiple harts might flow between a parent and child scheduler.

*Image source: Pan et al., PLDI 2010*

**Parallel Quicksort Example**

```
void sort(vector v) {
    par_sort(sched);  // Schedule partition
    lithe_enter();    // Enter Lithe
    par_sort_enter(); // Enter par_sort
    lithe_yield();    // Wait
    ...}

void par_sort(vector v, queue q) {
    if (v.size() < 1000) 
        sort(v);        // Small enough, do it ourselves
    else
        split();        // Split the vector
    lithe_enter();    // Enter Lithe
    par_sort_enter(); // Enter par_sort
    lithe_yield();    // Wait
    ...}
```

*Image source: Pan et al., PLDI 2010*

**Lithe Runtime Functions and Scheduler Callbacks**

<table>
<thead>
<tr>
<th>Lithe Runtime Interface</th>
<th>Scheduler Callback Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>sched_register(sched)</td>
<td>register(child)</td>
</tr>
<tr>
<td>sched_unregister(sched)</td>
<td>unregister(child)</td>
</tr>
<tr>
<td>sched_request(nharts)</td>
<td>request(child, nharts)</td>
</tr>
<tr>
<td>sched_enter(child)</td>
<td>enter(child)</td>
</tr>
<tr>
<td>sched_yield()</td>
<td>yield(child)</td>
</tr>
<tr>
<td>sched_resume()</td>
<td>enter(child)</td>
</tr>
<tr>
<td>ctx_init(ctx, stack)</td>
<td>N/A</td>
</tr>
<tr>
<td>ctx_fini(ctx)</td>
<td>N/A</td>
</tr>
<tr>
<td>ctx_resume(ctx)</td>
<td>N/A</td>
</tr>
<tr>
<td>ctx_block(ctx)</td>
<td>block(ctx)</td>
</tr>
<tr>
<td>ctx_unblock(ctx)</td>
<td>unblock(ctx)</td>
</tr>
</tbody>
</table>

*Image source: Pan et al., PLDI 2010*

**Lithe – SPMD Scheduler**

```
void spmd_schedule actionTypes
    void (vector v, queue q)
    spmd_schedule(sched);  // Schedule partition
    par_sort(v, sched);     // Schedule par_sort
    lithe_register();       // Enter Lithe
    par_sort_enter();       // Enter par_sort
    lithe_yield();          // Wait
    ...}
```

*Image source: Pan et al., PLDI 2010*