On the Impact of OpenMP Task Granularity

Thierry Gautier, Christian Perez, Jérôme Richard

Most of this work has been published at IWOMP 2018, Barcelona, Spain.
Gysela, 2D Advection, and COMET

Simulate the plasma of tokamaks
Developed by CEA, since 2000
~80,000 lines of FORTRAN

Poisson solver
Compute the electromagnetic field

Vlasov solver
Move particles

1D advections
Transposition
2D advections

Displacement fields comp.
Interpolation

Section

1D advections

Displ-TaskImpl

Spline-TskImpl

IoManager

Constants
From COMET to OpenMP Code: Stencil-Stencil Use Case

Runtime execution

==

OpenMP program

Metatask

Implementation

OpenMP dependent task

Repartition task
Performance of ST-ST: Gant Chart

**Completion time for 20 iterations**

<table>
<thead>
<tr>
<th>bloc size</th>
<th>1024x512</th>
<th>1024x256</th>
<th>1024x128</th>
<th>1024x64</th>
<th>1024x32</th>
<th>1024x16</th>
<th>1024x8</th>
</tr>
</thead>
<tbody>
<tr>
<td>#tasks per it</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
<td>2048</td>
<td>4096</td>
<td>8192</td>
</tr>
<tr>
<td>time (s)</td>
<td>1.20</td>
<td>1.19</td>
<td>1.05</td>
<td><strong>1.01</strong></td>
<td>1.19</td>
<td>1.21</td>
<td>1.42</td>
</tr>
</tbody>
</table>

24 cores, bloc size 1024x64 ➔ 1024 tasks

1) **Seems a very long submission time. Is it normal?**
   - but short task duration (~ 1.1µs)
   - about 42968 cycles of the machine / tasks

2) **No cache reuse:**
   producer task metatask Mt1 does not activate consumer Mt2 because it is not yet submitted!

3) Possible cache reuse whenMt1 activates consumer Mt2, with high probability that a running thread executes both dependent tasks consecutively, but no guarantee (in OpenMP)

**Can performance be improved?**
Dependent Task Management Overheads (LLVM)

Dependent task management ~ 10 times cost of independent task management

- Dependent version (Jacobi Dep, sparselu Dep) versus independent version (Jacobi, sparselu)
- Checking dependencies

Costly operation → task throttling to limit creation overhead
LLVM Overhead in Checking Deps

Checking dependencies in LLVM

- Retrieving the last created task that write data
- Hash table, key = memory address

We have been faced with a classical performance problem

- Too small static size for hash table (997)
  - Why this constant?
- Many collisions => search operation becomes linear with number of visited dependencies
2D Gysela Advection

Generation of task follows
- for each component
  - for each plane

Problem: few cache reuse
- breath first creation order
- work-stealing policy is non cache constructive
One Line Rescheduling

During task insertion
- Keep track of the range of tasks in the ready list on which it depends

Little overhead with respect to current code for checking dependencies
- Complexity $O(1)$ with respect to default depend task implementation

Change of work stealing implementation
- Constructive cache sharing version where both thief/victim access to same endpoint of the dequeue
Final 2D Gysela Advection Results

Up to 40% gain with respect to standard LLVM runtime

- Larger hash table: always interesting if many dependencies
- Reordering gain when the core count is high

with:
- hash table means hash table size=132069 in place of 997
- reorder= rescheduling with big hash table

- LLVM runtime, branch release_5.0
- Broadwell socket 24 cores@2.2Ghz
# Bilan

**libKOMP: research in productive runtime (LLVM based)**

- Extensions: e.g. task/data affinity, scheduling heuristic, …
- OMPT module for tracing execution
- [http://gitlab.inria.fr/openmp/libkomp](http://gitlab.inria.fr/openmp/libkomp)

<table>
<thead>
<tr>
<th>Limitation</th>
<th>Type</th>
<th>Solution</th>
<th>Solution in libKOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>task creation</strong></td>
<td>size of internal descriptor</td>
<td>optimization</td>
<td>reduce size of task descriptor</td>
</tr>
<tr>
<td>overhead</td>
<td></td>
<td>parallelization</td>
<td>waiting for weak dependencies implementation in LLVM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>work first principle to checks deps on steal operation [15]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>better algorithm [21]</td>
<td>dynamic resizing of hash table, Purpose of OpenMP-5.0 depobj ?</td>
</tr>
<tr>
<td><strong>scheduling</strong></td>
<td>task throttling</td>
<td>better heuristics [12] could be disable</td>
<td>suppress heuristic</td>
</tr>
<tr>
<td></td>
<td>plugin algorithm</td>
<td>OpenMP extension + expert function</td>
<td>plug in infrastructure for new scheduling algorithm</td>
</tr>
</tbody>
</table>
Conclusion

Task implementation (in LLVM-) OpenMP runtime has limitations

- Overhead would be reduced in the future
- Several algorithmic decisions are hard coded
  - e.g. task throttling = dequeue size in LLVM (for libGOMP: \#task <= 64 \#threads)
  - e.g. hash table size = 997 in LLVM,
- Scheduling algorithms cannot be plug in the runtime

Perspectives

- Future nested tasks + weak dependencies may reduce time in task management
  - if implementation support high degree of concurrency with many fine grain tasks
- Interaction with the way component assembly generates task parallelism
  - impact in our compiler and may be into the component model is not well define

Open questions

- Should OpenMP specify basic task scheduling constraints?
  - a task creation is always a non blocking operation, except if explicit ‘if’ clause is false
- Universal scheduling algorithm does not exist. How to specialize it in OpenMP?