Enhancing Productivity on Heterogeneous Supercomputers with Task-based Programming Model

Adrien Roussel, Mickael Boichot, Manuel Ferat, Romain Pereira
CEA, DAM, DIF

SIAM CSE, Amsterdam
February 28, 2023
Introduction

- Most of the time, scientific applications are already parallelized for CPU (or partly for GPU).

- To take advantage of GPUs, developers have to rewrite their codes (in part? in whole?).

- Goals:
  - Build a preliminary analysis to determine the potential of the application for a GPU port.
  - Port mini-application to heterogeneous architectures with this information.
Contributions

Goal: Inform the user

- that there exists a test case of the application suitable for a GPU port
- about the strengths and weaknesses of the application for a GPU port

- We analyzed the variations of two metrics:
  - the degree of parallelism (→ saturate GPU cores)
  - and the working set (→ bounded GPU memory)

- Porting app. To GPU Computing
  - Using Task based prog. Model
    - MPI+OpenMP (tasks + target)

```c
int main(...)
{
    /* parallel CPU code */
}
```

Par
tlel
compu
tations

working set

projection

Relate the two metrics

Extrapolate application behavior

GPU Porting

- varying dataset input size

- Relate the two metrics
- Extrapolate application behavior
- GPU Porting
Contents

1. Introduction

2. Application Characterization Method

3. Proxy Applications Analysis

4. Porting Proxy App to GPU: LULESH

5. Conclusion
2. Application Characterization Method
Targeted GPU Characterization

Maximizing FFMA throughput ("floating point fused multiple add")

\[ \text{RP} = \#\text{cores} \times L \]

where

- L: 32 bits FFMA instruction latency

Characterization

- #cores
- L2 Cache
- DRAM

Required Parallelism (RP)

Already known (GPU documentation)
Metrics to Characterize the Application

Applications analyzed at Loop Granularity
(already parallelized with OpenMP)

L: for (i = 0; i < N; ++i)
compute;

Set of data necessary to run the loop

Working Set

# independent Iterations
(i.e., tripcount)

Parallelism Degree P(L)

measured at two Granularities,
i.e., the size of smallest accessed
memory element (using DynamoRIO)

WS_{malloc}(L)  WS_{cache line}(L)
Extrapolation Methodology

\[ \forall L \in S, \ n \in \{n_1, n_2, \ldots\} \rightarrow \text{measure} \]
- working set \( WS(L) \)
- parallelism degree \( P(L) \)

\[ P(L) = \mathit{g}(n) \]

For each loop

<table>
<thead>
<tr>
<th>loop</th>
<th>working set</th>
<th>parallelism degree</th>
<th>input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>( WS(L) )</td>
<td>( P(L) )</td>
<td>( n_1 )</td>
</tr>
<tr>
<td>L</td>
<td>( WS(L) )</td>
<td>( P(L) )</td>
<td>( n_2 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Extrapolation Methodology

\[ \forall L \in S, n \in \{n_1, n_2, \ldots\} \rightarrow \text{measure} \]
- working set \( WS(L) \)
- parallelism degree \( P(L) \)

\[ WS(L) = f(P(L)) \]
\[ P(L) = g(n) \]

Test cases fitting target GPU memory and exposing enough parallelism
3. Proxy Applications Analysis

LULESH, miniFE
Experimental Protocol

- **Evaluation Machine**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>Intel Xeon E5-2680</td>
</tr>
<tr>
<td>#sockets</td>
<td>2</td>
</tr>
<tr>
<td>#cores/socket</td>
<td>8</td>
</tr>
<tr>
<td>RAM (GB)</td>
<td>128</td>
</tr>
<tr>
<td>OS</td>
<td>Linux 3.10</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel 20.0.0</td>
</tr>
</tbody>
</table>

- **Evaluated mini-applications**

  - LULESH (hydrodynamic simulation)
  - MiniFE (finite element problem)
  - Quicksilver (Monte Carlo particle transport)
  - XSBench (Monte Carlo particle transport kernel)

  All these applications have CPU OpenMP, GPU CUDA and HIP versions.

- **Considered targeted GPUs**

<table>
<thead>
<tr>
<th>GPU</th>
<th>RAM</th>
<th>#cores</th>
<th>minimal required parallelism RP</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100</td>
<td>16 GB</td>
<td>5120</td>
<td>20480</td>
</tr>
<tr>
<td>A100 40GB</td>
<td>40 GB</td>
<td>6912</td>
<td>27648</td>
</tr>
<tr>
<td>A100 80GB</td>
<td>80 GB</td>
<td>6912</td>
<td>27648</td>
</tr>
<tr>
<td>MI100</td>
<td>32 GB</td>
<td>7680</td>
<td>30720</td>
</tr>
</tbody>
</table>
LULESH

Projection of dataset input size $s_{\text{max}}$ saturating GPU memory and corresponding $\rho$ ratio ($\rho(L) = P(L) / \text{RP}$)

- Sufficient exposed parallelism ($\rho \gg 1$)
- Limiting factor: memory size
- Advised data transfers granularity: malloc

- Similar conclusions for miniFE

<table>
<thead>
<tr>
<th>target GPU</th>
<th>$s_{\text{max}}$</th>
<th>$\rho$ (loop 1773)</th>
<th>$\rho$ (other loops)</th>
<th>$s_{\text{max}}$</th>
<th>$\rho$ (loop 1773)</th>
<th>$\rho$ (other loops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V100 16 GB</td>
<td>285</td>
<td>19.2</td>
<td>1130.3</td>
<td>289</td>
<td>19.4</td>
<td>1178.6</td>
</tr>
<tr>
<td>A100 40 GB</td>
<td>387</td>
<td>19.3</td>
<td>2099.4</td>
<td>393</td>
<td>19.6</td>
<td>2195.4</td>
</tr>
<tr>
<td>A100 80 GB</td>
<td>487</td>
<td>24.3</td>
<td>4198.7</td>
<td>495</td>
<td>24.7</td>
<td>4386.8</td>
</tr>
<tr>
<td>MI100 32 GB</td>
<td>359</td>
<td>16.1</td>
<td>1511.5</td>
<td>365</td>
<td>16.4</td>
<td>1586.6</td>
</tr>
</tbody>
</table>
4. Porting Proxy App to GPU: LULESH
Porting LULESH to MPI+OpenMP(tasks)

- **LULESH**
  - Hydrodynamic problems solver - **Iterative** simulation
  - Original code express **loop-based parallelism** (parallel-for)

- **Porting to MPI+OpenMP (tasks)**
  - Loops splitted into tasks with **dependencies**
  - Single producer/multi-consumer scheme
  - MPI communications nested within OpenMP tasks
    - Overlap communications by computations

```c
for (int j = 0; j < n_nodes; j += BS) {
  # pragma omp task depend(out : mesh->nodes[j])
  {
    for (int i = j; i < j + BS; ++i)
      work_on_nodes(mesh->nodes[i]);
  }
}

for (int j = 0; j < n_cells; j += BS) {
  # pragma omp task depend(in : [...])
  {
    for (int i = j; i < j + BS; ++i)
      work_on_cells(mesh->cells[i]);
  }
}
```

Adrien Roussel - SIAM CSE

Feb. 28, 2023
Concept Generalization for Heterogeneous Architectures with OpenMP Target

- Porting to MPI+OpenMP (tasks + GPU target)
  - GPU « super-tasks » merging several CPU tasks
  - From CPU tasks to GPU offloading while enhancing asynchronism
    - Sending/Retrieving data to/from GPU with ‘update’ clause
    - Executing GPU kernels through ‘nowait’ clause
  - Offloading 6 loops given by application characterization
    - IntegrateStressForElems, CalcKinematicsForElems, CalcAccelerationForNodes, CalcFBHourglassForceForElems1, CalcHourglassControlForElems, CalcMonotonicQGradientsForElems
- Use of cudaMallocHost for the GPU buffers
  - Pinned memory to enable asynchronous execution

```c
1 buffer = cudaMallocHost(…); 
[…]
2 for (int j = 0 ; j < n_nodes; j += BS) 
3 { 
4     # pragma omp task depend(out : mesh->nodes[j]) 
5     { 
6         for (int i = j ; i < j + BS ; ++i) 
7             work_on_nodes(mesh->nodes[i], buffer[i]); 
8     }
9 }
```

Adrien Roussel - SIAM CSE
## Experimental Protocol

<table>
<thead>
<tr>
<th>Processor</th>
<th>AMD EPYC 7H12</th>
</tr>
</thead>
<tbody>
<tr>
<td>#sockets</td>
<td>2</td>
</tr>
<tr>
<td>#cores/socket</td>
<td>64</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA A100</td>
</tr>
<tr>
<td>#GPU/socket</td>
<td>2</td>
</tr>
<tr>
<td>RAM (GB)</td>
<td>256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC-OpenMP (No Offloaded part)</td>
<td>4.1.0</td>
</tr>
<tr>
<td>LLVM (Offloaded part)</td>
<td>16.x</td>
</tr>
<tr>
<td>Open MPI</td>
<td>4.0.5</td>
</tr>
</tbody>
</table>
LULESH Performance Evaluation

- Problem size
  - 264³ elements
    - 80% of the NUMA domain memory capacity
    - 20% of the GPU memory capacity
  - 128 iterations

- Placement
  - Per MPI Process:
    - 16 threads (on 16 cores)
    - + 1 GPU

- Weak scaling
  - Median runtime of 10 runs
  - Each version scale very well
  - OpenMP tasks version significantly improves performances due to cache reuse
  - Significant performance gain with GPU offloading

<table>
<thead>
<tr>
<th>ranks</th>
<th>parallel-for</th>
<th>no-offloading</th>
<th>offloading</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>565.84</td>
<td>325.02</td>
<td>209.31</td>
</tr>
<tr>
<td>8</td>
<td>572.45</td>
<td>331.09</td>
<td>209.43</td>
</tr>
<tr>
<td>27</td>
<td>581.26</td>
<td>333.98</td>
<td>212.78</td>
</tr>
</tbody>
</table>
5. Conclusion
Conclusion

- An **approach** : (int main(...){...
} ) → { test cases such that

  int main(...){...
  }

fit on RAM and do not lack parallelism

- **complementary** to existing work (analysis of several application runs → extrapolation)

- We have **evaluated** our approach and shown that we can **qualitatively guide developer** in their GPU porting

- We have **ported** LULESH on heterogeneous machine with a **task-based programming model**
  - Significant gains are observed
Thank you!

Questions?