OpenACC – Case Study on Productivity & Performance

1st Experiences with Real-World Applications

Sandra Wienke
Center for Computing and Communication (RZ)
RWTH Aachen University

Agenda

- Introduction
- Real-world Applications
- Performance
- Productivity
- Conclusion & Outlook
Nowadays: low-level GPU APIs (like CUDA, OpenCL)

→ Unproductive development process

Directive-based programming model delegates responsibility for low-level GPU programming tasks to compilers

→ Data movement

→ Kernel execution

→ “Awareness” of particular GPU type

→ …

→ OpenACC
Experiences with 2 real-world software packages +

- OpenACC (Cray)
- PGI Accelerator
- OpenCL

Performance

- Runtime (including data transfers)
- OpenACC-compiler-dependent

Programmability, productivity

- Modified lines of source code

Ratio of development effort to performance
Real-World Applications: *KegelSpan*

- 3D simulation of bevel gear cutting process
- Compute key values (i.a. chip thickness) to analyze tool load and tool wear
- Fortran code (chip thickness computation)
  - Loop nest
  - Dependencies in inner loop

**Implementation**

- **basic**: outer loop in parallel on threads, inner loop serially
  - parallel, gang vector, vector_length data clauses
- **restruct**: basic + optimized data access pattern
  - adjusting of data clauses
- **locMem**: storing input/intermediate data in shared memory (OpenCL only)

---

Real-World Applications: *NINA*

- **Software**: for the solution of *Neuro-magnetic INverse lArge-scale problems*
- **Matlab code** (main program), **C code** (objective function; 1\textsuperscript{st}- and 2\textsuperscript{nd}-order derivatives)
  - Matrix-vector multiplications, vector operations

**Implementation**

- **basic**: outer loop in parallel on threads, inner loop serially
  - several parallel regions, store intermediate values in array + globally synchronize + loop interchange to avoid race conditions, sum reductions
- **l2par**: outer loop to thread-blocks, inner loop to threads
  - reductions on inner loop (not possible with PGI Accelerator)
- **blocked**: blocked matrix-vector multiplication
  (OpenCL: + async data/ kernel, pinned memory, fixed loop trip count)
- Loop unrolling important (pragma vs. manual unrolling)

---

# Performance: Setup

<table>
<thead>
<tr>
<th>OpenACC</th>
<th>PGI Accelerator/ OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NVIDIA Tesla C2050 (Fermi)</strong></td>
<td><strong>PGI Accelerator/ OpenCL</strong></td>
</tr>
<tr>
<td>ECC on</td>
<td>ECC on</td>
</tr>
<tr>
<td>CUDA toolkit 4.0</td>
<td>CUDA toolkit 4.0</td>
</tr>
<tr>
<td><strong>AMD Magny-Cours</strong></td>
<td>Intel Westmere</td>
</tr>
<tr>
<td>12 cores</td>
<td>4 cores</td>
</tr>
<tr>
<td>(Cray experimental system)</td>
<td></td>
</tr>
<tr>
<td><strong>SUSE Enterprise Server 11</strong></td>
<td>Scientific Linux 6.1</td>
</tr>
<tr>
<td><strong>Cray Compiler 8.1.0</strong></td>
<td>PGI Compiler 12.3/ Intel Compiler 12.1.2</td>
</tr>
</tbody>
</table>

---

OpenACC – Case Study on Productivity & Performance

Sandra Wienke | Rechen- und Kommunikationszentrum
Performance: *KegelSpan*

![Graph showing performance comparison between single and double precision for OpenCL and PGIAcc versions.](image)

Performance: \textit{NINA}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{performance_chart.png}
\caption{Comparison of runtime between different versions and implementations.}
\end{figure}

## Productivity

### KegelSpan

<table>
<thead>
<tr>
<th></th>
<th>OpenCL</th>
<th>PGI Acc</th>
<th>OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>106/35</td>
<td>0/3</td>
<td>0/3</td>
</tr>
<tr>
<td>restruct</td>
<td>183/39</td>
<td>84/14</td>
<td>84/14</td>
</tr>
<tr>
<td>locMem</td>
<td>183/58</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Total modified lines of code: 241
- Serial version: ~150 kernel code lines

### NINA

<table>
<thead>
<tr>
<th></th>
<th>OpenCL</th>
<th>PGI Acc</th>
<th>OpenACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>-</td>
<td>9/121</td>
<td>9/31</td>
</tr>
<tr>
<td>l2par</td>
<td>-</td>
<td>-</td>
<td>9/37</td>
</tr>
<tr>
<td>blocked</td>
<td>330/300</td>
<td>12/109</td>
<td>12/85</td>
</tr>
</tbody>
</table>

- Total modified lines of code: 630
- Serial version: ~100 kernel code lines

---

OpenACC – Case Study on Productivity & Performance
Sandra Wienke | Rechen- und Kommunikationszentrum
Conclusion

- **KegelSpan** (moderately-complex kernel)
  - OpenACC ≈ 80% of OpenCL (DP) perf
  - Matches expectation
  - 98 vs. 241 modified code lines

- **NINA** (complex kernel)
  - OpenACC ≈ 40% of OpenCL (DP) perf
  - Probable reason: lack of support of local memory + manual global sync
  - 46 vs. 630 modified code lines

⇒ Promising ratio of effort to performance

⇒ 1.3x slower, but 2.5x fewer modified code lines

⇒ 2.6x slower, but 13.7x fewer modified code lines
Recent/ further performance & programmability investigations

→ More OpenACC features: kernels, cache, async

→ Other OpenACC compilers (Nvidia GPUs): PGI

→ Other target hardware architectures: MIC, AMD

Full porting of NINA software

→ Interaction with Matlab possible?

Our opinion on OpenACC

→ Essential for further growth and acceptance of accelerators

→ Important step towards (OpenMP-) standardization (*OpenMP for Accelerators*)
German Heterogeneous Computing Group (GHCG)

→ Independent interest group with respect to high performance computing using accelerators in the German-speaking zone

→ Aim: Intensify the technical and scientific exchange of ideas on projects, hardware and algorithms

Take part!

→ www.ghc-group.org

→ Usergroup Meetings

→ Everyone is welcome