Hybrid implementation of the Vegas Monte-Carlo algorithm

G. Grasseau, S. Lisniak and D. Chamont
Leprince-Ringuet Laboratory (LLR),
Ecole polytechnique, Palaiseau
France
Motivations

- Needs from the LLR CMS team
- Cutting edge analyses techniques often imply *multi-dimensional integrations* that are computing intensive. In addition, the samples to analyze are huge ($10^6$ events).
- The reconstruction of the mass of a tau-lepton pair (SVFit) requires $\sim 1s$/event while the typical time of the rest of the analysis chain is $1ms$.

- matrix-element methods (MEM) can reach $60s$/event
- The High Luminosity will increase again the analysis time
- For an analysis team, the situation is difficult if the processing of all samples exceeds 2-3 weeks (elapsed time)
- We focus on the MC integration which is common numerical bottleneck within the LLR CMS group analyses

$\rightarrow$ Provide a powerful implementation for Monte Carlo integrations for data analysis

$\int_V d^n x \ f(x)$
VEGAS MC Integration

- The ROOT-based MC integration environment is popular with the CMS collaboration
- ROOT MC implementation is a GSL one
- GSL contains 3 MC algorithm: classical, miser, VEGAS

Reducing drastically the restitution time:
- Hardware: aggregate the computing power of nodes and accelerators (GPU, Xeon PhI, ...)
- Software: hybrid model using MPI and OpenCL standards to glue the HW

The purpose of this talk is to present all the machinery to compute fast MC integral
VEGAS - Principles


- **MC, $M$ points $x \in V$**
  
  $I \rightarrow \frac{V}{M} \langle f(x) \rangle_p$

  with $p(x)$ probability density

- **Importance sampling**
  
  $p(x) \propto |f(x)|$

  $\rightarrow$ minimize $\sigma^2$

- **Stratified sampling**
  
  $V = \bigcup V_i$ with $i = 1, \ldots, N$

  $\sigma_i^2 = \frac{\sigma^2}{N}$

  $\rightarrow$ concentrate sampling for high $\sigma^2$ i.e. $p(x) \propto |df(x)/dx|$
Parallelism considerations

- Tried to use the APUs as CPUs → run all the program (like MPI, OpenMP)
- Easy to do, avoid splitting the program
- Avoid starting kernel latencies
- Optimize the computing load
- Turned out to be not possible
  - Limit of the work size of 1024 → performance on NVidia - Intel OK
  - No simultaneous Kernels with NVidia OpenCL
  - Not possible to synchronize WorkGroups (1) - same in CUDA

- Simplified numerical scheme
  - Loop <until convergence>
    - Loop <internal>
      - Loop <N points x>
        - x = rand()
        - f(x) = F(x)
        - update <f>, var, p(x)
      - Update l, $\sigma^2$, $\chi^2$
    - End Loop
  - End Loop

- Constrained to split !!!

(1) thread blocks in CUDA
Node architecture

- At the node level (OpenCL)
- The platforms are feed by an Event Dispatcher
- Each Event to process (MC) is sent to a Device
- When a Device is idle a new event is sent to be processed

OpenCL:
- asynchronous processing
- write, read, running kernels
- clEvent

GPU in HEP, Pisa, September 2014
Benchmark description (1)

GridCL Platform

Funded by the P2IO (french) project
Benchmark description (2)

Function to integrate

\[ I = \int_0^{2\pi} \prod_{i} \frac{\sin(x_i)}{x_i} \, d^n x \]

- Special function known as sinus integral, \( n=5 \)
- Number of points \( \text{cst} = 5 \times 5 \times 0.5 \times 10^6 \)
- Only 12 boxes in each direction → 2 points box
- Preliminary performance study → global landscape

Compilation
- mpicc / icc -O3
- icc v-13.0.1
- OpenMPI 1.6.5

OpenCL
- Intel 1.2
- NVidia 1.1

GG, SL, DC

GPU in HEP, Pisa, September 2014
## Single node performances (1)

<table>
<thead>
<tr>
<th></th>
<th>GSL</th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL + 2xMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>APUs</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Time/Ev (s)</td>
<td>11.0</td>
<td>0.60</td>
<td>0.22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed up</td>
<td>1</td>
<td>18.5</td>
<td>50.0</td>
<td>56.0</td>
<td>56.0</td>
<td>94.5 110</td>
</tr>
</tbody>
</table>

**Platform**
- 2 x Intel Xeon E5-2650
- 2 x NVidia Kepler K20M

---

**Good news**
- CPUs: benefit of // and vectorization

**... but suboptimal**
## Single node performances (1)

<table>
<thead>
<tr>
<th>Platform</th>
<th>2 x Intel Xeon E5-2650</th>
<th>2 x NVidia Kepler K20M</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th></th>
<th>GSL</th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL + 2xMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPUs</strong></td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td><strong>APUs</strong></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Time/Ev (s)</strong></td>
<td>11.0</td>
<td>0.60</td>
<td>0.22</td>
<td>0.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Speed up</strong></td>
<td>1</td>
<td>18.5</td>
<td>50.0</td>
<td>56.0</td>
<td>56.0</td>
<td>94.5 110</td>
</tr>
</tbody>
</table>

### Good news
- CPUs: benefit of // and vectorization

### ... but suboptimal
- GPGPUs: not enough work for the kernel (cost of the latencies > 25 %, because of the splitting ...)

---

GG, SL, DC

GPU in HEP, Pisa, September 2014
## Single node performances (1)

<table>
<thead>
<tr>
<th></th>
<th>GSL</th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL + 2xMPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APU</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time/Ev (s)</td>
<td>11.0</td>
<td>0.60</td>
<td>0.22</td>
<td>0.20</td>
<td>0.20</td>
<td>0.15</td>
<td>0.10</td>
</tr>
<tr>
<td>Speed up</td>
<td>1</td>
<td>18.5</td>
<td>50.0</td>
<td>56.0</td>
<td>56.0</td>
<td>94.5</td>
<td>110</td>
</tr>
</tbody>
</table>

### Platform
- 2 x Intel Xeon E5-2650
- 2 x NVidia Kepler K20M

### Good news
- CPUs: benefit of // and vectorization
- Good behavior of the OCL Event Dispatcher

### ... but suboptimal
- GPGPUs: not enough work for the kernel (cost of the latencies > 25 %, because of the splitting ...)
- Nvidia OCL: doesn't allow to feed async. 2 cards → Thanks to MPI

GG, SL, DC
### Single node performances (2)

<table>
<thead>
<tr>
<th></th>
<th>GSL</th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL + 2xMPI</th>
<th>OCL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPUs</strong></td>
<td>1</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>1</td>
<td>All</td>
</tr>
<tr>
<td><strong>APUs</strong></td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Time/Ev (s)</strong></td>
<td>11.0</td>
<td>0.60</td>
<td>0.22</td>
<td>1.08</td>
<td>0.55</td>
<td>0.16</td>
</tr>
<tr>
<td><strong>Speed up</strong></td>
<td>1</td>
<td>18.5</td>
<td>50.0</td>
<td>10.1</td>
<td>19.9</td>
<td>70.3</td>
</tr>
</tbody>
</table>

**Platform**
- **2 x Intel Xeon E5-2650**
- **2 x Intel Xeon Phi 5110P**

---

**Good news**
- Same remarks as for K20M platform
- Nice Intel OpenCL implementation: use simultaneously of two cards

**... but suboptimal**
- Same remarks as for K20M platform
- Xeon Phis: very bad performance ... splitting. VTune to study the issues
## Single node performances (3)

<table>
<thead>
<tr>
<th></th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL</th>
<th>OCL + 6xMPI</th>
<th>OCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>All (1)</td>
<td>All</td>
<td></td>
<td></td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>APUs</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Time/Ev (s)</td>
<td>8.55</td>
<td>0.21</td>
<td>0.15</td>
<td>0.15</td>
<td>0.026</td>
<td>0.089</td>
</tr>
<tr>
<td>Speed up</td>
<td>1</td>
<td>39.1</td>
<td>56.0</td>
<td>55.6</td>
<td>328</td>
<td>95.3</td>
</tr>
</tbody>
</table>

### Platform
- **2 x Intel Xeon E5-2650**
- **6 x NVidia Titan**

**Good news**
- Same remarks as for K20M platform
- Clock of the processor change

**... but suboptimal**
- Same remarks as for K20M platform
Multi nodes architecture

- Nodes aggregation with MPI
## Multi-nodes performances

<table>
<thead>
<tr>
<th># Nodes</th>
<th>GSL</th>
<th>OCL</th>
<th>OCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>1</td>
<td>All (1)</td>
<td>All</td>
</tr>
<tr>
<td>APUs</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Time/Ev (s)</td>
<td>1</td>
<td>11.0</td>
<td>0.22</td>
</tr>
<tr>
<td>Speed up</td>
<td>1</td>
<td>1</td>
<td>50.0</td>
</tr>
<tr>
<td>Time/Ev (s)</td>
<td>2</td>
<td>11.0</td>
<td>0.127</td>
</tr>
<tr>
<td>Speed up</td>
<td>2</td>
<td>1</td>
<td>85.99</td>
</tr>
<tr>
<td>Speed up</td>
<td>1/2</td>
<td>1</td>
<td>1.72</td>
</tr>
</tbody>
</table>

### Platforms
- 2 x Intel Xeon E5-2650
- 2 x NVidia Kepler K20M

1 node

2 nodes

← global speed-up
Node architecture for high dimensional integral

- For large Integrals with a high dimensionality
- Dispatch sub-domains of the integral
- If needed integration domain can be spread on several nodes thanks to MPI
Conclusion

- Global map of the application. Identified the bottlenecks.

- If we take into account
  - Improving kernel x 2 (workload)
  - Running 2 MPI per node
  - Using 2xK20 nodes (0.9 dispatcher efficiency)

- We target a reasonable global speed up of 450 on K20s, 600 for Titan node

- Remaining work (months):
  - The functions to integrate (LHAPDF, …) written in Fortran, C++
  - 80 % translated in C99 (kernels)
  - CMSSW integration
  - Other method: MCMC
Thanks

- To P2IO funds
- To the organizers of this meeting
- For your attention