Characterizing Optimizations To Memory Access Patterns Using Architecture Independent Program Features

PRESENTER: ADITYA CHILUKURI[1]
CO-AUTHORS: JOSH MILTHORPE[1], BEAU JOHNSTON[2,1]
[1] AUSTRALIAN NATIONAL UNIVERSITY
[2] OAK RIDGE NATIONAL LABORATORY
Introducing: Heterogenous Computing

• Shift towards incorporating diverse range of computer architectures: CPUs, GPUs, FPGAs, ASICs.

• OpenCL language designed for code to be executed on diverse hardware “targets”.
Why Memory Access Behaviour Matters

- Memory accesses are a major cause of bottlenecks on modern computer architectures.
- Spatial Locality for Caches: Programs that frequently access nearby memory addresses tend to have better performance.
What patterns in the memory accesses performed by a program are good for performance on varying hardware targets?
“Develop a method to help HPC developers understand how their \textit{code} interacts with \textit{memory} – independent of the target hardware platform.”
Introducing: AIWC ('air-wik)

• Architecture Independent Workload Characterisation (AIWC) tool for OpenCL – Developed by Beau Johnston and Josh Milthorpe.

• Plugin for the Oclgrind simulator for OpenCL.
  ➢ Executes OpenCL kernels on abstract virtual OpenCL devices
  ➢ Follows OpenCL memory and execution model

• Architecture-Independent Oclgrind simulation allows for architecture-independent analysis of OpenCL code.
Introducing: AIWC ('air-wik')

• Collect metrics that characterise parallel programs.

• Metrics collected are independent of the hardware target of an OpenCL kernel.

• Memory based metrics:
  ➢ Total Memory Footprint: How much memory access occurs. \((Lower\ is\ better)\)
  ➢ Memory Address Entropy: Measure of spread of memory regions accessed. \((Lower\ is\ better)\)
A Test-Case Kernel for Optimisation

```c
__kernel void simpleMultiply(
    __global float *A,
    __global float *B,
    __global float *C,
    int N)
{
    float acc = 0.0f;
    for (int k = 0; k < N; ++k) {
        acc += B[k * N + globalCol] 
            * A[globalRow * N + k];
    }
    // Store the result
    C[globalRow * N + globalCol] = acc;
}
```

(NVIDIA Corporation. Cuda C best practices guide. 2019.)
Coalescing Accesses to Matrix A (coalescedA)
Coalescing Accesses to Matrix B (coalescedAB)
Efficient Local Memory Usage (**coalescedABT**)
Performance Results
Creating new AIWC Metrics!

- **Observation**: Accesses to “local” memory (or fast access on-chip memory) are good
  - **New Metric**: Relative Local Memory Usage (RLMU). *(Higher is better)*

- **Observation**: Parallel accesses to nearby memory addresses are good
  - **New Metric**: Parallel Spatial Locality (PSL).

![Set of memory addresses](image-url)

- Entropy Statistic (Real number)

  - Set of memory addresses
  - 3 bits skipped
The Parallel Spatial Locality Metric

**Formal Definition:**
Calculate entropy (or spread) of memory addresses at each timestep.
Repeat entropy calculations at varying "skipped-bits"
Calculate the following:

\[ PSL_{n-bits}(t) = \sum_{\alpha \in A_n(t)} p_\alpha \log_2(p_\alpha^{-1}) \]  

(1)

with \( A_n(t) \) the set of addresses accessed at time \( t \) accessed after skipping \( n \) bits, \( p_\alpha \) the probability of a specific address.
Average this value across all timesteps of program execution to obtain \( PSL_{n-bits} \).
A higher number of threads in an OpenCL workgroup leads to higher \( PSL_{n-bits} \) values. We normalise the \( PSL_{n-bits} \) by dividing by \( \log_2(n_{threads-per-group}) \).

• Main takeaway: the steeper the drop in PSL as the number of bits skipped increases, the more localised the memory accesses are.
## Preliminary Findings

<table>
<thead>
<tr>
<th></th>
<th>simple</th>
<th>coalescedA</th>
<th>coalescedAB</th>
<th>coalescedABT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total memory footprint</strong></td>
<td>196608</td>
<td>196608</td>
<td>196608</td>
<td>196608</td>
</tr>
<tr>
<td><strong>90% Memory Footprint</strong></td>
<td>118196</td>
<td>56176</td>
<td>489</td>
<td>489</td>
</tr>
<tr>
<td><strong>Global MAE</strong></td>
<td>17.02</td>
<td>13.18</td>
<td>9.78</td>
<td>9.78</td>
</tr>
<tr>
<td><strong>LMAE #bits=3</strong></td>
<td>16.02</td>
<td>12.18</td>
<td>8.78</td>
<td>8.78</td>
</tr>
<tr>
<td><strong>LMAE #bits=10</strong></td>
<td>9.02</td>
<td>5.18</td>
<td>1.78</td>
<td>1.78</td>
</tr>
<tr>
<td><strong>Relative Local Memory Usage</strong></td>
<td>0</td>
<td>0.50</td>
<td>0.94</td>
<td>0.94</td>
</tr>
</tbody>
</table>
Findings
Testing on Extended OpenDwarfs (EOD) Benchmark Suite

• The EOD benchmarks are a set of diverse OpenCL codes satisfying each of the 13 Berkeley Dwarfs:
  ◦ N-body methods
  ◦ Dense Linear Algebra
  ◦ Finite State Machines
  ◦ Structured Grids
  ◦ Graph Traversal
  ◦ and more...

• OpenCL codes representative of each dwarf typically induce similar memory access patterns.
Results

Figure 3: Parallel spatial locality metric for selected OpenDwarfs benchmark kernels
GEM: N-body Methods OpenDwarfs Benchmark

![Diagram showing vertices and atoms distribution across workgroups](image-url)
Needleman-Wunsch: Dynamic Programming OpenDwarfs Benchmark
CSR: Sparse Linear Algebra OpenDwarfs Benchmark

Regular matrix format

\[
\begin{bmatrix}
2 & 0 & 1 & 0 \\
0 & 3 & 6 & 0 \\
8 & 0 & 0 & 0 \\
0 & 0 & 10 & 12
\end{bmatrix}
\]

\[
A \rightarrow \begin{bmatrix}
2 & 1 & 3 & 6 & 8 & 10 & 12 \\
0 & 2 & 1 & 2 & 0 & 2 & 3 \\
0 & 2 & 4 & 5 & 7
\end{bmatrix}
\]

Sparse format

Work-group 0
Work-item 0 Work-item 1

\[
\begin{bmatrix}
2 & 1 \\
3 & 6 \\
8 \\
10 & 12 \\
8 & 3 \\
24 & 18 \\
32 \\
30 & 60
\end{bmatrix}
\]

Work-group 1
Work-item 0 Work-item 1

\[
\begin{bmatrix}
4 & 8 \\
3 & 5 \\
11 & 42 \\
32 \\
90
\end{bmatrix}
\]

Graph showing the relationship between the number of bits skipped and the Parallel Spatial Locality.
Conclusions and Future Work

• Proposed two new metrics to AIWC framework.

• Parallel Spatial Locality is the first architecture independent metric of its kind for parallel programs.
  ➢ Tested the metric against the Extended OpenDwarfs Benchmarking Suite.

• Improve AIWC to help HPC developers better understand (and optimise) their complex codes.

• Extend current methodology to create metrics for:
  ➢ Different optimisation strategies (not only memory-based ones).
  ➢ Different target architectures – CPUs and FPGAs.