COMPILER-BASED MEMORY ALLOCATION FOR DRAM-HBM HYBRID MEMORY SYSTEM

Onkar Patil, Frank Mueller, Latchesar Ionkov, Michael Lang
North Carolina State University, Los Alamos National Laboratory
TECHNOLOGY

- Hybrid Memory systems with High Bandwidth Memory (HBM) exist
  - Summit @ ORNL -> Volta GPUs with HBM2, NV-link
- Potential devices for future systems
  - AMD Vega GPU architecture
  - Intel’s Stratix 10MX FPGA
- HBM device manufacturers
  - Samsung
  - Micron
  - Intel
  - SK Hynix
• Language key words extensions or macros help in memory allocation, but…
  • Onus of utilizing the memory technologies optimally, falls on the skill and knowledge of the programmer
• Not every data structure and compute kernel benefits from the same memory
PROPOSED SOLUTION

• Use Static analysis from the compiler to automatically identify and classify critical data structures and kernels based on
  • Scope
  • Nesting (Nesting score)
  • Access Pattern (r, w, rw)
  • Proximity
  • Effective bandwidth ratios

• Perform source-to-source transformation to change the allocation and then recompile
  • Allocate memory using a single interface for all memory devices (SICM)
ASSUMPTIONS

- Memory architecture consistent with the envisioned Exascale Node architecture
- Data Flow can exist in two ways
  - Partitioned address space
  - Cached Memory system
- NUMA access to all memory devices

MEMORY DEVICE CHARACTERIZATION

- Run micro-benchmark once
  - Based on STREAM
  - Double loop timed
  - 512 MB workload
  - Averaged over 30 runs
- Identify NUMA devices with the underlying memory technology
  - Classification using K-means
  - Memory classes provided by user (HBM, DRAM, NVM)
- Bandwidth and Latency numbers for different types of operations

<table>
<thead>
<tr>
<th></th>
<th>HBM</th>
<th>DRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg_BW</td>
<td>1521.722599</td>
<td>1273.181041</td>
</tr>
<tr>
<td>R_BW</td>
<td>1012.372162</td>
<td>991.869843</td>
</tr>
<tr>
<td>W_BW</td>
<td>2031.073037</td>
<td>1554.492239</td>
</tr>
<tr>
<td>RW_BW</td>
<td>294.457057</td>
<td>279.220723</td>
</tr>
<tr>
<td>Ran_BW</td>
<td>7.235049</td>
<td>8.159102</td>
</tr>
<tr>
<td>Lin_BW</td>
<td>53.822347</td>
<td>53.39672</td>
</tr>
<tr>
<td>Same_BW</td>
<td>273.052658</td>
<td>266.690019</td>
</tr>
<tr>
<td>Diff_BW</td>
<td>132.100891</td>
<td>130.594896</td>
</tr>
<tr>
<td>Avg_lat</td>
<td>2.9E-09</td>
<td>3.4E-09</td>
</tr>
<tr>
<td>R_lat</td>
<td>3.8E-09</td>
<td>3.9E-09</td>
</tr>
<tr>
<td>W_lat</td>
<td>0.000000002</td>
<td>2.9E-09</td>
</tr>
<tr>
<td>RW_lat</td>
<td>0.000000013</td>
<td>1.37E-08</td>
</tr>
<tr>
<td>Ran_lat</td>
<td>5.273E-07</td>
<td>4.675E-07</td>
</tr>
<tr>
<td>Lin_lat</td>
<td>7.09E-08</td>
<td>7.14E-08</td>
</tr>
<tr>
<td>Same_lat</td>
<td>0.00000014</td>
<td>1.43E-08</td>
</tr>
<tr>
<td>Diff_lat</td>
<td>2.89E-08</td>
<td>2.92E-08</td>
</tr>
</tbody>
</table>

BW - MB/s
lat - s
STATIC ANALYSIS

- Information Gathered
- Data Structures
  - Scope → global, function, loop, conditional
  - Memory operation → r, w, r+w
  - Access Pattern → sequential, strided, linear, random
  - Aliasing → pass by reference
  - Code Location → place of declaration
  - Proximity → accessed in the same expression or scope
STATIC ANALYSIS

- Loops and Conditionals
  - Code Location ➔ Start and end of code, line & column nos.
  - Nesting score ➔ How nested is the loop/conditional? Function = 1, Loops/conditionals = 2, 3, ...
PROXIMITY SCORE

• Metric for
  • Grouping data structures that are frequently accessed together
  • Indicating appropriate memory allocation

\[ \text{Proximity\_score}(a,b) = p(a,b) \times \frac{L_{\text{Ins}}}{C_{\text{cns}}} \sum_{x = \{a, b\}} (o_{\text{s}}(x) \times p_{\text{s}}(x)) \]

• \( p \rightarrow \) proximity \{Normalized to different expression BW\}
• \( L_{\text{F}} \rightarrow \) loop factor \{10\}
• \( C_{\text{F}} \rightarrow \) conditional factor \{2\}
• \( L_{\text{ns}} \rightarrow \) loop nesting score \{1…n\}
• \( C_{\text{ns}} \rightarrow \) conditional nesting score \{1…n\}
• \( o_{\text{s}} \rightarrow \) operation score \{Normalized to read-only BW\}
• \( p_{\text{s}} \rightarrow \) pattern score \{Normalized to random access BW\}
int main()
{
    ....
    for(i=0;i<max,i++){
        a[i] = b[i]*c[i] - d[i];
    }
    ....
}

ARENA ALLOCATION

- Clusters of data structures
  - Proximity scores
  - Group them using K-means clustering algorithm
- Allocate higher scored clusters on HBM

Arena 1:
- a, b, c, d, i

Arena 2:
- DRAM

Arena 3:
- NVRAM

HBM
IMPLEMENTATION

• Microbenchmark characterization
• Clang obtains code locations and scope of all data structures and loops
• LLVM opt obtains SSA form and perform analysis
  • Obtain access patterns, memory operations, aliasing and nesting score for all data structures and loops
  • Make allocation decisions
• Shell script to make source-to-source transformations and re-compile
EXPERIMENTATION

• Test on Benchmarks (C, OpenMP, MPI)
  • LULESH
  • VPIC
  • SNAP
  • HPCC
  • KRIPKE
  • CLAMR
  • AMG2013
  • MCB
  • QMCPACK
  • CAM-SE

• Compare the framework with manual allocation for DRAM-HBM system
  • BCDA analysis in LLVM
  • Uses hbw_malloc

  • Creates a memory monitoring tool using LLVM

  • Checks a directory if data is present before access and moves data accordingly
FUTURE WORK

• Finish the framework
  • Handling of complex variables
  • Implement the proximity score calculation
  • Source-to-source translation
• Evaluate on the benchmarks shortlisted
• Write up the paper
THANK YOU

• Questions?