A Framework for Emulating Non-Volatile Memory Systems with Different Performance Characteristics

Dipanjan Sengupta\textsuperscript{1,2}, Qi Wang\textsuperscript{1,3}, Haris Volos\textsuperscript{1}, Lucy Cherkasova\textsuperscript{1}, Guilherme Magalhaes\textsuperscript{1}, Jun Li\textsuperscript{1}, Karsten Schwan\textsuperscript{2}

\textsuperscript{1}Hewlett-Packard Labs
\textsuperscript{1}Georgia Institute of Technology
\textsuperscript{3}The George Washington University
New HP Labs Project “The Machine” aims to create a new computing architecture

- Shift to *non-volatile, byte-addressable memory* (e.g., Phase-Change Memory and Memristor)
- NVM is not commercially available yet
- Future NVM technologies will offer a variety of different performance characteristics (2-10 slower than DRAM)
- It is extremely difficult to *predict and optimize* performance of complex application’s on a future hardware:
  - Which ranges of latencies and bandwidth are critical for achieving good performance and scalability of different application groups?
Future machines will have both DRAM and NVM

There are many open questions:

- Shall we consider DRAM as a caching layer for NVM?
- Shall we build systems with two types of memory: fast DRAM and slower NVM?
- What are the strategies for efficient data placement?
Goal: Build a *performance emulator* for NVM using a commodity hardware (DRAM)

Two performance knobs for NVM emulation:
- *bandwidth*
- *latency*

Additional challenge and goal:
- Validation experiments to check correctness and accuracy of the emulation platform
Throttling memory bandwidth using hardware-based approach

- Thermal Control Registers in recent Intel-based processors
- Separate knobs for controlling memory read and write bandwidth
Challenges for Latency Emulation

Commodity hardware doesn’t support hardware mechanism to control the memory latency

Challenges for *software-based latency emulation*

- Cannot instrument every memory reference from the application due to high overhead
- Not all memory references access main memory
  - They might be cached by private and shared last level cache
- Memory level parallelism in modern processors
**Memory Latency Emulation**

**Software-based approach:**
- Modeling average application perceived memory latency to be close to target NVM latency
- Injecting software created delays at *epochs* granularity
- Performance-counters based memory model

![Memory Access Chart](chart.png)
A very simple memory model for computing the additional delay $\Delta_i$ for a given epoch $i$ is the following:

$$\Delta_i = M_i \cdot (NV M_{lat} - DRAM_{lat})$$

where $M_i$ is the number of memory references for a given epoch $i$. 
Modern processors take advantage of memory level parallelism (MLP)

Latency emulation model needs to be MLP aware

\[ \Delta_i = \frac{M_i}{O_i} \cdot (NVM_{lat} - DRAM_{lat}) \]
Latency Emulation Model

\[ \Delta_i = \frac{LDM\text{-STALL}_i}{DRAM_{lat}} \times (NVM_{lat} - DRAM_{lat}) \]

We have implemented this model and prototyped our emulator for two popular Intel processor families:
- Sandy Bridge
- Ivy Bridge
How to evaluate correctness and accuracy of the proposed model and its implementation?

- Exploit that memory access latencies are different across different NUMA domains in a multi-socket machine:
- Access latency to local node and remote memory:
  - Ivy Bridge: 87 ns and 176 ns
  - Sandy Bridge: 97 ns and 162 ns
Validation Experiments

Emulating Remote DRAM Latency

Local node

Remote node

Executing on Remote DRAM

Local node

Remote node
LLC miss per 1000 instructions signifies the memory intensity of an application.

**Average emulation error** across various memory and compute intensive SPEC benchmarks is **1.8 %**.
Conclusion & Future Work

• This work: proof of concept for single-threaded applications
  • Combination of hardware- and software-based online approach for NVM emulation

• Ongoing work: emulation support for multi-threaded applications

• Support for NVM and DRAM using the same platform
Thank you!

Questions?