Remove the Memory Wall:  *From performance modeling to architecture optimization*

Xian-He Sun  
*Scalable Computing Software Laboratory*  
Illinois Institute of Technology  

sun@iit.edu  

Nov. 12, 2006
Scalable Computing Software (SCS) Lab.

Distributed Optical Testbed (Grid)

Parallel Computers at SCS

- Performance Environment for Grid Computing
  - Performance prediction, task scheduling, and workflow
- Performance Environment for HEC
  - Reduce data access delay (this talk)
- Distributed Computing Environment
  - Process migration, dynamic virtual machine
The Memory Wall

Processors are getting faster more quickly than memory

“Moore's Law”

Processor-Memory Performance Gap: (grows 50% / year)

Solutions
- Cache memories
- Prefetching
- Multithreading
Limitation of Memory Hierarchy

- Does not work well when application lacks locality
- Traditionally, processing processor issues data access instructions
- Processor stalls when data has to be accessed from lower levels of memory hierarchy
- Solution: Prefetching - fetch the data before the processor demands for it
Existing Prefetching

- Move data closer to the processor before it is demanded
- Prefetch data as close as possible to the processor in the memory hierarchy
- Key to prefetching
  - What data should be prefetched?
  - When should prefetching occur?
What to Fetch?

• Requires prediction of what data the processor is going to access **in the future**
• Prefetching Strategies
  – Sequential
  – Adaptive Sequential
  – Strided
  – Markov Prefetching
  – Distance Prefetching
When to Fetch?

• Not too early – might replace useful data
• Not too late – causes a cache miss / page fault
• Best, if time between now and next access is equal to prediction time + overhead to fetch the data (performance evaluation)

• Prefetch-on-miss
  – when a cache miss occurs, next predicted address is prefetched

• Prefetch Always
  – for each demand reference

• Tagged Prefetching
  – when a data block is first accessed
Limitation of Existing Prefetching

- No universal strategy to dynamically adapt to data access patterns
  - Different prefetching algorithms work well for different data access patterns
    (Performance modeling)
- No prediction strategy on when to prefetch
  - Triggered by cache miss
    (Performance evaluation)
- Only works for simple access patterns with locality
  - Aggressive prefetching is too costly
    (Performance optimization)
Existing Memory Servers

- Large memory and large cache at each node is costly
- Provide extra memory space over network
  - Accessing the memory over network is faster than accessing disk
  - GMS, DoDo, NMS, PNR
- MME helps the client locate the server and provide interface to move data from the server to the client
- **Limitation:** Improvement is insignificant
Helper Thread for Multi-core Processors

• Use a helper thread to pre-execute slices of code and cause cache misses early for main thread
• Compiler creates helper threads
• Memory side prefetching
  – Prediction based memory-side prefetching
  – Pre-execution based data forwarding
• Depends on compilers
• Thread creation, synchronization, and termination
• Focus on hardware support
The Data Access Memory Server (DMS)

• **Separate** data access with information processing

• A **push model** based **data access** service

• In addition to provide memory space, as the conventional memory server, it pushes **data** to the processing unit **on time**

• Special architectures are designed. **Aggressive Prefetching**, **data access pattern** identification, and **performance modeling** mechanisms are adopted at cache level and at the memory level

• Cache hit rates increase well above **90%** for various bad benchmarks

Sun and Byna 05
DMS – Prefetch Strategy

- **Prefetch Engine (PFE)**
  - Prefetch predictor (*What*)
  - Request generator (*When*)
    (software solution)

- **Memory Management Engine (MME)**
  - Data Propeller: Issues the prefetch instructions
  - Pushes the data from the server to the clients
  - Deals with raw cache misses or page faults
    (hardware support)
Challenges in Implementing the DMS

**Performance modeling, evaluation, optimization**

- Aggressive and on-time prefetching
- Fetch and replacement policies
- Recognizing the data access patterns

**Hardware support**

- Accessing the L1 cache and the data bus
- Writing the modified data back
- Effect of network latency and traffic
What to Fetch

• Aggressive Prefetching
  – Overhead to predict the future accesses is no longer a issue
  – New aggressive methods to predict irregular data accesses

• Adapt a prefetch strategy based on the data access pattern

• Reduce prediction time by using hints provided by compiler and application/user
Identify and Match Access Pattern

- Classification of data access patterns based on non-contiguity between accesses and the repetitive behavior of patterns

Byna, Sun, Gropp, Thakur 04
When to Fetch

Three factors

• Overhead to predict the future accesses
  – Based on the chosen prefetching method
• Data transfer overhead
  – Memory access delay model
• Time till next cache miss
  – Data access model
• Overlapping the network latency by increasing the prefetch distance
• Adapting the prefetch distance based on the network latency variation
Predicting Memory Access Cost

Cameron and Sun 03

Average memory access cost = Hit time + Miss Rate * Miss Penalty

= (Number of TLB hits * Time to access TLB) +
(Number of TLB misses * TLB miss penalty) +
(Number of L_1 hits) * (Time to access L_1) +
(L_1 misses * L_1 penalty) + (L_2 misses * L_2 penalty)

+ … +

(L_M misses * L_M penalty)

Total Miss penalty:

\[ T_m = \sum_{k=1}^{M} (M_k * T_k) - \alpha \]

\[ M_k = \sum_{i=1}^{m} M_{(k,i)}^c + \sum_{i=1}^{n} M_{(k,i)}^n \]
DMS – Architecture Design

• Multiprocessor Platforms
  – Clusters
  – SMP
  – Multicore Processor

• Classified based on the functionality of PFE and MME

• Pure Server Model
  – DMS manages the local memory of clients as well as the memory on the server and the memory provided by idle nodes

• Hybrid Server Model
  – Local memory is maintained by the client and DMS pushes the data towards the client’s memory

• I/O Server Model
**Potential of DMS – Benchmark Kernels**

- Kernels from SPEC 2000, BLAS, Stream benchmarks
- Represent various data access patterns
- Copy – contiguous
- Combinations of contiguous and non-contiguous patterns
- Irregular patterns
- Irregular pointer chasing accesses
- I/O accesses

### Table 1. Benchmark kernels

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Operation</th>
<th>Access Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>for (i = 0; i &lt; N; i++) y[i] = x[i];</td>
<td>y: contiguous; x: contiguous</td>
</tr>
<tr>
<td>2d-matrix transpose</td>
<td>for (i = 0; i &lt; N; i++) for (j = 0; j &lt; N; j++) y[i][j] = x[i][j];</td>
<td>y: contiguous; x: non-contiguous</td>
</tr>
<tr>
<td>2d-matrix multiplication</td>
<td>for (i = 0; i &lt; N; i++) { for (j = 0; j &lt; N; j++) { t = 0; for (k = 0; k &lt; N; k++) { t += a[i][k]*b[k][j]; c[i][j] = t; } } }</td>
<td>a: contiguous; b: non-contiguous; c: contiguous</td>
</tr>
<tr>
<td>struct accesses</td>
<td>for (i = 0; i &lt; N; i++) { type_a[i]-&gt;longval1 = a[i]; type_a[i]-&gt;longval4 = b[i]; type_a[i]-&gt;longval8 = c[i]; }</td>
<td>type_a: non-contiguous, irregular stride of repeating 1,64 and 64, a, b, c: contiguous</td>
</tr>
<tr>
<td>pointer chasing</td>
<td>for (i = 0; i &lt; N; i++) { ptr = a[i]; while (ptr) { &lt;compute&gt; ptr-&gt;next = ptr; } }</td>
<td>a: Array of linked list nodes ptr: linked list data structure traverse</td>
</tr>
<tr>
<td>file accesses</td>
<td>for (i = 0; i &lt; N; i++) { fsets (buf, bufsize, fname); fseek(fd, offset, current); &lt;compute&gt; }</td>
<td>File is accessed with an offset between each access, non-contiguous pattern</td>
</tr>
</tbody>
</table>
Potential of DMS – L1 cache miss rate

![Bar graph showing potential of DMS prefetching - Hitrate](chart.png)
L1 Cache Miss Rate – SPEC2000 Benchmark

- SPEC2000 benchmarks with high L1 cache miss rates
Benchmark – IPC Improvement

![IPC Improvement Graph]

- ammp: 245.25%
- applu: 15.52%
- art: 28.23%
- mcf: 23.95%
- mgrid: 70.69%

Legend:
- Strided
- DMS (without dedicated core)
- DMS (with dedicated core)
Potential of DMS – File accesses

![File access - Page hit rate graph](image)

- Hit rate %
- Offset (bytes)
- Base case
- Strided prefetching
- DMS prefetching
Conclusion

• New challenges in performance measurement, evaluation, and optimization (PMEO)
  – Combine PMEO in both application and computer architecture
  – Application PMEO at low level
  – System software solution with hardware support

• Memory (I/O) as a service
  – DMS proactively and adaptively pushes the data closer to the processor
  – Adaptive and timely prefetching strategies
  – Has the potential to avoid CPU stall time

• An Exciting New Era of PMEO
  – Application driven and Dynamic adaptation
  – QoS, Trusted computing, Fault tolerance, Dynamic environment, Architecture design
Questions?