FULL-W2V: Fully Exploiting Data Reuse for Word2Vec on GPU-Accelerated Systems

Thomas Randall, Tyler Allen, Rong Ge
{tlranda, tnallen, rge}@clemson.edu

Funded in part by NSF Grants: CCF-1551511 and CNS-1551262
GPU W2V Not Faster than CPU

- 3-layer ANN
  - Words $w \rightarrow d$-dimensional embeddings $e$
  - Prior ports based on data-intensive implementation
  - Suboptimal usage of GPU memory hierarchy

Sentence Contents ("Context Windows")

Noise Contrastive Samples ("Negatives")

- Context windows include adjacent words
- Noise contrastive samples include "negatives"
GPU W2V Not Faster than CPU

- 3-layer ANN
  - Words $w \rightarrow d$-dimensional embeddings $e$
  - Prior ports based on data-intensive implementation
  - Suboptimal usage of GPU memory hierarchy

Sentence Contents ("Context Windows")

```
Context windows include adjacent words
```

Noise Contrastive Samples ("Negatives")

```
put pastel rehearsal
```

Input  Projection  Output

$w_i$  $e_i$  $e_{i+2}$
$e_i$  $e_{i+1}$
$e_{i-1}$  $e_{i-2}$
GPU W2V Not Faster than CPU

- 3-layer ANN
  - Words $w \rightarrow d$-dimensional embeddings $e$
  - Prior ports based on data-intensive implementation
- Suboptimal usage of GPU memory hierarchy

Sentence Contents ("Context Windows")

Noise Contrastive Samples ("Negatives")

Input $\rightarrow$ Projection $\rightarrow$ Output

- $w_i$ $\rightarrow$ $e_i$
- $e_{i-1}$
- $e_{i-2}$
- $e_{i+1}$
- $e_{i+2}$
GPU W2V Not Faster than CPU

- 3-layer ANN
  - Words $w \rightarrow d$-dimensional embeddings $e$
- Prior ports based on data-intensive implementation
  - Suboptimal usage of GPU memory hierarchy
- FULL-W2V reduces access and improves locality
  - Leverage memory hierarchy based on algorithm’s access pattern

Sentence Contents (“Context Windows”)

Input $\rightarrow$ Projection

Noise Contrastive Samples (“Negatives”)

Noise Contrastive Samples (“Negatives”)

- $e_{i+2}$
- $e_{i+1}$
- $e_{i-1}$
- $e_{i-2}$

Context windows include adjacent words

- agree
- ash
- short

Introduction

Problem

Techniques

Results

Conclusions
Negative Sample Reuse: Register-W2V

• Challenge: Negatives are random and have lower reuse
Negative Sample Reuse: Register-W2V

- Challenge: Negatives are **random** and have **lower reuse**
- Opportunity: Operations have **independent order**
Negative Sample Reuse: Register-W2V

- Challenge: Negatives are random and have lower reuse
- Opportunity: Operations have independent order
Negative Sample Reuse: Register-W2V

- Challenge: Negatives are random and have lower reuse
- Opportunity: Operations have independent order
- Solution: Use registers for maximum reusability
  - Minimize up-front memory latency, maintain locality
  - Improved pipeline utilization
  - Maintain scheduling flexibility, reduce stress for Shared Memory
Negative Sample Reuse: Register-W2V

- Challenge: Negatives are random and have lower reuse
- Opportunity: Operations have independent order
- Solution: Use registers for maximum reusability
  - Minimize up-front memory latency, maintain locality
  - Improved pipeline utilization
  - Maintain scheduling flexibility, reduce stress for Shared Memory
Context Word Reuse

• Different Pattern: Context Words have **more reuse**

$$i-1 \quad i \quad i+1 \quad i+2$$

Context windows **include** adjacent words

$${n_0} \quad {n_1} \quad {n_2}$$

**put** **pastel** **rehearsal**
Context Word Reuse

- Different Pattern: Context Words have more reuse
Context Word Reuse

- Different Pattern: Context Words have more reuse
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer
Context Word Reuse

- Different Pattern: Context Words have more reuse
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer

\[
i-1 \quad i \quad i+1 \quad i+2
\]

Context windows include adjacent words

Buffer:

\[
\text{Context} \quad \text{windows} \quad \text{include} \quad \text{adjacent}
\]

put pastel rehearsal
Context Word Reuse

- Different Pattern: Context Words have more reuse
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer

Buffer:

<table>
<thead>
<tr>
<th>Context</th>
<th>windows</th>
<th>include</th>
<th>adjacent</th>
<th>words</th>
</tr>
</thead>
</table>

\[ i-2 \quad i-1 \quad i \quad i+1 \quad i+2 \]

\[ n_3 \quad n_4 \quad n_5 \]
Context Word Reuse

- Different Pattern: Context Words have more reuse
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer
Context Word Reuse

- Different Pattern: Context Words have more reuse
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer

Buffer:

<table>
<thead>
<tr>
<th>Context</th>
<th>windows</th>
<th>include</th>
<th>adjacent</th>
<th>words</th>
<th>tiger</th>
<th>cemetery</th>
<th>scramble</th>
</tr>
</thead>
</table>

i−2       i−1       i       i+1       n6       n7       n8
Context Word Reuse

- Different Pattern: Context Words have **more reuse**
- Allocation: Shared Memory leverages longer-term reuse
  - High performance; Explicit control; Flexible scheduling
- Management: Ring buffer

![Diagram of Context Word Reuse]

Buffer:

```
windows | include | adjacent | words
```

Context words include adjacent words.
Results

- **FULL-W2V**: Register-W2V + Context Word Reuse
  - **4.35X** total speedup previous best on V100
  - **3.85X** speedup from Register-W2V only
  - Sum data demand reduced by **91.65%**

### Demand in GB/Epoch

<table>
<thead>
<tr>
<th>Implementation</th>
<th>L1/TEX</th>
<th>L2</th>
<th>DRAM</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closest Prior</td>
<td>1,134.448</td>
<td>493.614</td>
<td>226.578</td>
<td>100.0%</td>
</tr>
<tr>
<td>Register-W2V</td>
<td>885.065</td>
<td>781.576</td>
<td>66.555</td>
<td>78.02%</td>
</tr>
<tr>
<td>FULL-W2V</td>
<td>94.760</td>
<td>88.723</td>
<td>41.851</td>
<td>8.35%</td>
</tr>
</tbody>
</table>

### Millions Words/Second

- **FULL-W2V**
  - GPU: V100, TitanXP, Broadwell, P100
  - CPU: Skylake, Broadwell, Haswell

**Introduction**

**Problem**

**Techniques**

**Results**

**Conclusions**
Insights and Conclusion

• We present FULL-W2V
  • 4.35X prior SOTA on V100
  • 2.99X scaling from P100 to V100
• Different storage for different data
  • Register-W2V: maximize short term reuse in register
  • FULL-W2V: maximize long term reuse in shared memory
• Looking for more?
  • Our code is open source: https://github.com/tlranda/FULL-W2V
  • See the extended presentation for additional details
Acknowledgements

• Thomas Randall
  • tlranda@clemson.edu
  • tlranda.people.clemson.edu
  • https://www.researchgate.net/profile/Thomas-Randall-5
• Tyler Allen
  • tnallen@clemson.edu
  • tnallen.people.clemson.edu
  • https://www.researchgate.net/profile/Tyler-Allen-2
• Rong Ge
  • rge@clemson.edu
  • people.cs.clemson.edu/~rge/
• Support from NSF Grants CCF-1551511 and CNS-1551262
• Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster