Learning Memory Access Patterns

Milad Hashemi [1] **Kevin Swersky** [1] **Jamie A. Smith** [1] **Grant Ayers** [2 *] **Heiner Litz** [3 *] **Jichuan Chang** [1]
Christos Kozyrakis [2] **Parthasarathy Ranganathan** [1]

Appendix

A. Interpreting t-SNE Plots

By mapping PCs back to source code, we observe that the model has learned about program structure. We show examples from two of the most challenging *SPEC CPU2006* applications to learn, *mcf* and *omnetpp*.

[*]Equal contribution [1]Google [2]Stanford University [3]University of California, Santa Cruz. Correspondence to: Milad Hashemi <miladh@google.com>, Kevin Swersky <kswersky@google.com>.

Proceedings of the 35^{th} *International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018.

A.1. mcf

The following function from *mcf* appears in two different t-SNE clusters:

```
1       while( node )
2       {
3           if( node->orientation == UP )
4               node->potential = node->
        basic_arc->cost + node->pred->potential
        ;
5           else /* == DOWN */
6           {
7               node->potential = node->pred->
        potential - node->basic_arc->cost;
8               checksum++;
9           }
10          tmp = node;
11          node = node->child;
12          node = tmp;
13
14          while( node->pred )
15          {
16              tmp = node->sibling;
17              if( tmp )
18              {
19                  node = tmp;
20                  break;
21              }
22              else
23                  node = node->pred;
24          }
25      }
26
```

One cluster contains only different instances of line 4, unrolled into three different instructions at three different PCs. We show the line of code, followed by the assembly code in (PC: Instruction) format:

```
node->potential = node->basic_arc->cost
 + node->pred->potential;
401932: mov  0x18(%rdx),%rsi
401888: mov  0x18(%r10),%rsi
4018df: mov  0x18(%r11),%rsi
```

A second cluster identifies only the PCs responsible for the linked list traversal, at lines 11 and 16:

```
node = node->child;
401878: mov  0x10(%rdx),%r10
40187c: mov  %rcx,(%rdx)

tmp = node->sibling;
4019a2: mov 0x20(%r9),%rcx
```

A.2. omnetpp

We show the result of running t-SNE on the learned (Δ, PC) embeddings of *omnetpp* in Figure 1.

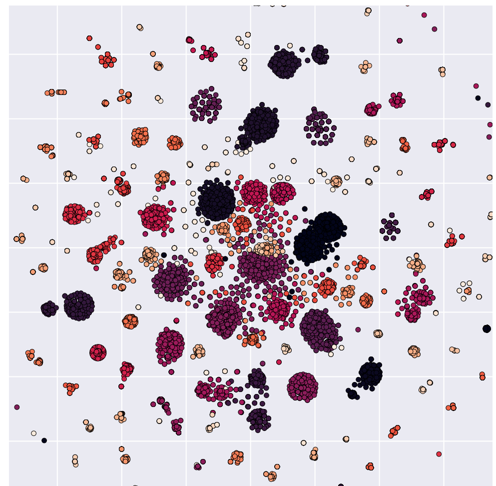

Figure 1. A t-SNE visualization of the concatenated (Δ,PC) embeddings on *omnetpp* colored according to PC instruction.

Examining some of the clusters closely, we find interesting patterns. The following code inserts and removes items into an owner's list:

```
// remove from owner's child list
if (ownerp!=NULL)
{
     if (nextp!=NULL)
          nextp->prevp = prevp;
     if (prevp!=NULL)
          prevp->nextp = nextp;
     if (ownerp->firstchildp==this)
          ownerp->firstchildp = nextp;
     ownerp = NULL;
}
// insert into owner's child list as
first elem
if (newowner!=NULL)
{
     ownerp = newowner;
     prevp = NULL;
     nextp = ownerp->firstchildp;
     if (nextp!=NULL)
          nextp->prevp = this;
     ownerp->firstchildp = this;
}

```

The main insertion and removal path are both shown in the same t-SNE cluster:

```
// Removal
nextp->prevp = prevp;
448a6a: mov  0x20(%rbx),%r12
448a6e: mov  %r12,0x20(%r10)
//Insertion
nextp = ownerp->firstchildp;
44c23a: mov  0x30(%rax),%r13
44c23e: mov  %r13,0x28(%r12)
```

omnetpp's t-SNE clusters also contain many examples of comparison code from very different source code files that are used as search statements being mapped to the same t-SNE cluster. Since these comparators are long, they get compiled to many different assembly instructions, so we only show the source code below. Lines 3 and 17 are both mapped to the same t-SNE cluster among other similar comparators:

```
1   cObject *cArray::get(int m)
2   {
3       if (m>=0 && m<=last && vect[m])
4           return vect[m];
5       else
6           return NULL;
7   }
8
9   void cMessageHeap::shiftup(int from){
10      // restores heap structure (in a
    sub-heap)
11      int i,j;
12      cMessage *temp;
13
14      i=from;
15      while ((j=2*i) <= n)
16      {
17      if (j<n && (*h[j] > *h[j+1]))   //
    direction
18          j++;
19      if (*h[i] > *h[j])  //is change
    necessary?
20      {
21           temp=h[j];
22           (h[j]=h[i])->heapindex=j;
23           (h[i]=temp)->heapindex=i;
24           i=j;
25      }
26      else
27          break;
28      }
29  }
30
```

B. Experimental Results

The experimental results for precision/recall are given in Table 1/Table 2 respectively.

C. LSTM Hyperparameters

The hyperparameters for both LSTM models are given in Table 3

D. K-Means Clustering on an Address Trace

In Figure 2 we show the results of running k-means with 6 clusters on 10^6 addresses from *omnetpp*.

Table 1. Experimental Results: Precision

Dataset	Stream	GHB	Embedding	Kmeans	Only PCs	Only Deltas
bwaves	0.65	0.07	0.89	0.93	0.89	0.89
gems	0.61	0.05	0.76	0.82	0.76	0.59
leslie3d	0.72	0.21	0.99	0.80	0.99	0.93
libquantum	0.99	0.99	0.99	0.99	0.99	0.99
soplex	0.68	0.18	0.73	0.83	0.73	0.70
sphinx3	0.72	0.08	0.97	0.81	0.96	0.86
astar	0.34	0.25	0.60	0.51	0.60	0.32
lbm	0.0001	0.0001	0.99	0.59	0.99	0.99
mcf	0.0001	0.18	0.33	0.45	0.33	0.28
milc	0.0001	0.02	0.56	0.82	0.56	0.56
omnetpp	0.08	0.06	0.63	0.53	0.62	0.51
websearch	0.1	0.12	0.43	0.55	0.41	0.41
Geometric Mean	0.11	0.06	0.70	0.69	0.70	0.61

Table 2. Experimental Results: Recall

Dataset	Stream	GHB	Embedding	Kmeans	Only PCs	Only Deltas
bwaves	0.86	0.38	0.10	0.93	0.05	0.06
gems	0.83	0.36	0.20	0.85	0.04	0.20
leslie3d	0.87	0.41	0.99	0.80	0.38	0.98
libquantum	0.99	0.99	1.00	1.00	1.00	1.00
soplex	0.95	0.41	0.14	0.83	0.14	0.14
sphinx3	0.89	0.30	0.57	0.81	0.46	0.58
astar	0.55	0.51	0.15	0.59	0.03	0.15
lbm	0.98	0.61	1.00	0.82	0.98	0.98
mcf	0.21	0.31	0.13	0.50	0.12	0.13
milc	0.21	0.05	0.10	0.82	0.001	0.04
omnetpp	0.64	0.22	0.19	0.59	0.18	0.19
websearch	0.57	0.20	0.32	0.59	0.23	0.27
Geometric Mean	0.72	0.39	0.27	0.75	0.12	0.24

Table 3. Training hyperparameters for each model.

Embedding	Network Size	128x2 LSTM
	Learning Rate	.001
	Number of Train Steps	500k
	Sequence Length	64
	Embedding Size	128
Clustering	Network Size	128x2 LSTM
	Learning Rate	.1
	Number of Train Steps	250k
	Sequence Length	64
	Number of Centroids	12

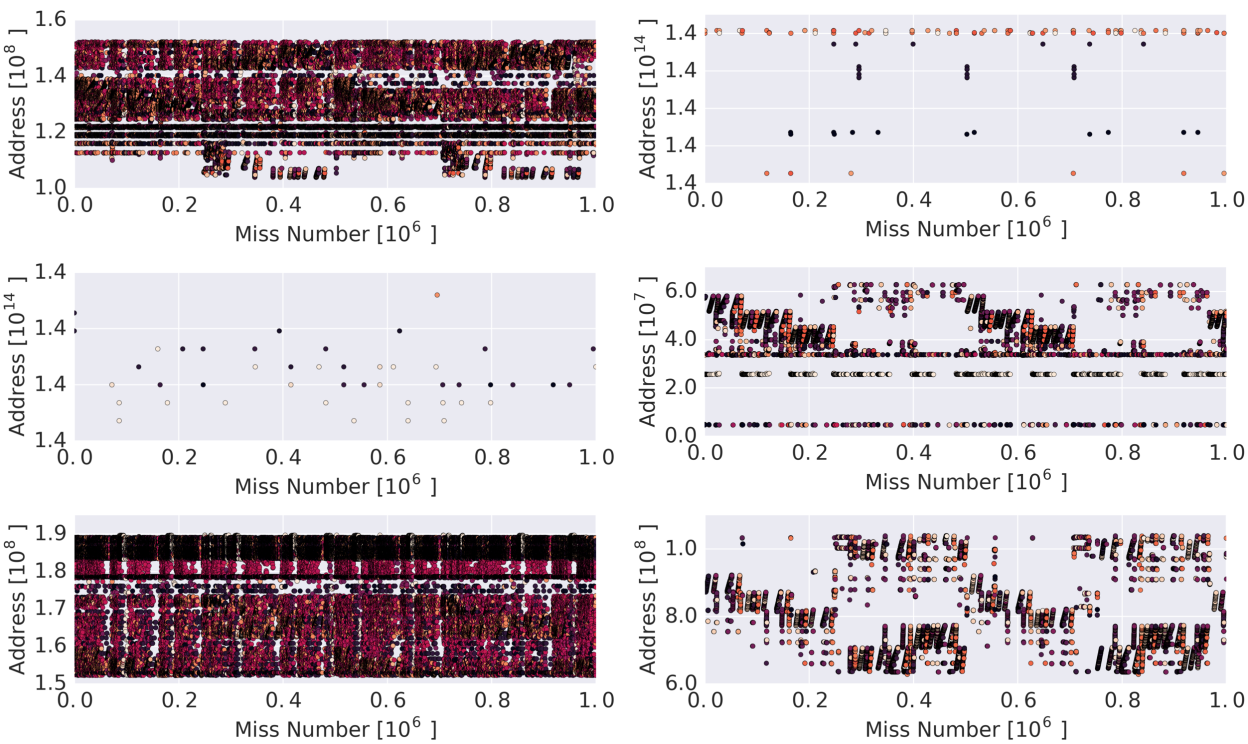

Figure 2. One million memory accesses from *omnetpp* after running k-means clustering on the address space.