Visualization and data reordering using EcoRAM

Robert G. Clapp

ABSTRACT

Memory size and input/output (IO) performance have not kept pace with the ever increasing size of seismic data volumes. Processing steps that involve random, or pseudo-random, access to data (such as visualizing, sorting, and transposing) further degrade performance. I use Spansion's EcoRAM to replace out-of-core visualization and data transpose schemes. I show between one and two orders of magnitude improvement in performance over conventional out-of-core solutions.

INTRODUCTION

A growing number of seismic applications are becoming IO bound. In some cases, an application can be IO bound because the amount of computation per input sample is low, such as with Normal Move-Out. Another category is applications where the input, intermediate, and output space which are to large to simultaneously reside in physical Random Access Memory (RAM), such as conventional 3-D Reverse Time Migration (RTM) (Baysal et al., 1983).

Several different methods exist to improve IO performance. Raid based disk arrays stripe data over many different disk drives. Network storage uses a collection of nodes as temporary storage. Large shared-memory and distributed-memory machines can distribute the required memory storage over 100s to 1000s of nodes, therefore allowing the entire problem to sit in RAM. Recently, solid state disks have offered promise. These disks, when used in a raid system, can offer 2-5x the IO performance of conventional disks. A new weapon in the IO bottleneck battle is EcoRAM from Spansion. EcoRAM sits in conventional RAM slots but is composed of a series of solid state disks. An accelerator replaces one of the CPU motherboard slots to facilitate low latency, read speeds of 30x conventional disk and $2-10$x write speeds.

I show that applications that are, or can be made, read dominant can benefit dramatically from EcoRAM with minimal programming changes. I begin by comparing IO characteristic of EcoRAM versus conventional RAM, disk, network, and solid state disk. I then describe how, and the results of, writing a visualization and transpose program to maximize EcoRAM performance.

BACKGROUND

EcoRAM is a new product from Spansion. One of its goals is to fill the very large performance gap between DRAM and conventional (and solid state) disks. Its main competitor in this market place is Fusion IO. Its four most relevant claimed advantages are: $\frac{1}{8}$ power use of conventional DRAM; four times the memory per DRAM slot; $2-10$x write speed advantage over conventional disk and solid state drives; and most importantly, read speeds on the same order of magnitude as conventional DRAM (1.3 GB/s vs 4-6GB/s). The first claim, though important, would not make it a worthwhile investment at the University level. The last three claims were of more interest to us. Two commonly run applications at SEP, visualization and data transposition, are extremely IO bound for large data sizes.

Visualization

The first example application visualization application (Clapp et al., 2008; Clapp and Nagales, 2008). Four and five dimensional spaces are common in processing. Often we are comparing multiple volumes of these sizes. For even 'small' sized problems, we quickly exceed conventional RAM. Several approaches can be taken to get around this bottleneck.

Out-of-core The data can be read 'on the fly'. In this approach, the desired view is created only when the user requests it. Predictive algorithms can be used to guess the next desired view. The downside of this approach is the significant difference in IO performance between RAM and disk, made worse when the user request a view that the smart algorithm didn't expect.

Compression The data can be compressed using techniques ranging from conversion of four to one byte to more sophisticated compression schemes such as curvelets. The downsides of these approaches include the difficulty associated with building an appropriate clip function, the need for higher order accuracy (such as seeing lower amplitude events) when interacting with the data, artifacts associated with compression, and for high compression schemes, some viewing options are difficult to achieve.

Cluster The data can be stored in RAM on a series of nodes. Requested views are translated into requests to different nodes and reconstructed by the host machine. The disadvantages of this scheme include the significant cost of these extra machines and latency af networking bottlenecks.

All of these approaches also suffer from a significant additional coding overhead.

Transpose

Transposing a dataset is a common seismic processing step. A given processing step might need data in offset gathers while the data exists in common offset sections. For source-receiver based wave equation migration, the natural axis order is: midpoint, offset, and depth while further processing steps need the data in depth, offset, and midpoint. For large data volumes (significantly beyond the CPUs memory) what looks like a rather trivial operation can take hours to days using a single CPU. The transpose time is dominated for large problems by disk IO. A solution to this problem is to stream a data volume to a cluster where the transpose can be done completely in-core. This is an effective technique but again adds significant programming overhead.

RESULTS

I used EcoRAM for the both visualization and transpose application. The server had 32 GB RAM, 512 GB of EcoRAM , and two AMD quad-core 2.6GHz processors.

Visualization

The visualization test does not lend itself well to paper form. The only change to the basic visualization application was to introduce `mmap` buffer and IO module. The total time to add this feature was less than 3 hours. I observed that I could move to random locations within the volume with the delay more a function of the Gigabit netwrok connection between the host EcoRAM server and the X11 terminal.

Transpose

All transposes can be mapped into a single five-dimensional template $(n1, n2, n3, n4, n5)$. The first axis is the the product of the data element size and the length of the fastest axes that are not be transposed over. The second axis is the fastest axis that needs to be transposed. The third axis is the product of the axes lengths between transposed axes. The fourth axis is the slowest axis that we want to transpose over. The fifth axis is the product of all of the axes slower than the last axis we wish to transpose over. We can extend the definition of the second and fourth axes to compose sets of adjacent axes to enable more sophisticated transposes. For example, in the case of wave equation migration, we begin with a five dimensional dataset with element size $esize$ that is ordered (midpoint x, midpoint y, hx, hy, depth) or of size $(ncmpx, ncmpy, nhx, nhy, nz)$ and we wish to end up with $(nz, nhx, nhy, ncmpx, ncmpy)$. We can simulate merging the first two axes of the input and map to our template $(esize, ncmpx * ncmpy, nhx * nhy, nz, 1)$.

Using this template, the transpose algorithm can be written in a push (loop over input) or pull (loop over output) manner. For EcoRAM , and in many other cases

(such as wanting to pipe the output), the pull method is more efficient. The basic transpose algorithm using the five dimensional template then takes the following form. This simple algorithm assumes that you can hold both the input and output matrices

Algorithm 1 Simple transpose

```
iout=0;
for i5=0; i5 ¡ n5;i5++ do
  for i2=0; i2 ¡ n2;i2++ do
    for i3=0; i3 ¡ n3; i3++ do
      for i4=0; i4 ¡ n4; i4++,iout+=n1 do
        iin = (i2 + i3 + i4 + i5) * n1 + (i3 + i4 + i5) * n2 + (i4 + i5) * n3 + i5 * n4
        memcpy(&out[iout],&in[iin],n1)
      end for
    end for
  end for
end for
```

in RAM. A problem with this simple approach is the very poor use of input cache lines for small $n1$.

If we can hold $n1 * n2 * n3 * n4$ is memory we can get acceptable performance with either algorithm 1 or slight modification that processes each $i5$ block in turn. We could still use the basic template for large problems by `mmapping` the input and output file but the cache miss problem would be further exacerbated. A better alternative is to introduce two temporary buffers, `tin` and `tout`. These buffers are of size $n1 * n4 * n3 * nmx$, where nmx is chosen so that the combined size of `tin` and `tout` does not exceed DRAM*. The buffered algorithm then takes the following form. The larger nmx, the better the performance. Figure 1 shows that even a RAM-based system benefits from the buffered approach. Note how we can gain a performance advantage of greater than six with larger $n1 * nbuf$ sizes.

Figure 1: The number of elements per second vs $log(nbuf * n1)$ that can be read using a completely in-core solution. The problem size, using the generic template, is $(1, 500, 72, 131072, 1)$. Note how we can achieve a factor of six improvement by better cache line use. [**NR**]

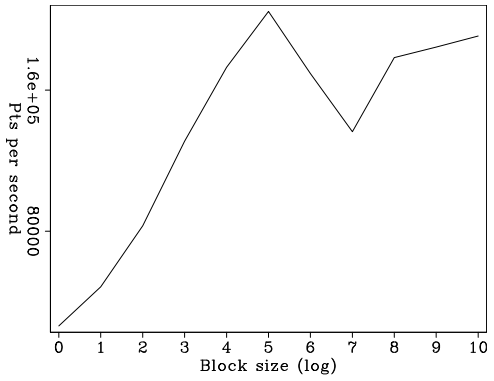

Figure 2 compares the performance of EcoRAM versus RAM. Note the similarity of the two curves. The '*' in the figure shows the comparable disk approach. The disk

*This approach only works if $n1 * n4 * n3 * 2$ does not exceed DRAM size.

Algorithm 2 Simple transpose

```
for i5=0; i5 ¡ n5;i5++ do
  i2 = 0
  while i2¡ n2 do
    nbuf = min(n2 − i2, nmx)
    read tin
    iout=0;
    for i2=0; i2 ¡ nbuf;i2++ do
      for i3=0; i3 ¡ n3; i3++ do
        for i4=0; i4 ¡ n4; i4++,iout+=n1 do
          iin = (i2 + i3 + i4) ∗ n1 + (i3 + i4) ∗ n2 + (i4) ∗ n3
          memcpy(&out[iout],&in[iin],n1)
        end for
      end for
    end for
    i2 = i2 + nbuf
  end while
end for
```

approach is $\frac{1}{10}$th speed of the slowest EcoRAM result and $\frac{1}{200}$th the optimal buffer choice.

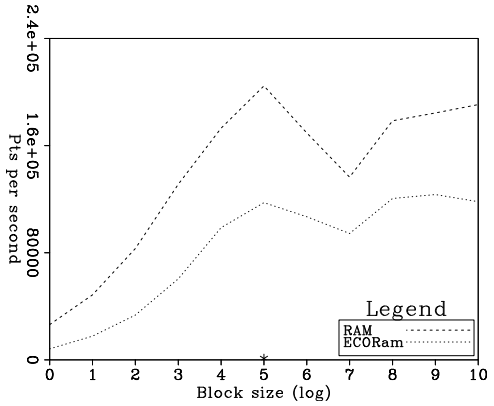

Figure 2: The number of elements per second vs. $log(nbuf * n1)$. Note the '*' indicating the disk IO performance. The problem size, using the generic template, is $(1, 500, 72, 131072, 1)$. [**NR**]

As a final test, we transposed a float dataset of size $(672, 216, 72, 1, 2000)$ switching axes 1,2 with axis 5. Using the buffered approach of algorithm 2, a conventional disk took 1293 minutes (using an intermediate buffer size of 1 GB) while the same dataset took 22 minutes using EcoRAM , a $60x$ performance improvement.

OTHER APPLICATIONS

The characteristics of EcoRAM seem best suited for large databases where reads dominate. In addition to the visualization and transpose algorithm described above,

data sorting is an obvious area that could benefit from EcoRAM . Another interesting application is the ability to visualize pre-stack volumes without any preprocessing. As processing continues to outstrip IO, more applications will begin to benefit significantly from EcoRAM . Interferometry, Hessian-based inversions, and even 3-D surface related multiple attenuation, are, or soon will be, read-dominant. One can envision a future where raw data sits on an EcoRAM -like system and all processing steps are carried out on the fly, eliminating the need for the multitude of intermediate data volumes that are now common.

CONCLUSIONS

Read-dominant applications such as visualization and data reordering are IO bound at large sizes. EcoRAM , with its low latency and high bandwidth (particularly compared to disk at small read sizes) proves to be one to two orders faster than conventional disk approaches. These speed improvements can be achieved with minimal programming changes.

ACKNOWLEDGMENTS

I would like to thank Lou Gagliardi of Spansion for useful discussions on how to optimize application performance.

REFERENCES

Baysal, E., D. D. Kosloff, and J. W. C. Sherwood, 1983, Reverse time migration: Geophysics, **48**, 1514–1524.

Clapp, R. G., D. M. Chen, and S. Luo, 2008, Hypercube viewer: 2008, **134**, 179–192.

Clapp, R. G. and N. Nagales, 2008, Hyercube viewer: New displays and new datatypes: 2008, **136**, 125–130.