MULTICORE DISTRIBUTED DICTIONARY LEARNING: A MICROARRAY GENE EXPRESSION BICLUSTERING CASE STUDY

Stephen Laide, John McAllister

Institute of Electronics, Communications and Information Technology (ECIT),
Queens University Belfast, UK

ABSTRACT

The increasing pervasion and scale of machine learning technologies is posing fundamental challenges for their realisation. In the main, current algorithms are centralised, with a large number of processing agents, distributed across parallel processing resources, accessing a single, very large data object. This creates bottlenecks as a result of limited memory access rates. Distributed learning has the potential to resolve this problem by employing networks of co-operating agents each operating on subsets of the data, but as yet their suitability for realisation on parallel architectures such as multicore are unknown. This paper presents the results of a case study deploying distributed dictionary learning for microarray gene expression bi-clustering on a 16-core Epiphany multicore. It shows that distributed learning approaches can enable near-linear speed-up with the number of processing resources and, via the use of DMA-based communication, a 50% increase in throughput can be enabled.

Index Terms— Multicore, Machine Learning, Distributed, Biclustering, Microarray Gene Expression

1. INTRODUCTION

Machine learning (ML) and artificial intelligence technologies increasingly pervade intelligent computing systems. As the amount of data these process increases, traditional ML algorithms, which require access to the entire dataset at all times, quickly and increasingly encounter the memory wall - fundamental limits on throughput and latency imposed by multiple parallel processing elements attempting to access a single, centralized data resource concurrently [1]. As the scale of ML problems, algorithms and processing technologies increases, it is imperative that they do so in a way that leads to high performance on parallel processing platforms.

Distributed learning algorithms forego global visibility, undertaking learning via a network of locally-connected *agents*, each of which has visibility of only a subset of the data [2, 3]. This approach appears highly promising with respect to the highly-parallel multicore [4, 6, 7], Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA) platforms which are emerging to power these systems. However, to the best of the authors' knowledge, there has been no experimental study of distributed ML on multicore platforms.

This paper studies deployments of a sample ML workload - dictionary learning [5] for microarray gene expression bi-clustering - on a highly parallel processing platform. This case study is taken from [3]. The algorithm realised is both decentralised and online, in that the network is exposed to successive input samples in turn, in a streaming manner. Specifically, the following contributions are made:

1. A realisation of distributed dictioinary learning for micorarray gene expression bi-clustering on 16-core Epiphany-III multicore is presented.
2. It is shown how, via batch processing of input samples, near-linear speedup of throughput with the number of processing resources can be achieved.
3. It is shown how, by selective employment of Direct Memory Access (DMA) as compared to simple nearest-neighbour point-to-point communication, throughput increases of almost 50% may be achieved.

2. BACKGROUND & CONTEXT

Realising global dictionary learning using a network of N agents means resolving:

$$\begin{aligned} &\min_{W} \mathbb{E}\left[f\left(\mathbf{x}_t - W_{\mathbf{y}_t^o}\right) + h_y\left(\mathbf{y}_t^o\right)\right] + h_W\left(W\right) \\ &\text{s.t.} \qquad \mathbf{W} \in \mathcal{W} \end{aligned} \tag{1}$$

Each node k will possess a subset W_k of the full dictionary W. Each input sample $\mathbf{x}_t$ is received by a subset of the network, with each agent calculating an optimal sub-vector of the coefficients $\mathbf{y}_{k,t}^o$ relative to its own sub-dictionary. In [3] it is shown how distributed dictionary learning successfully approximates the classification of microarray gene expression data for the detection of lung cancer [8]. Specifically, for 56 test subjects each providing 12,625 gene expression levels a data matrix $X \in \mathbb{R}^{56\times12,625}$ may be decomposed according to:

$$X = \sum_{k=1}^{N} \mathbf{w}_k \mathbf{y}_k^T \tag{2}$$

Where $\mathbf{w}_k \in \mathbb{R}^{56\times 1}$ and $\mathbf{y}_k \in \mathbb{R}^{12,625\times 1}$. Each $\mathbf{w}_k$ represents a column, or an *atom* of the dictionary W and it is shown how the bi-clustering problem can be suitably resolved using $N = 3$, i.e. $W \in \mathbb{R}^{56\times 3}$. It does so by employing distributed dictionary learning with two distinct operating phases, illustrated in Fig. 1:

- **Sparse Coding/Inference**: sparse approximation of the input data using the existing dictionary
- **Dictionary Update**: gradient-descent based dictionary update

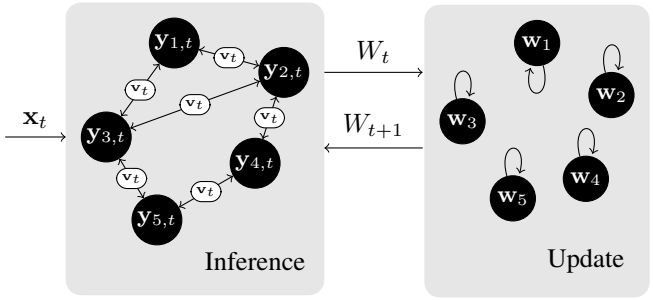


Fig. 1: Distributed Dictionary Learning

As shown, the learning prcess is undertaken by a network of agents, each of which estimates sub-components of $\mathbf{y}$ and W and which share error information, specifically an error vector $\mathbf{v}_t$, via diffusion to allow the network to converge to a consensus on the optimal value of the error. In this case, each component $\mathbf{w}_k$ of W is a single column. Once the optimal error is identified, each agent uses its local estimates of $\mathbf{v}$ and $\mathbf{y}$ to update its own dictionary via gradient descent with respect to its own dictionary. For each data sample the inference is iterated a number of times to ensure that the error converges toward a global minimum, with error estimates exchanged between nodes once per iteration, equating to multiple exchanges per data sample. Dictionary update only occurs once per data sample. In this case, the number of nodes $N = 3$ and full network connectivity is assumed. The algorithm operates as outlined in Algorithm 1.

It is assumed that $\mathbf{v}_0 = \mathbf{0}_{56}$[1] and $\mathbf{w}$ randomly initialised and projected onto the valid solution space using[2]:

$$\left[\prod_{\mathcal{W}_k}(X)\right]_{:,n} = \begin{cases} [X]_{:,n}, & ||[X]_{:,n}||_2 \leq 1 \\ \frac{[X]_{:,n}}{||[X]_{:,n}||_2} & ||[X]_{:,n}||_2 > 1 \end{cases} \tag{3}$$

[1]Where $\mathbf{k}_n$ denotes an n-element vector, each element of which takes value k

[2]The notation $[A]_{:,n}$ refers to column n of matrix A

where, in this case

$$\mathcal{W}_k = \left\{\mathbf{w} : \left(\mathbf{w} \in \mathbb{R}^{56\times 1}\right) \cap \left(||\mathbf{w}|| \leq 1\right)\right\} \tag{4}$$

Over 2000 iterations, the error vector is derived (lines 2 - 4), with the values of $\mathbf{y}$ determined (line 6) according to the soft thresholding operator

$$\left[\mathcal{T}_\alpha(\mathbf{x})\right]_n \triangleq \left(|[\mathbf{x}]_n| - \alpha\right) + \text{sgn}\left([\mathbf{x}]_n\right) \tag{5}$$

Throughout, $\mu_\mathbf{v} = 0.01$, $\mu_\mathbf{w} = 5 \times 10^{-3}$, $\delta = 0.01$, $\gamma = 0.5$, $\beta = 0.01$ and $A = \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T$. Note specifically the two phases of operation: multi-iteration inference (lines 2-4), which includes update of the error vector according to that experienced by the other nodes in the network and the dictionary update (line 7).

Algorithm 1 Node Infer-Update Program

1: **procedure** INFER-UPDATE($\mathbf{x}_t, \mathbf{v}_0, \mathbf{w}_t$)
2: **for** $i \leftarrow \{1, ..., 2000\}$ **do**
$\psi_i \leftarrow \mathbf{v}_{i-1} - \mu_\mathbf{v}\frac{1}{N}\left(\mathbf{v}_{i-1} - \mathbf{x}_t\right)$
3: $- \frac{\mu_\mathbf{v}}{\delta}\mathcal{T}_\gamma\left(\mathbf{w}_{t-1}^T\mathbf{v}_{i-1}\right)\mathbf{w}_{t-1}$
4: $\mathbf{v}_i \leftarrow \sum a_{lk}\psi_{l,i}$
5: $\mathbf{v} \leftarrow \mathbf{v}_i$
6: $\mathbf{y} \leftarrow \frac{1}{\delta}\mathcal{T}_\gamma\left(\mathbf{w}_{t-1}^T\mathbf{v}\right)$
7: $\mathbf{w}_t \leftarrow \prod_{||\mathbf{w}||\leq 1}\left\{\mathcal{T}_\beta\left(\mathbf{w}_{t-1} + \mu_\mathbf{w}\mathbf{v}\mathbf{y}^T\right)\right\}$

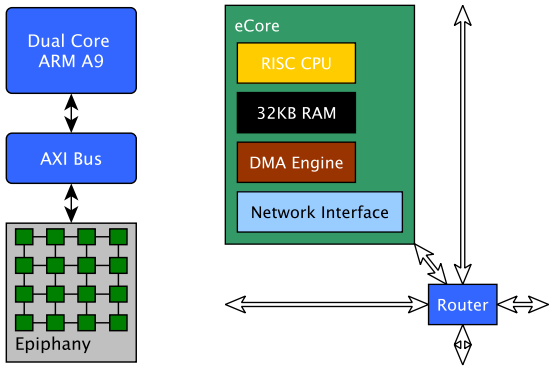
3. DISTRIBUTION

3.1. Distributed Implementation

In this paper, we analyse the performance of the distributed learning approach on the Adapteva Parallela III Microserver [9]. The architecture of this platform is illustrated in Fig. 2. The Parallela consists of a host Dual-Core ARM A9 and a 16-core Epiphany III many-core coprocessor [10] communicating via an AXI interconnect and 32 MB DRAM memory, which is the only external communication point with the Epiphany. The structure of each element of the Epiphany is shown in Fig. 2b.

Each Epiphany node or *eCore* consists of a superscalar RISC CPU, 32 KB of local scratchpad memory, an interface to the *eMesh* on-chip communications network and a DMA engine. Inter-core communication can be via one of two mechanisms: the DMA, or direct load/store interface to the memory space of each eCores neighbours to the north, south, east and west [11].

A number of aspects of the eCore are notable. Firstly, the amount of local memory is highly limited, meaning that highly localised learning is required if ML on large data objects is to be achieved without saturating the shared RAM resource. Secondly, a series of compromises have been made



(a) Parallela (b) Element Structure

Fig. 2: Multicore Architecture

in the design of the eCore which may influence its computational performance, specifically, the absence of native floating-point divider components, with division instead emulated in software.

The distributed implementation separates the dictionary W into three atoms, one per column, with the learning process associated with each atom undertaken by a single eCore. The workgroup is composed of four eCores, a single *master* node used to handle the streaming of input data to each of the agents, of which there are multiple referred to as the *network*. This arrangement is illustrated in Fig. 3, with master and network nodes denoted by *M* and *N* respectively.

The distributed implementation operates as follows:

- **Initialisation**: The master node draws $\mathbf{x}_t$ from shared DRAM, all network nodes initialise their local $\mathbf{w}$ and $\mathbf{v}$ vectors, and enter a low-power standby state until woken by the master.
- **Input**: Once all network nodes have initialised, the master writes $\mathbf{x}_t$ to the local memory of each network node and awakes each.
- **Iteration/Update**: The network nodes perform inference, iterating their local error vectors $\mathbf{v}$ and transferring these with their neighbours until convergence. Once complete, dictionary update is undertaken. Simultaneously the master node loads the next input and enters an idle state awaiting the network to complete.
- **Termination**: Once all network nodes have completed, they notify the master and enter a low-power state awaiting the next input sample from the master.

Hence the streaming implementation performs a single iteration of initialisation, followed by a potentially infinite number of iterations of *Input-Iteration-Termination*.

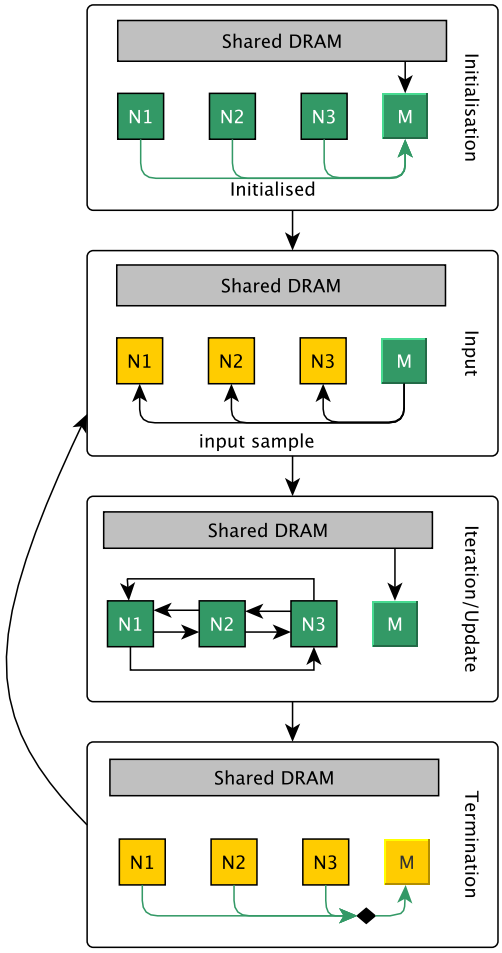


Fig. 3: Distributed Operation Data Exchange

3.2. Batch Operation

The batch-distributed realisation builds on the distributed version, executing the estimation steps for multiple input samples concurrently. In this case the dictionary is again subdivided into three atoms but multiple input samples are processed in batch mode. In this case, a $(4, 3)$ workgroup is used, with element (i, j) hosting atom $\mathbf{w}_j$. Specifically - each row of the workgroup processes atoms from a single sub-group as previously (Fig. 3), but in this case four rows are used to concurrently realise four iterations. This requires local variables to be be exchanged down the columns of the workgroup as the iterations proceed (Fig. 4).

3.3. Performance & Cost

The execution times and throughput for the distributed and batch realisations are illustrated in Fig. 5. As shown, the distributed implementation obtains a throughput of 8.31 Samples/s, with the batch realisation obtaining approximately 20

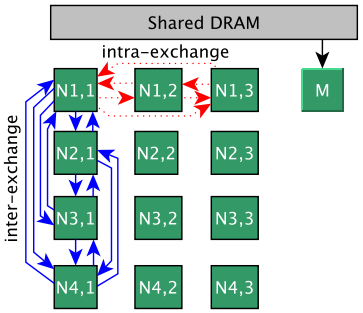


Fig. 4: Batch Operation Data Exchange

Samples/s. It is worth recalling that, in fact, each sample represents 2000 sparse approximation iterations, each processing 56-element vectors during inference and that actual processing rates may be more accurately represented as 8.96×10^5 and 2.24×10^6 Iterations/second.

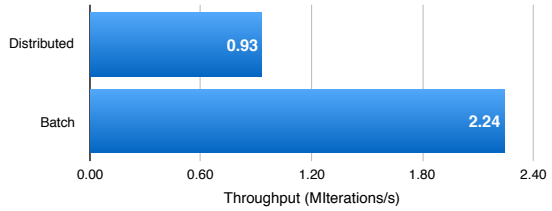


Fig. 5: Parallela Performance Metrics

Whilst the processing rates are substantial, the scaling of performance with the number of processors is somewhat underwhelming - specifically, despite increasing the number of processors by a factor four between the distributed and batch realisations, performance only scales by a factor of 2.4. This is likely due to the extra communication overhead associated with invoking inter-processor communication across iterations, as well as within each. Section 4 investigates the effect of alternative communication approaches on this scalability.

4. COMMUNICATION OPTIMISATION

As outlined in Section 3, the Epiphany supports two methods of communication between eCores - an eCore can directly access the memory of its adjacent north, south, east and west using load-store instructions; alternatively, DMA transfer over the eMesh is enabled. It is reported [11] that, for message sizes below 600 bytes, that direct on-chip write incurs a lower latency than DMA, due to initialisation and setup overhead associated with the DMA engines. Note that the transfers between eCores in this application are below this threshold: the inputs transferred from the master to each network node are 56-element floating-point vectors (224 bytes), with similar-sized objects transferred between each pair of network nodes for error propagation. Despite this, the impact on real-time performance as a result of employing either is variable, as illustrated in Fig. 6.

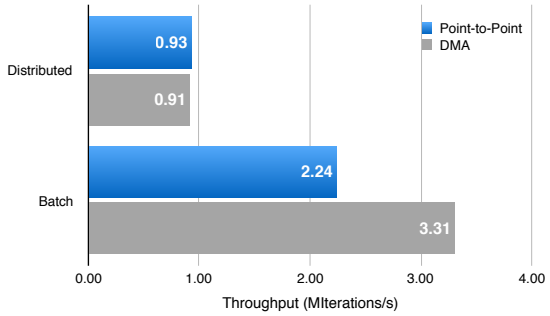


Fig. 6: Communication Performance Metrics

As shown, the distributed realisation experiences a marginally reduced performance when DMA is employed for inter-eCore communication, with throughput reduced by around 1.8% in the case where DMA is employed. However, the batch realisation experiences a considerable increase in performance - 47.6% - when DMA is employed rather than point-to-point communication. The throughput on employing DMA stands at 29.54 Samples/s (3.31×10^6 Iterations/s). It is noticeable that when DMA is employed for four distributed workgroups in the batch realisation, peak throughput is increased by a factor of 3.6 relative to a single distributed workgroup. It appears that DMA-based communication offers superior performance scalability than point-to-point in applications of this scale.

5. CONCLUSION

As machine learning and the data sets upon which they operate increase in scale, so ever-higher performing realisations will be required for real-time data processing. The highly parallel processing architectures which will enable these, however, have to work under memory access-rate and communication constraints which mean that not all parts of the ML algorithms have visibility of the entire dataset. Distributed ML approaches mitigate against the impact of these constraints, but are largely currently of theoretical interest.

This paper benchmarks the performance, scalability and suitability of the Adapteva Epiphany, a combination of 16 lightweight superscalar processors, when realising a distributed dictionary learning problem. It shows that, by employing a distributed realisation of the learning algorithm for each input sample and processing inputs in batches, performance scales near-linearly with the number of workgroups employed and that, by employ DMA-based rather than direct point-to-point communication between neighbours, throughput may be increased by almost 50%. This work can act as a benchmark against which other efforts in the emerging area of multicore distributed learning evolve.

6. REFERENCES

[1] H. Yu, Y. Wang, S. Chen, W. Fei, C. Weng, J. Zhao, and Z. Wei, "Energy Efficient In-Memory Machine Learning for Data Intensive Image-processing by Non-Volatile Domain-Wall Memory," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2014, pp. 191–196.

[2] A. H. Sayed, "Adaptive networks," *Proceedings of the IEEE*, vol. 102, no. 4, pp. 460–497, April 2014.

[3] J. Chen, Z. J. Towfic, and A. H. Sayed, "Dictionary Learning Over Distributed Models," *IEEE Transactions on Signal Processing*, vol. 63, no. 4, pp. 1001–1016, Feb 2015.

[4] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan 2008.

[5] I. Tosic and P. Frossard, "Dictionary Learning," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 27–38, March 2011.

[6] F. D. Igual, M. Ali, A. Friedmann, E. Stotzer, T. Wentz, and R. A. van de Geijn, "Unleashing the High Performance and Low Power of Multi-core DSPs for General-Purpose HPC," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[7] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2008, pp. 88–598.

[8] Mihee Lee, Haipeng Shen, Jianhua Z. Huang, and J. S. Marron, "Biclustering via Sparse Singular Value Decomposition," *Biometrics*, vol. 66, no. 4, pp. 1087–1095, 2010.

[9] *Parallela 1.x Reference Manual*, 2013.

[10] Adapteva, Inc., *Epiphany Architecture Reference*, 2013.

[11] A. Varghese, B. Edwards, G. Mitra, and A. P. Rendell, "Programming the Adapteva Epiphany 64-core Network-on-chip Coprocessor," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, pp. 984–992.