Solving billions of small multiple linear algebra problems

Detecting genetic interactions in multi-parental breeding populations.

Joshua C Bowden[(1)], B. Emma Huang[(2)], John A. Taylor[(2)]

[(1)]CSIRO IM&T SCIENTIFIC COMPUTING, [(2)]CSIRO DIGITAL PRODUCTIVITY FLAGSHIP

www.csiro.au

The ability of modern GPUs to operate in parallel on thousands of small matrices has huge potential for genetic analysis in agriculture. We explored this potential in a pipeline that incorporates batched versions of cuBLAS operations, matrix multiplication, LU decomposition and inversion and analysis of variance for comparison of model efficacy. This pipeline can be applied to detect genetic interactions in multi-parental populations, an analysis which would otherwise be extremely time-consuming.

Introduction

Genetic information is now increasingly available through modern sequencing methods and this data combined with specific genetic breeding trials such as the Multiparent advanced generation inter-cross (MAGIC) results in the input data for the studies. An R based package (mpMap)[1] has been developed for analysis of these studies, including detection of associations between traits and genetic markers. The package currently integrates R, C++, and CUDA code. This work extends the package using a pipeline that can solve the many small linear models needed to look at the influence of pairwise marker interactions.

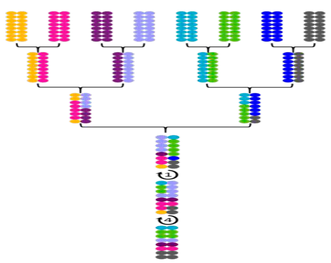

Figure 1: Diagram of the breeding design used to create MAGIC 8-way intercross in wheat. Genomes of inbred parents are combined through multiple generations of intercrossing and selfing to produce homozygous offspring with diverse genotypes and phenotypes.

$$\mathbf{X}^{-1} \cdot \mathbf{C}^{T} \cdot \mathbf{y} = \hat{\mathbf{b}}$$

$$\mathbf{X} = \mathbf{C} \cdot \mathbf{C}^{T}$$

Figure 2: The linear regression equation being calculated. C is the input probability cross term matrix, y is the response being modelled and b are the calculated model parameters that best predict the response from the given C data.

Materials/Methods

The set of transformations that are involved in the linear modelling problem have been implemented using the batched BLAS functions from the cuBLAS library combined with some CUDA based programming. The hybrid cuBLAS and CUDA kernels are compared to Intel MKL BLAS/Lapack code running on Xeon and Xeon Phi.

Due to the independence of each model computation, an algorithm was devised where processes are launched in an ad-hoc fashion (for example using a PBS array job), and rely on a file based (HDF5) reference system to coordinate between launched processes. The HDF5 file stores input and output data and a set of flags that indicate if a marker interaction needs to be computed, is being computed or has been computed. Access to the file is serialised using a file based locking procedure.

Possible linear correlations between input data columns requires that the linear algebra routines to report errors such as singularity and failures to invert. In event of a failure a more robust (and slower) routine (SVD) will be invoked.

Hardware	Number per node	Number of nodes	Software
Intel Xeon (Sandybridge) 8 cores @ 2.0 or 2.6 GHz	2		OpenMP + MKL Lapack DPOSV()
Intel Xeon Phi 7120P	2	16	OpenMP in offload mode + MKL Lapack DPOSV()
NVidia Kepler K20M GPUs	3	128	CUDA + cuBLAS Batched functions Dgemm Dgetrf() and Dgetri()

Table 1: Available hardware on the CSIRO accelerator cluster 'Bragg-I'. 2 CPU sockets are housed with either 2 Phi cards or 3 Kepler GPUs.

Results

Initial testing of an OpenCL based $C \cdot C^T$ implementation showed that occupancy was limited through a lack of shared memory used in the developed algorithm and this avenue was discontinued.

GPU function	% Runtime
CreateCrossTerms	3.7
Dgemm [$C \cdot C^T$]	34.5
Dgetrf	19
Dgetri	22
Dgemm [$C^T \cdot y$]	7.5
Dgemm [$X^{-1} \cdot [C^T \cdot y]$]	1.5
Dgemm [$C^T \cdot b$]	7
daxpy_batched	1.5
ddot_batched	0.2

Table 2: Profile information captured with Nsight 6.0 for batched computations of 10,240 models, each model having 64 terms and 300 individuals.

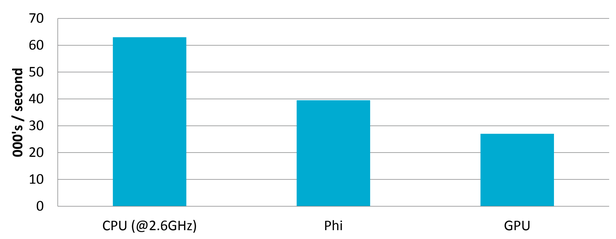

Figure 3: Throughput of models achieved on different hardware.. Each model tests for interaction between two markers in an 8-parent population and results in model matrices of 64 x number of individuals (= 300). 10,000-15,000 models per batch were then performed to get the throughput. CPU results are for all 16 cores available per node.

Profiling results of a CUDA version using batched cuBLAS functionality are shown in Table 2 and obtained from the Nsight Eclipse IDE. It is seen that the batched DGEMM functions takes over half the runtime and achieves ~200GFLOPs performance for the initial $C \cdot C^T$ multiplication. The inversion calculations take the bulk of the remaining processing time (~40%)

CUDA interoperability with cuBLAS functions streamlined the use of GPUs for this application.

Although using essentially identical code as written for the CPU, programming of the Intel Phi in offload mode had a number of unusual features that came from the dual object code production. Code regions required conditional compilation to prevent unusual behaviour such as multiple printf() outputs and to incorporate use of the HDF5 library.

Serialisation of access to the HDF5 control file was found to not negatively impact performance even when up to 30 separate processes were used to compute the set of solutions.

Conclusions

The use of accelerators has not proven to be as advantageous as we would like in terms of throughput of problems. The available 16 CPU cores obtained about 2× throughput over a single Kepler K20 GPU and 1.6× over Intel Phi 7120P. However, combining all resources in a node more than doubles the throughput of CPU's alone.

The cache based architecture of the CPU and Phi accelerator excels at the rapid computation of many small linear algebra functions due to the capacity to reuse recently used data from fast cache memory.

Performance of GPUs on this workload could possibly be improved. Enhancements to the batched cuBLAS Dgemm performance over the ~20% theoretical peak achieved would be desirable. Testing of the of the batched DGELS procedure available in the CUDA 6.5 is being undertaken. Integration of a batched Cholesky method for the GPU is also being looked into[2]. The shuffle instruction which is a feature of Kepler class GPUs is being investigated for efficient computation of the initial $C \cdot C^T$ matrix formation.

Interfacing the cuBLAS functionality with the OpenCL programming would a welcome addition for enhanced GPU programmability.

The resulting system should enable a comparison of a full sized problem of 60,000 markers and 500 individuals to be completed in around 1 hr using 5 nodes of the accelerator cluster.

FOR FURTHER INFORMATION

Josh Bowden

e josh.bowden@csiro.au

w www.csiro.au/ IM&T Science Software

REFERENCES

[1] Huang BE and George AW. 2011. R/mpMap: A computational platform for the genetic analysis of multi-parent recombinant inbred lines. Bioinformatics 27:727-729

[2] Dong T., Haidar A., Tomov S., and Dongarra J. 2014. A Fast Batched Cholesky Factorization on a GPU. 43rd International Conference on Parallel Processing, 432-440

ACKNOWLEDGEMENTS

Funding is gratefully acknowledged through the CSIRO Agricultural Flagship, the Australian Research Council and CSIRO eResearch Collaboration Project funding