Auto-testing efficiency of signal processing algorithms on embedded Graphics Processing Units (GPUs)

UPPSALA UNIVERSITET

Project in Scientific Computing 2018

Olle Frisberg

Gunnlaugur Geirsson

Vilhelm Urpi Hedbjörk

Jimmy Pettersson
Supervisor
High Performance Consulting

Maya Neytcheva
Course coordinator
Uppsala University

Conclusions

- Metrics can be deceptive: One algorithm implementation may get away with fewer operations than another, resulting in lower throughput or FLOPs per watt but a shorter computational time overall.
- In general: optimized code → greater utilization → greater power efficiency.
- The Jetson TX2 outperformed the TX1 in every benchmark
- Utilizing the power saving modes on the TX2 proved to be more energy efficient than running at full power.

Statement of work

Develop a framework to auto-test the performance of benchmarking algorithms on the Jetson TX1 and TX2 modules. Implement QR factorization, bicubic interpolation and pattern matching in CUDA C++ and evaluate the GPU performance with these using the framework.

Procedure

Theoretical FLOPs and global loads/stores were counted “by hand” and compared to the Nvidia visual profiler. The Jetson modules supply live datasheets with information about current power consumption.

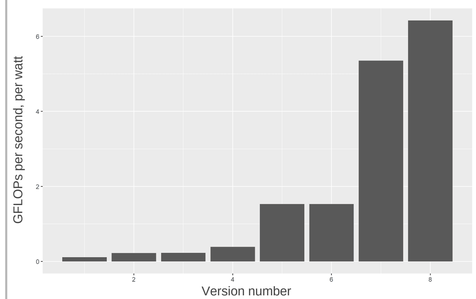
QR Factorization

Algorithm

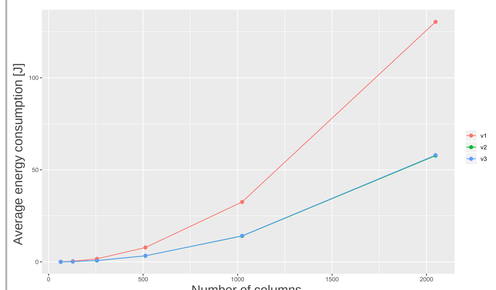
Decompose matrix A into an orthogonal matrix Q and an upper triangular matrix R so that A = QR.

Implementation

Every column vector of A is processed by a block of CUDA threads, with vector operations parallelized within the block. Some versions are designed for 64x64 matrices using only one block. This enables the use of low-level functions. Combined with smarter work partitioning and memory accesses, this yields greatly improved efficiency.



Floating point operation throughput efficiency for different implementations on 64x64 matrices. Versions 4-8 are specifically designed for 64x64 matrices.



Total energy consumption when varying the number of columns.

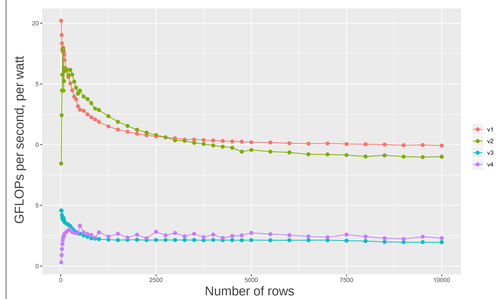
Bicubic Interpolation

Algorithm

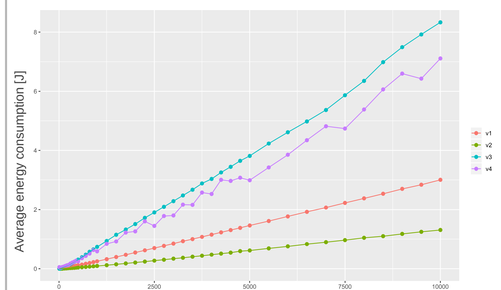
Interpolate unknown data points using cardinal cubic Hermite and B-splines.

Implementation

A CUDA thread can either represent a point in the scaled up or original data, the latter being more computationally heavy but having faster memory access and fewer redundant calculations. A third version transposes the working matrix for more efficient memory access, and the fourth splits the input data into smaller subsets.



Floating point operation energy efficiency for different algorithm implementations when varying the number of rows.



Average energy consumption for different algorithm implementations when varying the number of rows.

Pattern Matching

Algorithm

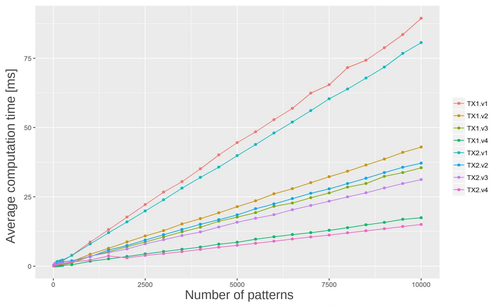
Used to compare a vector (often containing radar values) with existing templates. This is done by finding the best shift (offset) and the best corresponding amplitude scaling.

Implementation

In the first version each block dimension handles a pattern, shift and vector element respectively. Version two utilizes shared memory and versions three and four were both limited to vector sizes of 32 in order to make use of lower level CUDA warp functions.

Energy efficiency for the highest performance pattern matching version with different power modes on the Jetson TX2.

Power mode	Time (ms)	Power (W)	Energy (mJ)
0	7.55	11.3	85.3
1	9.52	7.0	66.6
2	8.18	8.8	72.0
3	8.14	9.3	75.7
4	8.24	7.6	62.6



Average computational time for different implementations on the TX1 and TX2 when varying the number of library patterns.