GPU Implementation of the Lean Algebraic Multi-grid (LAMG) Solver

Satish Kumar
Student
kumars@usi.ch

Prof Dr Rolf Krause
Advisor
rolf.krause@usi.ch

Riva Simone
Assistant
simone.riva@usi.ch

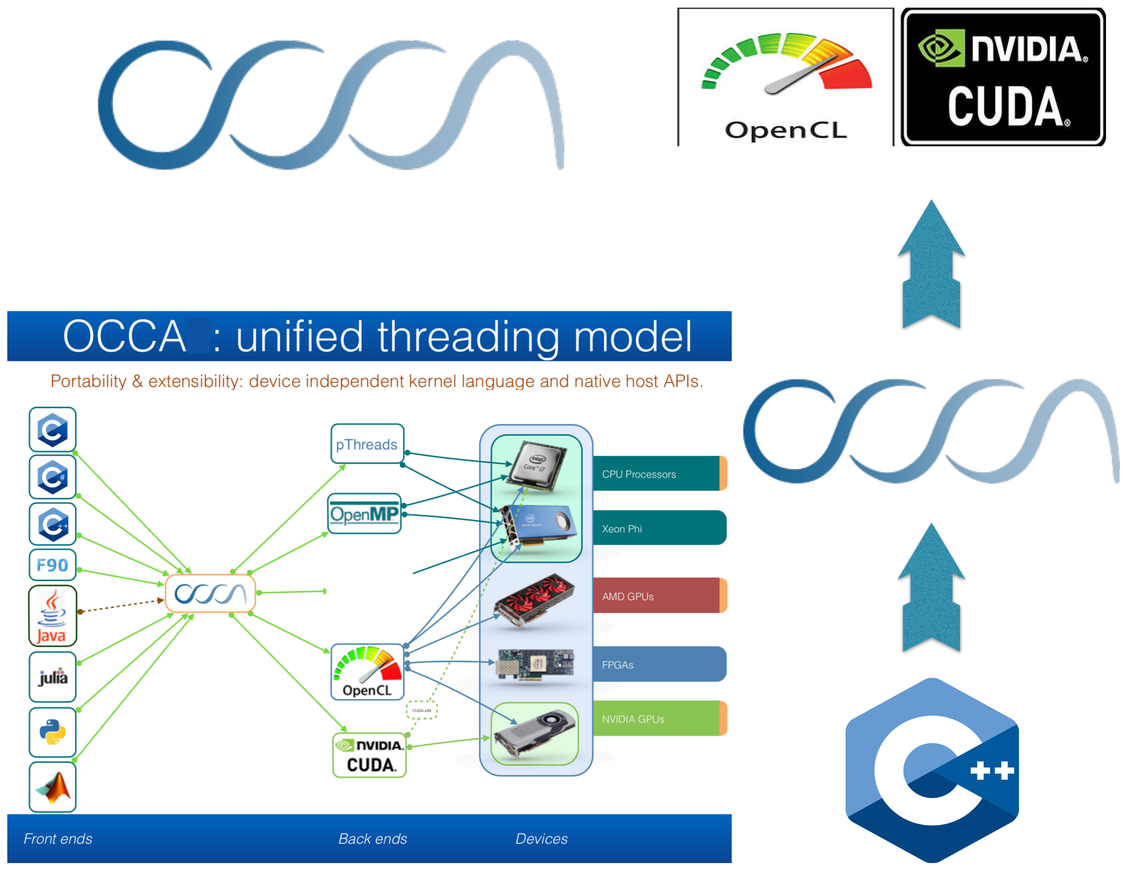
Abstract

A Lean Algebraic Multigrid (LAMG) solver of the linear system Ax=b is presented, where A is a graph Laplacian. LAMG's run time and storage are linear in the number of graph edges. Multigrid (MG) methods in numerical analysis are algorithms for solving differential equations using a hierarchy of discretization. They are an example of a class of techniques called multi-resolution methods, very useful in problems exhibiting multiple scales of behavior.

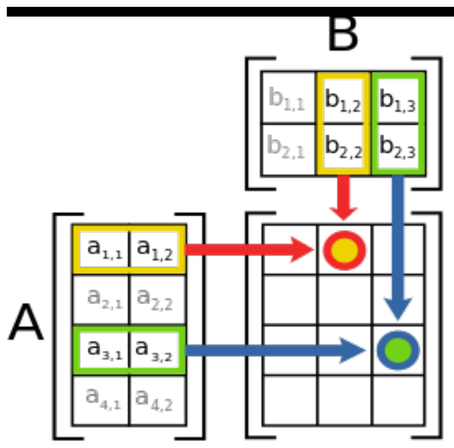
We have worked in OCCA because it is a open-source library that provide a kernel language that generalise the parallel programming paradigms.

In this project, we have created linear algebra API using OCCA and C++. We demonstrated that the individual linear algebra components are faster when using OCCA and GPU as compared to the CPU. We implemented the matrix multiplication. Multiplication between matrix and vector, Sparse matrix in CSR format, sparse matrix in CSR format with vector multiplication, matrices addition, matrices subtraction, dot product between vectors, with reduction and Multi-grid method for dense matrix and sparse matrix. Thus we have analysed and compared the performance between OCCA and CPU.

Tools Uses

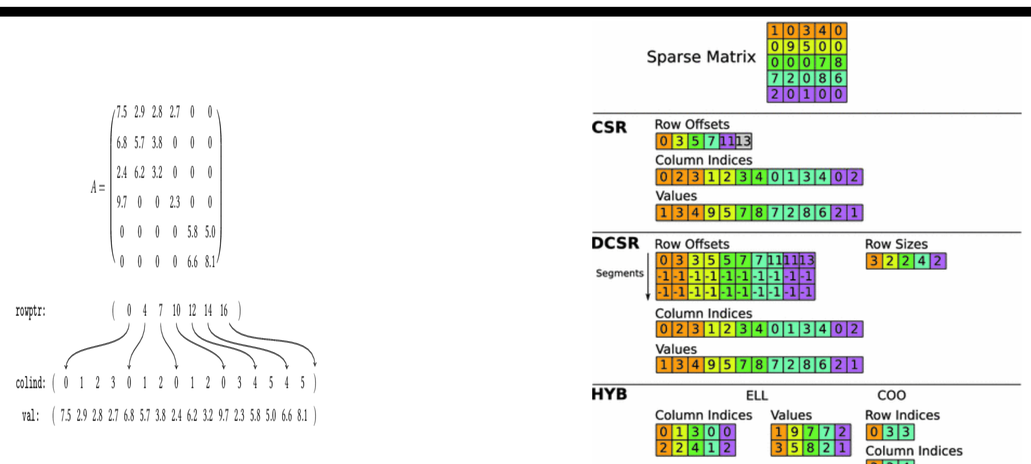


Dense Matrix



$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 5 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 4 \\ 47 \\ 5 \\ 68 \end{bmatrix}$$

Sparse Matrix



CPU Implementation C++

```
for (int i=0; i<row; i++)
{

    for (int j=0; j<colume2nd; j++)
    {
        result[j+i*colume2nd] = 0;
        for(int k = 0; k < colume; k++){
            result[j+i*colume2nd] +=  (x[k+i*colume] *
                y[j+k*colume2nd]);
        }
    }
}
```

OCCA Impementation

```
for(int k =0; k < colume; k++){
    for (int o1 = 0; o1 < row; o1+=16; outer1) {
        for (int o0 = 0; o0 < colume2ndmatrix; o0+=16;outer0) {
            for (int y = o1; y < (o1+16); y++; inner1) {
                for (int x = o0; x < (o0+16); x++;inner0) {
                    if (y < row  && x < colume2ndmatrix) {
                        ab[y*colume2ndmatrix+x] += a[y*colume+k] *
                            b[k*colume2ndmatrix+x];
                    }
                }
            }
        }
    }
}
```

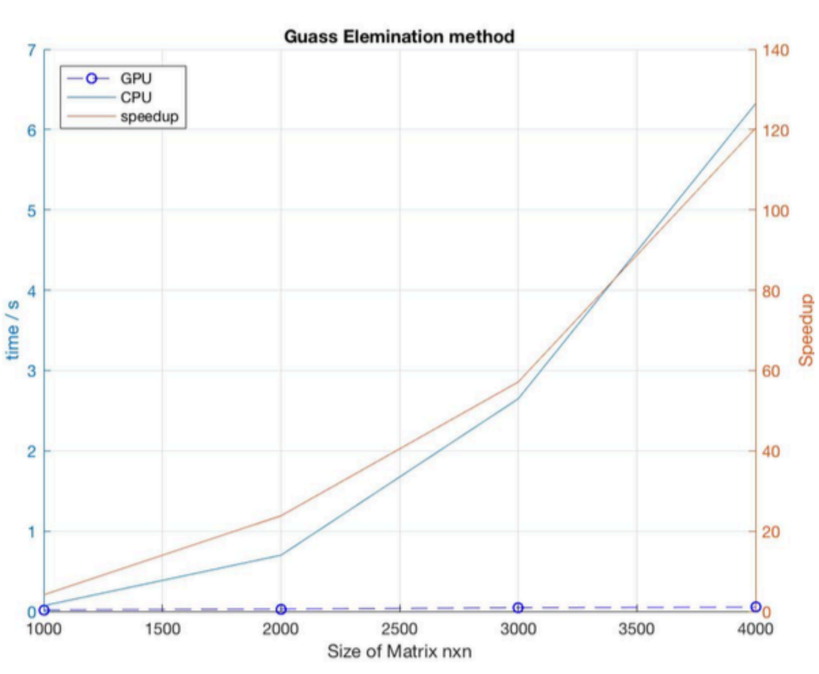
CPU Implementation C++

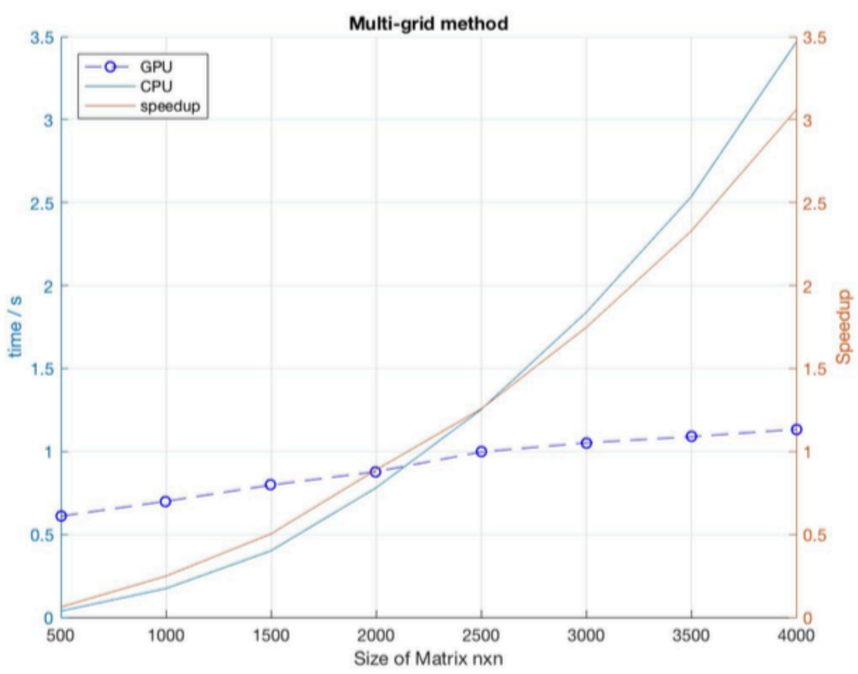
```
int m =0;

for (int j =0; j<size; j++) {
    for (int k = 0; k< size_b; k++) {
        if (a_col_number[j] == b_col_number[k]) {
            int s[1];
            if (checkInArray(a_row[j]*colume2ndmatrix+b_col[k],m,point,s)) {
                ab[s[0]] += a_non_zero[j]*b_non_zero[k];
            }
            else{

                ab[m] = a_non_zero[j]*b_non_zero[k];

                point[m] = a_row[j]*colume2ndmatrix+b_col[k];

                m++;
            }
        }
    }
```

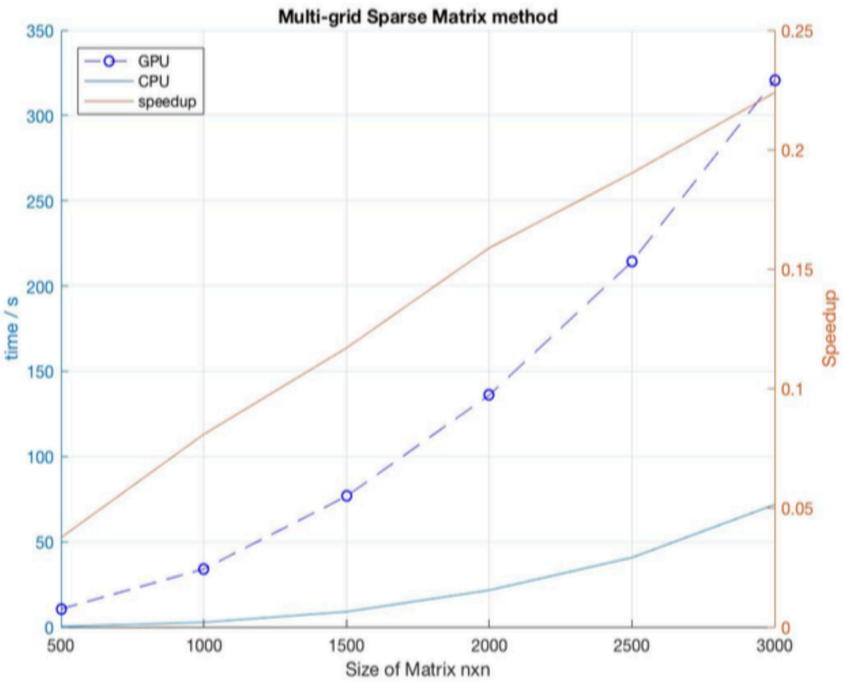
OCCA Impementation

```
for (int j = 0; j < size; j++) {
    if (j<size) {
        for (int o0 = 0; o0 < size_b; o0+=16;outer0) {
            for (int k = o0; k < (o0+16); k++;inner0) {
                if(k < size_b){
                    if (a_col_number[j] == b_col_number[k]) {
                        ab[a_row[j]*colume2ndmatrix+b_col[k]] += a_non_zero[j]*b_non_zero[k];
                    }
                }
            }
        }
    }
}
```

Implementation







Conclusion

In the conclusion i found that GPU is faster than CPU. But if matrix size is small the CPU performs better than GPU. If an application uses very large matrices GPU become more performant than CPU. I think OCCA is good solution for parallel programming with GPU. Because an application can works on all different devices like Nvidia, Intel and Radeon GPUs. So the developers are not constrained to a specific family of products. But at the moment, as described, the big problem of OCCA is that is a work progress. So, It is very interesting language, but now, it is not recommendable for long term or big projects.
OCCA provide a **unified API** for interacting with backend device APIs (e.g. *OpenMP*, *CUDA*, *OpenCL*).
It is easy to understand and use.
But at a moment OCCA is a work in progress: thus all the classical features used in parallel programming are not yet implemented.