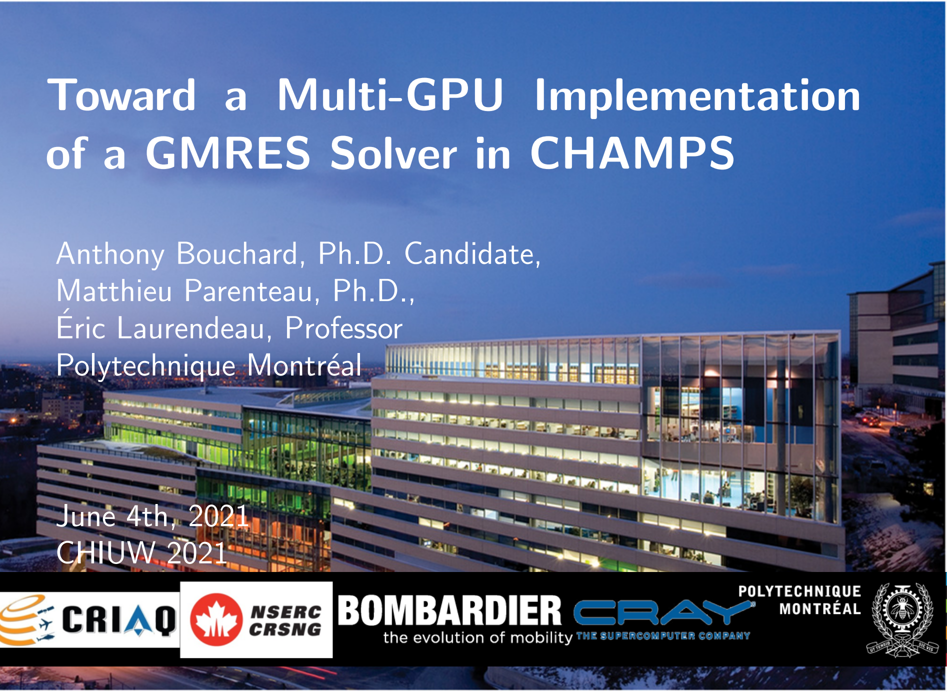


Toward a Multi-GPU Implementation of a GMRES Solver in CHAMPS

Anthony Bouchard, Ph.D. Candidate,
Matthieu Parenteau, Ph.D.,
Éric Laurendeau, Professor
Polytechnique Montréal

June 4th, 2021
CHIUW 2021

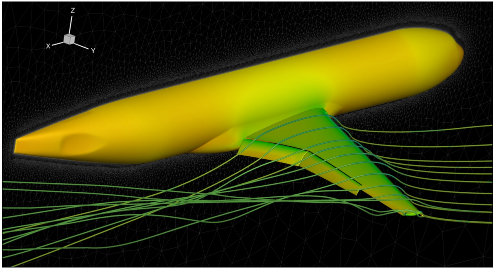
CRIAQ
NSERC CRSNG
BOMBARDIER the evolution of mobility
CRAY THE SUPERCOMPUTER COMPANY
POLYTECHNIQUE MONTRÉAL

Table of Contents

1. Framework
2. GPU Implementation
3. Results
4. Conclusion and Future Work

CHAMPS

- 3D Multi-physics CFD code developed by the research team of Professor Éric Laurendeau
- Written in CHAPEL
- (U)RANS flow solver using an unstructured mesh-based second-order cell centered finite volume method



Computed flow solution with CHAMPS around a high-lift configuration aircraft

RANS Equations

CHAMPS solves the Reynolds-Averaged Navier Stokes equations that can be expressed as:

$$\Omega\frac{\partial \boldsymbol{W}_c}{\partial t} = -\boldsymbol{R}(\boldsymbol{W}_c)$$

where $\boldsymbol{W}_c = [\rho, \rho u, \rho w, \rho w, \rho E]$ is the average state vector of the conservative flow variables of a cell, Ω the cell volume and $\boldsymbol{R}$ the residual vector. Using an inexact Newton iteration, the equation becomes

$$\left[\frac{\Omega}{\Delta t}\mathcal{I} + \Omega\frac{\partial \overline{\boldsymbol{R}}}{\partial \boldsymbol{W}_c}\right]^n \Delta \boldsymbol{W}_c^{n+1} = -\boldsymbol{R}(\boldsymbol{W}_c^n)$$

where Δt is the pseudo time step, $\frac{\partial \overline{\boldsymbol{R}}}{\partial \boldsymbol{W}_c}$ is an approximation of the true Jacobian matrix using a first-order discretization of the dissipation fluxes and the Thin Shear Layer (TSL) approximation of the viscous fluxes and $\Delta \boldsymbol{W}_c^{n+1} = \boldsymbol{W}_c^{n+1} - \boldsymbol{W}_c^n$.

GMRES Solver

- Usual linear solvers used in production CFD codes (like SGS) are not well suited for GPUs
- GMRES is well known for its fast convergence when given a suitable initial solution
- Jacobian-free version:
  - Prevents the need to store the true Jacobian matrix and invert it
  - The Jacobian-vector product is calculated using a finite difference approximation, which requires another evaluation of the residuals that is efficient on GPUs
  - The Jacobian-vector product accurately sees the true Jacobian, enhancing the convergence

GMRES Solver (Cont'd)

Usually, in practice, the GMRES algorithm consists of the Arnoldi-Modified Gram-Schmidt orthogonalization method

```
$w_j = Av_j$ [communication point];
for $i = 1, ..., j$ do
    $h_{i,j} = (w_j, v_i)$ [communication point];
    $w_j = w_j - h_{i,j} v_i$;
end
$h_{j+1,j} = \|w_j\|_2$ [communication point];
$v_{j+1} = w_{j+1}/h_{j+1,j}$;
```

and the least square problem $\left\|\beta e_1 - \bar{H}_m y\right\|_2$, for which givens rotation are applied beforehand to the Hessenberg matrix $h_{i,j}$ to obtain a triangular matrix that can be solved efficiently.

GPU Implementation

```
class DistributedVectorGPU_c : Vector_c
{
    /** Number of local elements **/
    const niLocal_ : int;
    /** Local task id **/
    const localTaskId_ : int;
    /** Domain for the local values **/
    const localDomain_ : domain(1) = {0..#niLocal_};

    proc init(ni : int = 0, niLocal : int = 0, localTaskId : int = 0)
    {
        super.init(ni);

        niLocal_ = niLocal;
        localTaskId_ = localTaskId;
    }

    /**
        Computes the norm
    **/
    override proc norm(param comm : bool = true) : real_t
    {
        var res : real;
        local
        {
            if (updateGPU_)
            {
                this.GPUcopy(0);
            }

            res = cublas_Dot(cublasHandle_, niLocal_, d_values_, d_values_); // CUDA function
        }

        if (comm)
        {
            var val = globalReduction.reduceValues(localTaskId_, [res], REDUCE_OPERATION_t.SUM);
            res = val[0];
        }

        return sqrt(res);
    }

```

GPU Implementation (Cont'd)

```
    /**
        Main procedure to solve the linear system

        :arg lhs: The LHS of the linear system to solve
        :arg rhs: The RHS of the linear system to solve
        :arg buildPreconditioner: If the LHS has changed, the preconditioner is updated
      **/
      proc solve(lhs : BSRmatrix_c, rhs : Vector_c, buildPreconditioner : bool = true) : int
      {
        var totalIteration : int = 0;

        var b_norm : real = rhs.norm();

        x_.reset();

        if (buildPreconditioner) then preconditioner_.buildPreconditioner();

        while (totalIteration < maxIterations_)
        {
          ref Q0 = krylovVectors_[0].Q_;
          Q0.set(rhs);

          if (totalIteration > 0)
          {
            x_.exchangeAuxValues(interfaceCommunicator_);
            lhs.dot(x_, workingArray_);
            Q0.addScale(workingArray_,-1.0);
          }

```

GPU Implementation (Cont'd)

- 1 CUDA thread per dimension of array
- cuBLAS library is used for complex vector operations
- Each zone is assigned to a task and distributed to the available *Locales*
- Each task is assigned to an asynchronous CUDA stream
- Communication is done on the CPU side, copying a buffer array from the GPU to the CPU and back to the GPU once the exchange is done

Multi-GPU Implementation

$$
\begin{array}{r}
\text{task: 0}\left\{\begin{array}{c}\\\end{array}\right.\\
\\
\text{task: 1}\left\{\begin{array}{c}\\\end{array}\right.\\
\\
\text{task: 2}\left\{\begin{array}{c}\\\end{array}\right.
\end{array}
\begin{bmatrix}
\color{red}2 & \color{red}0 & | & \color{blue}4 & \color{blue}0 & | & \color{blue}4 & \color{blue}1\\
\color{red}0 & \color{red}3 & | & \color{blue}5 & \color{blue}6 & | & \color{blue}0 & \color{blue}8\\
- & - & - & - & - & - & - & -\\
\color{blue}0 & \color{blue}2 & | & \color{red}5 & \color{red}0 & | & \color{blue}8 & \color{blue}1\\
\color{blue}1 & \color{blue}0 & | & \color{red}0 & \color{red}1 & | & \color{blue}6 & \color{blue}0\\
- & - & - & - & - & - & - & -\\
\color{blue}0 & \color{blue}4 & | & \color{blue}7 & \color{blue}0 & | & \color{red}1 & \color{red}1\\
\color{blue}1 & \color{blue}2 & | & \color{blue}0 & \color{blue}6 & | & \color{red}0 & \color{red}3
\end{bmatrix}
\times
\begin{Bmatrix}
\color{red}1\\
\color{red}3\\
-\\
\color{red}2\\
\color{red}5\\
-\\
\color{red}4\\
\color{red}9
\end{Bmatrix}
$$

Illustration of a distributed matrix and vector for multiple GPUs (one task per GPU)

Multi-GPU Implementation (Cont'd)

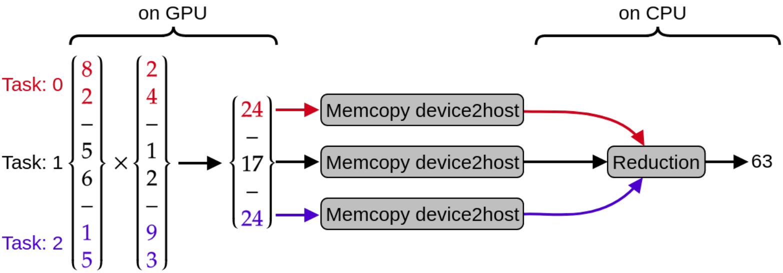
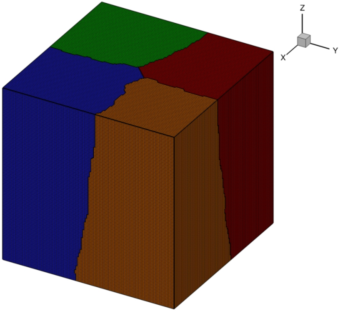


Illustration of a parallel dot product with multiple GPUs (one task per GPU)

Results

Cartesian Grid:

- 2M elements
- 4 zones
- Euler equations solved for 5 unknowns (10M total)
- Cases are run on Béluga

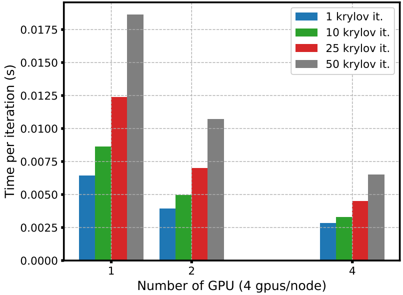


Cluster Hardware

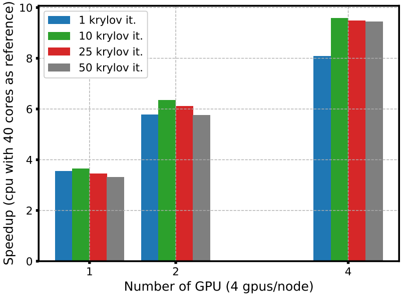
Summary of the CPUs and GPUs on one Beluga node

	Intel Xeon Gold 6148 Skylake	**NVidia V100SXM2**
Number of CPUs/GPUs	2	4
Clocks	2.4 GHz	1.53GHz
Number of cores/SMs per CPU/GPU	20 cores, 40 threads	80 SMs, 32 DP CUDA cores/SM
DP Peak TFLOPS per CPU/GPU	0.704	7.8
Peak memory bandwidth	128 GB/s	900 GB/s
Power per CPU/GPU	150 W	300 W

Speedup

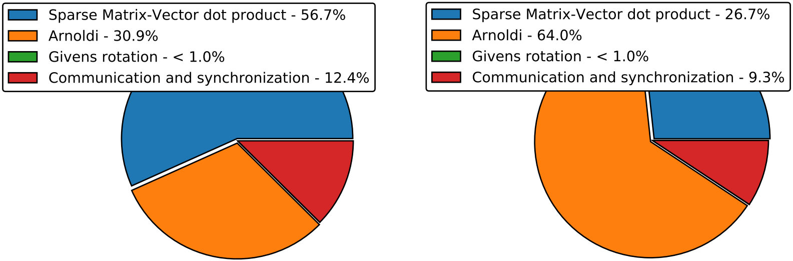


Real time per iteration in seconds for the GMRES algorithm on the GPU for different number of Krylov iterations



Speedup of the GPU version of GMRES compared to 2 CPUs

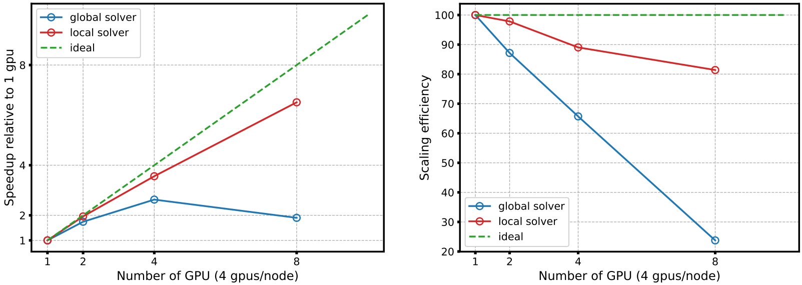
Time Distribution on a Single Node



(a) (b)

Comparison of the time distribution for (a) 10 Krylov iterations and (b) 50 Krylov iterations with 2 CPUs and GPUs

Strong Scaling

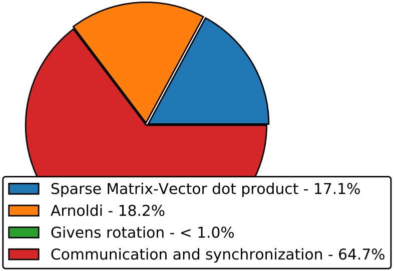


(a)

(b)

Strong scaling analysis of the GPU version of GMRES showing (a) the speedup and (b) the efficiency up to 8 GPUs

Time Distribution over 2 Compute Nodes



Time distribution with 8 GPUs distributed over two compute nodes for 10 Krylov iterations

Conclusion and Future Work

Conclusions:

- The speedup is close to the ratio of memory bandwidth without communication
- The Arnoldi orthogonalization is more and more costly when increasing the number of Krylov iterations
- The use of network atomics prevents efficient scaling on infiniband system

Future Work:

- Work with Cray on a fix for the synchronization problem on infiniband systems
- Port the preconditioners, the JFNK solver and the rest of the flow solver (& turbulence model) over GPU
- Compute end-to-end convergence speedups of the RANS solver

Acknowledgements

- Bombardier Aerospace
- Cray/HPE
- Natural Sciences and Engineering Research Council of Canada (NSERC)
- Consortium for Research and Innovation in Aerospace in Québec (CRIAQ)

Thank you for your attention!

Questions?