Towards an efficient parallel skeleton for generic iterative stencil computations in distributed GPUs

Manuel de Castro, Inmaculada Santamaria-Valenzuela, Sergio Miguel-López, Yuri Torres and Arturo Gonzalez-Escribano

Universidad de Valladolid, Spain

{manuel.castro|sergio.miguel.lopez}@alumnos.uva.es

{msantamaria|yuri.torres|arturo}@infor.uva.es

SuperComputing 2021, November 2021

Universidad de Valladolid

Grupo Trasgo

Universidad de Valladolid

GOBIERNO DE ESPAÑA MINISTERIO DE CIENCIA E INNOVACIÓN

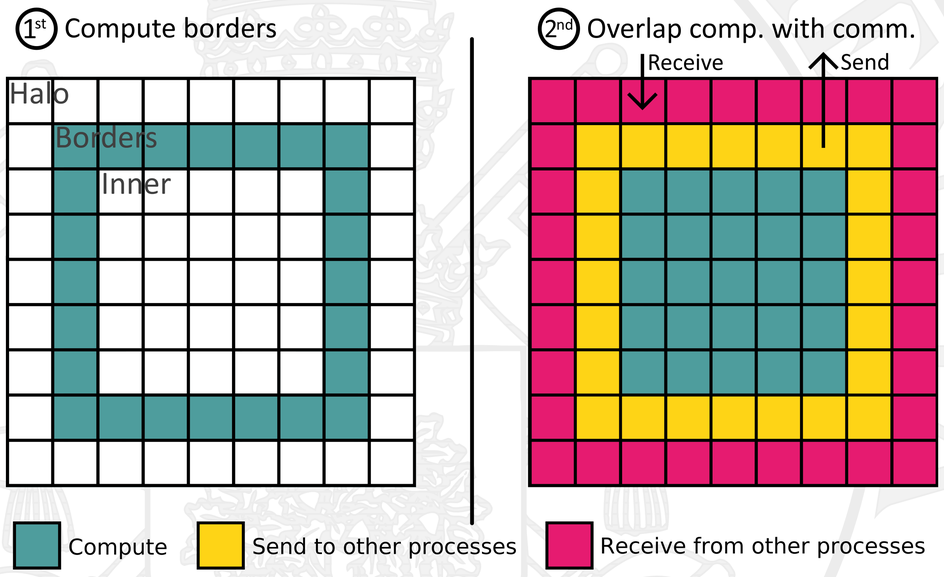
NOS IMPULSA Junta de Castilla y León

European Union
European Regional Development Fund

Introduction

- **Iterative Stencil computations** update array elements with previous iteration values according to an stencil pattern (neighbors and weights).
- **Manual multi-GPU approach:** Specific programs developed using CUDA/OpenCL + MPI
  - √ High performance using specific optimization techniques
  - × Lower portability and complex development: Explicit device and processes/nodes management: **communication and synchronization**
- **Multi-GPU stencil frameworks:** Restricted stencil types, device specific optimizations (lower portability); not always hide all programming complexities
- **Using modern abstract models**: E.g. **Celerity** [1], built of top of **SYCL**
- **Proposal: Parallel skeleton for multi-GPU stencil computations** as a C99 library function, built on top of **Controller and Hitmap** [2]
  - ⇒ Supports **any kind of n-dimensional geometric stencils**: compact and non-compact, symmetric and asymmetric (described as a pattern of weights)
  - ⇒ Transparent partition of data, **asynchronous data transfers and communications**, and computation/communication **overlapping**
  - ⇒ Support for a predefined generic kernel, or specific **optimized kernels** (user-defined or automatically generated)

Skeleton iterations

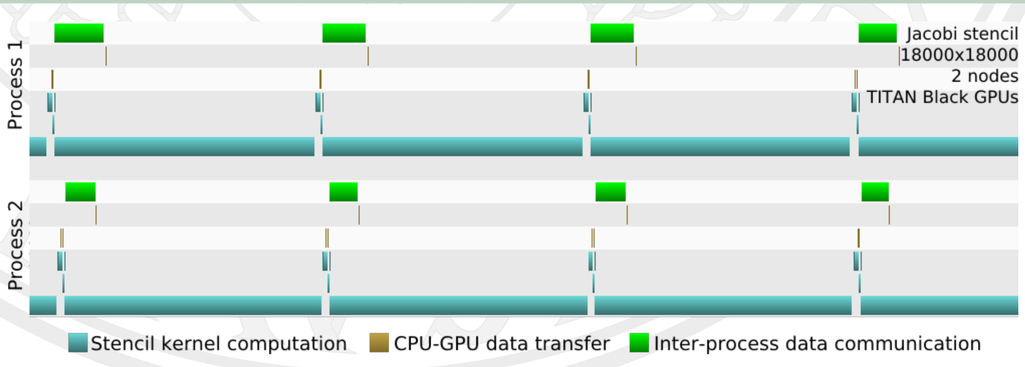


Example stencil program

```
/* Optimized stencil kernel definition */
CTRL_KERNEL_FUNCTION(Jacobi, CUDA, DEFAULT,
  KHitTile_float mat, KHitTile_float copy) {
  int x = thread_id_x;
  int y = thread_id_y;

  hit( mat, x, y ) = (
    hit( copy, x-1, y ) +
    hit( copy, x+1, y ) +
    hit( copy, x, y-1 ) +
    hit( copy, x, y+1 ) ) / 4;

  CTRL_KERNEL_END(CUDA);
}
/* Declare kernel to be used in the parallel skeleton */
REGISTER_STENCIL(Jacobi, CUDA, DEFAULT);

/* Parallel skeleton prototype included in header file */
void stencilComputation(int sizes[], HitShape stencilShape, float stencilData[],
  float factor, int numIterations, stencilFunction f_stencil,
  initDataFunction f_init, outputDataFunction f_output);
```

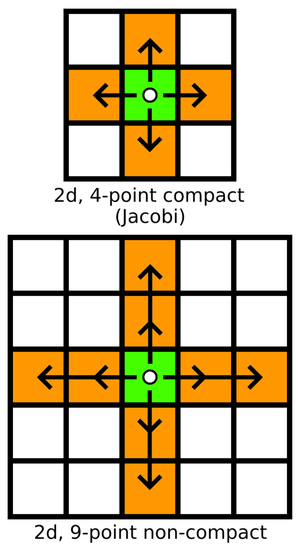
Video demo

Profiling (Comp./Comm. overlap)




```
/* Main program: stencil execution */
int main(int argc, char *argv[]) {
  ...
  HitShape shpStencil = hitShape( (-1, 1), (-1, 1) );
  float stencilPattern[] = {
    0, 1, 0,
    1, 0, 1,
    0, 1, 0
  };

  stencilComputation( sizes, shpStencil, stencilPattern, 0,
    numIter, Jacobi, initData, outputData );
  ...
}
```

Experimental results



2D stencils tested

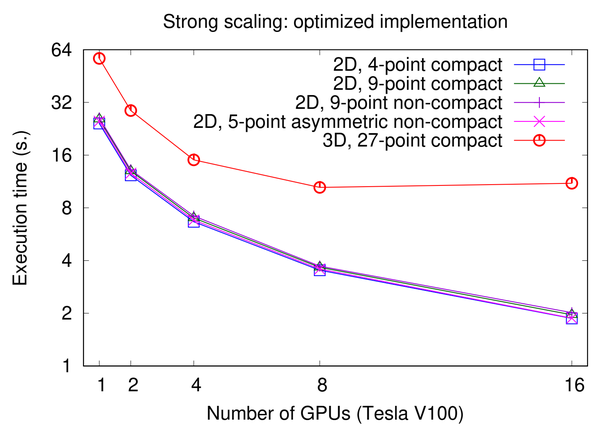
	Sequential reference	Proposal
1 GPU performance	306 Gflops	300 Gflops

Proposal vs. CUDA reference program: Performance comparison of 2D 4-point Jacobi stencil using the proposed skeleton and a CUDA reference program in 1 Tesla V100 GPU. It shows the efficiency of the kernel integration system.

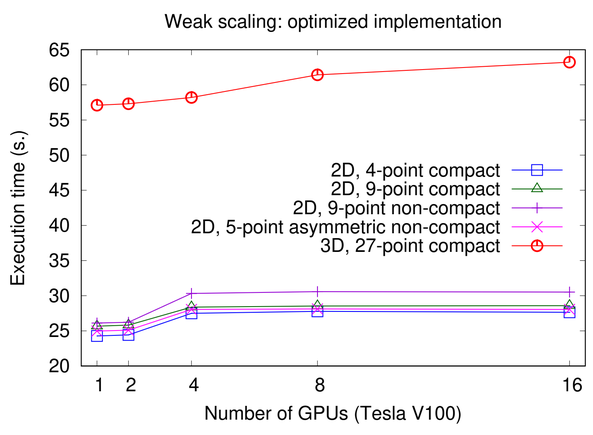
GPUs	Celerity avg. time (s)	Proposal avg. time (s)
1 Black	20.94	15.83
2 Black	11.99	9.12
2 Black + 2 V100	5.92	4.77

Proposal vs. Celerity: Strong scaling comparison of 2D 4-point Jacobi stencil using 2 TITAN Black + 2 Tesla V100 in two different nodes.

Strong and weak scaling: Up to 16 Tesla V100 GPUs, in 4 nodes



Strong scaling



Weak scaling

Conclusion and future work

- We present a **parallel skeleton for distributed multi-GPU stencil computations**
- Generic: It supports **any kind of n-dimensional geometric stencil**
- Built on top Controller + Hitmap (abstraction and portability)
- Transparent **data partition**, **communications** and **overlap** of data communication and computation
- **Good strong and weak scaling** in a cluster of up to 16 Tesla V100 GPUs in 4 nodes
- Higher performance **than a Celerity implementation** for 2D 4-point Jacobi stencil
- **Future work:** More complete experimentation, improved support for more heterogenous clusters

References

[1] Thoman, Peter and Salzmann, Philip and Cosenza, Biagio and Fahringer Thomas Celerity: High-Level C++ for Accelerator Clusters, 2019, DOI: 10.1007/978-3-030-29400-7_21

[2] Moreton-Fernandez, Ana and Ortega-Arranz, Hector and Gonzalez-Escribano, Arturo Controllers: An abstraction to ease the use of hardware accelerators. May, 2017, DOI: 10.1177/1094342017702962

http://trasgo.infor.uva.es/controller/

Acknowledgements

This research has been partially funded by the Spanish Ministerio de Ciencia e Innovación with the ERDF program of the European Union, project PCAS (TIN2017-88614-R); Junta de Castilla y León - FEDER Grants, projects PROPHET and PROPHET-2 (VA082P17, VA226P20); the Spanish Ministerio de Educación, Cultura y Deporte with a Beca de Colaboración Universitaria 2020/2021 (20CO1/013177); and by the program Salvador de Madariaga/Fulbright Scholar Grant (PRX17/00674).