

Liberté Égalité Fraternité

GPU PROGRAMMING

Through external scientific libraries





The complexity of GPUs programming A complex software environment



Different hardwares compatibility

Distribution	GCC	Clang	NVHPC	XLC	ArmC/C++	ICC
x86_64	6.x - 13.2	7.x - 17.0	23.x	No	No	2021.7
Arm64 sbsa	6.x - 13.2	7.x - 17.0	22.x	No	23.04.1	No

Compiler vs. Driver/Lib version



Different languages and API



The complexity of GPUs programming Memory handling



Data transfers in both directions



The complexity of GPUs programming Example on Adastra MI250 partition



Complexity of the connections

(nría_

The complexity of GPUs programming Example on Adastra MI250 partition

									W	/ith HS	A_E	NABLE_SD	MA=1													
Unidirectional copy peak Gio/s																		Bidire	ctiona	l copy	peak G	io/s				
	GPUs																					GP	Us			
		CPU I	NUMA	4	0	1	2	3	4	5	6	7			(NUMA	۱	0	1	2	3	4	5	6	7
From/to	0	1	2	3	c1	c6	c 9	ce	d1	d6	d9	de		From/to	0	1	2	3	c1	c6	c9	се	d1	d6	d9	de
0	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5		0	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
1	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5		1	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.3	29.3	28.7
2	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5		2	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
3	-	-	-	-	25.5	25.5	25.5	25.5	25.5	25.5	25.5	25.5		3	-	-	-	-	29.3	29.2	29.3	28.7	29.3	29.2	29.3	28.7
c1	25.2	2 25.2	25.2	25.2	-	50.9	36.9	37.5	37.5	36.9	51.0	51.0		c1	29.3	29.3	29.3	29.3	-	101.5	67.6	69.6	69.6	67.6	101.5	101.5
c6	25.1	25.1	25.1	25.1	51.0	-	37.5	36.9	36.9	36.9	51.0	51.0		c6	29.2	29.2	29.2	29.2	101.5	-	69.6	67.6	67.6	67.6	101.9	101.8
c9	25.2	25.2	25.2	25.2	36.9	37.5	-	51.0	51.0	51.0	37.5	36.9		c9	29.3	29.3	29.3	29.3	67.6	69.6	-	101.9	101.7	102.0	69.6	67.5
се	25.2	25.2	25.2	25.2	37.5	36.9	51.0	-	51.0	51.0	36.9	36.9		се	28.7	28.7	28.7	28.7	69.6	67.6	101.9	-	101.3	101.7	67.5	67.7
d1	25.2	25.2	25.2	25.2	37.5	37.0	51.0	51.0	-	51.1	36.9	37.5		d1	29.3	29.3	29.3	29.3	69.6	67.6	101.7	101.3	-	101.8	67.6	69.6
d6	25.2	25.2	25.2	25.2	36.9	36.9	51.0	51.0	51.0	-	37.5	36.9		d6	29.2	29.3	29.2	29.2	67.6	67.6	102.0	101.7	101.8	-	69.6	67.6
d9	25.2	25.2	25.2	25.2	51.0	51.0	37.5	36.9	36.9	37.5	-	51.0		d9	29.3	29.3	29.3	29.3	101.5	101.9	69.6	67.5	67.6	69.6	-	101.5
de	25.2	25.2	25.2	25.2	51.0	51.0	36.9	36.9	37.5	36.9	51.0	-		de	28.7	28.7	28.7	28.7	101.5	101.8	67.5	67.7	69.6	67.6	101.5	-

Data transfer throughput: non uniform

Ínría

The complexity of GPUs programming Asynchronism handling

for (int i = 0; i < nStreams; ++i) {
 int offset = i * streamSize;
 cudaMemcpyAsync(&d_a[offset], &a[offset], streamBytes, cudaMemcpyHostToDevice, stream[i]);
 kernel<<<streamSize/blockSize, blockSize, 0, stream[i]>>>(d_a, offset);
 cudaMemcpyAsync(&a[offset], &d_a[offset], streamBytes, cudaMemcpyDeviceToHost, stream[i]);
}

opy Engine	H2D - Stream 0		D2H - 0
ernel Engine		0	

Asynchronous Version I

Copy Engine	H2D - I		D2H - I H2D - 2		D2H - 2 H2D - 3		D2H - 3 H2D - 4		D2H - 4
Kernel Engine		1		2		3		4	

Asynchronous Version 2

Copy Engine	H2D - I	H2D - 2	H2D - 3	H2D - 4	D2H - 1	D2H - 2	D2H - 3	D2H - 4
Kernel Engine		1	2	3	4			

Time



The complexity of GPUs programming Fine grain parallelism, high number of parallel tasks





Find the good external library

- Find the nice library with appropriate features
- Avoid white elephant, be precise about features needs
- Software quality: tests, releases, stable API, neat documentation, active issues system
- Allows to focus on our algorithms level

Examples of external libraries Python ecosystem



Nvidia, numpy on CUDA GPUs



NumPy & SciPy for GPU

Numpy and Scipy on CUDA (+/-AMD Roc) GPUs



9

Examples of external libraries C/C++, Fortran ecosystem, GPU vendors libraries



Dense, Sparse and Tensor linear algebra, FFT, DNN Nvidia GPUs

CUDA Library	ROCm Library	Description
cuBLAS	rocBLAS	Basic Linear Algebra Subroutines
cuFFT	rocFFT	Fast Fourier Transfer Library
cuSPARSE	rocSPARSE	Sparse BLAS + SPMV
cuSolver	rocSolver	Lapack Library
AMG-X	rocALUTION	Sparse iterative solvers $far{far{ heta}}$ preconditioners with Geometric $far{far{ heta}}$ Algebraic MultiGrid
Thrust	rocThrust	C++ parallel algorithms library
CUB	rocPRIM	Low Level Optimized Parallel Primitives
cuDNN	MIOpen	Deep learning Solver Library
cuRAND	rocRAND	Random Number Generator Library
EIGEN	EIGEN	C++ template library for linear algebra: matrices, vectors, numerical solvers
NCCL	RCCL	Communications Primitives Library based on the MPI equivalents

AMD ROCm equivalence

Intel oneAPI ...

Examples of external libraries C/C++, Fortran ecosystem, not vendor specific

Ø

0

S

Ø

Ø

MAGMA

HYBRID ALGORITHMS

MAGMA uses a hybridization methodology, where algorithms of interest are split into tasks of varying granularity, and their execution is scheduled over the available hardware components. Scheduling can be static or dynamic. In either case, small non-parallelizable tasks, often on the critical path, are scheduled on the CPU, and larger more parallelizable ones, often Level-3 BLAS, are scheduled on the GPU. When CPU-GPU communication overhead becomes significant, the entire computation might run exclusively on the GPU, leading to what is termed as "GPU-native algorithms".



Ginkgo FEATURES AND SUPPORT MAGMA 2.7.2 FOR CUDA OR HIP CORE Library core contains architecture-MAGMA Pre-release FOR SYCL/DPC++ agnostic algorithm implementation Library Infrastructure **Algorithm Implementations** 👇 Ginkgo CUDA HIP SYCL Iterative Solvers Runtime polymorphism selects the right kernel Preconditioners depending on the target architecture Ø Ø Linear system solvers • Eigenvalue problem solvers Architecture-specific kernels execute the Auxiliary BLAS algorithm on target architecture Batched LA Sparse LA **CPU/GPU** Interface Multiple precision support REFERENCE OpenMP CUDA HIP DPC++ Mixed precision (including FP16) OpenMP kernels Reference kernels **CUDA GPU kernels** HIP GPU kernels **DPC++ GPU kernels** SpMV SpMV SpMV SpMV SpMV . Non-GPU-resident factorizations Solver kernels Solver kernels Solver kernels Solver kernels Solver kernels . . · Precond kernels · Precond kernels . Precond kernels Precond kernels Precond kernels **GPU-only factorizations** ... Multicore and multi-GPU support MAGMA DNN v1.4 Optimized architecture-specific kernels LAPACK testing Reference kernels are sequential kernels to Linux check correctness of algorithm design and Windows optimized kernels intel macOS **NVIDIA**

Linear algebra, 1 node of CPUs + GPUs

Ínría_ 11

Examples of external libraries C/C++, Fortran ecosystem, Distributed (MPI)

Application Codes

Higher-Level Libraries

PETSc

			TS (T	ime Ste	eppers)					DM (D	omain	Managen	ient)	1	
Bacl Ei	kward uler	RK	BDF	SSP	ARKIME	EX R	osenbroc W	k		Distribut Array	ed F st	lex (Un- ructured)			
		SN	IES (N	onlinea	r Solvers	;)				ТА	0 (Opt	mization)		
n Line rch	Newto Re	n Trus gion	t FAS	5 NG	MRES	NASM	ASPI	N		Newtor	n L N	evenberg- larquardt		Corrs (marine	
				K	SP (Kryl	ov Sub	ospace	Method	ls)					1 ma	
ES	Richard	son	CG	CGS	Bi-CGStab	TFG	MR N	INRES	GCF	Cheb	yshev P	pelined CC		• MUMPS	
					PC (Preco	ndition	ers)						• PaStiX	
litive warz	Bloo Jaco	:k bi	lacobi	ІСС	ILU	LU	J SC	DR I	MG	AMG	BDDC	Shell		• HPDDM	
					М	at (Op	erators	;)							
npresse Irse Ro	ed ow	Block CSR	E	Symmetr Block CS	ic SR De	ense	CUSPAF	SE Vie	ennaCL	FFT	Shell				
			Veo	: (Vect	ors)					I	S (Inde	x Sets)			
andard	1	CUE	DA	Vie	ennaCL					General	Blo	ck S	tride		
	E Line ch S itive varz	Backward Euler Line Newto ch Rep S Richards itive Bloc varz Jaco	Backward Euler RK SN Line Newton Trus ch Region S Richardson itive Block varz Jacobi npressed Block cSR ndard CUI	Backward Euler RK BDF SNES (N Line Newton Trust Region FAS S Richardson CG S S Richardson CG S S S S S S S S S S S S S S S S S S S	Backward Euler RK BDF SSP SNES (Nonlinea Line Newton Trust Region FAS NG ch Newton Trust Region FAS NG S Richardson CG CGS itive Block Jacobi Jacobi ICC npressed Block CSR Symmetr Block CS Vec (Vect undard CUDA Vie	Backward Euler RK BDF SSP ARKIME SNES (Nonlinear Solvers Line Newton Trust Region FAS NGMRES N KSP (Kryl S Richardson CG CGS Bi-CGStab Varz Block Jacobi Jacobi ICC ILU Inpressed rse Row Block CSR Symmetric Block CSR De Vec (Vectors) ManaCL	Backward Euler RK BDF SSP ARKIMEX Rd SNES (Nonlinear Solvers) Line Newton Trust Region FAS NGMRES NASM KSP (Krylov Sut :S Richardson CG CGS Bi-CGStab TFG PC (Preconditive Block Jacobi ICC ILU LU itive Block Jacobi ICC ILU LU Mat (Oppose Mat (Oppose Mat (Oppose Mat (Oppose indard CUDA ViennaCL	Backward Euler RK BDF SSP ARKIMEX Rosenbroc W SNES (Nonlinear Solvers) SNES (Nonlinear Solvers) SNES (Nonlinear Solvers) SNES (Nonlinear Solvers) Line ch Newton Trust Region FAS NGMRES NASM ASPIN SNES (Richardson CG CG CGS Bi-CGStab TFQMR N SREs (Nonlinear Solvers) ICC ILU LU SOME SOME SNES (Richardson CG CGS Bi-CGStab TFQMR N PC (Preconditione ICC ILU LU SO itive varz Block Jacobi Jacobi ICC ILU LU SO Some construction Symmetric Block CSR Dense CUSPAF Indard CUDA ViennaCL	Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W SNES (Nonlinear Solvers) Line ch Newton Trust Region FAS NGMRES NASM ASPIN KSP (Krylov Subspace Method S Richardson CG CGS Bi-CGStab TFQMR MINRES Itive varz Block Jacobi Jacobi ICC ILU LU SOR I Mat Operators) Mat CUDA ViennaCL	Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W SNES (Nonlinear Solvers) Line ch Newton Trust Region FAS NGMRES NASM ASPIN KSP (Krylov Subspace Methods) KSP (Krylov Subspace Methods) S Richardson CG CGS Bi-CGStab TFQMR MINRES GCR Itive varz Block Jacobi Jacobi ICC ILU LU SOR MG Pressed rse Row Block CSR Symmetric Block CSR Dense CUSPARSE ViennaCL undard CUDA ViennaCL	Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W Image: Construct of the second secon	Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W SNES (Nonlinear Solvers) Distributed Array P st Line ch Newton Trust Region FAS NGMRES NASM ASPIN KSP (Krylov Subspace Methods) Newton Lon M KSP (Krylov Subspace Methods) PC (Preconditioners) Itive varz Block Jacobi Jacobi ICC ILU LU SOR MG AMG BDDC Mat Operators) IS (Inde Itive varz Block Scr Symmetric Block CSR Dense CUSPARSE ViennaCL FFT Shell Madard CUDA ViennaCL IS (Inde	Backward Euler Backward Euler Backward Euler Backward Euler Charles Steppers) Sinte Steppers) TAO (Optimization) Newton Trust Region FAS NASM ASPIN TAO (Optimization) Newton Trust Region FAS NGMRES NASM ASPIN TAO (Optimization) Newton Krylov Subspace Methods) ES Richardson CG CGS Biock CSS CUSPARSE General Block Symmetric Jacobi ICC ILU LU Solution Trust (Pieceonditioners) This (Index Sets) <th c<="" td=""><td>Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W Distributed Array Plex (Un- structured) TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN Newton Levenberg- Marquardt KSP (Krylov Subspace Methods) IS Richardson CG CGS Bi-CGStab TFQMR MINRES GCR Chebyshev Pipelined CG Mat (Operators) Itive Mat (Operators) Block CSR Symmetric Block CSR Dense CUSPARSE ViennaCL FFT Shell Vec (Vectors) IS (Index Sets) Indard CUDA ViennaCL General Block Stride</td></th>	<td>Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W Distributed Array Plex (Un- structured) TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN Newton Levenberg- Marquardt KSP (Krylov Subspace Methods) IS Richardson CG CGS Bi-CGStab TFQMR MINRES GCR Chebyshev Pipelined CG Mat (Operators) Itive Mat (Operators) Block CSR Symmetric Block CSR Dense CUSPARSE ViennaCL FFT Shell Vec (Vectors) IS (Index Sets) Indard CUDA ViennaCL General Block Stride</td>	Backward Euler RK BDF SSP ARKIMEX Rosenbrock- W Distributed Array Plex (Un- structured) TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN TAO (Optimization) Line Newton Trust Region FAS NGMRES NASM ASPIN Newton Levenberg- Marquardt KSP (Krylov Subspace Methods) IS Richardson CG CGS Bi-CGStab TFQMR MINRES GCR Chebyshev Pipelined CG Mat (Operators) Itive Mat (Operators) Block CSR Symmetric Block CSR Dense CUSPARSE ViennaCL FFT Shell Vec (Vectors) IS (Index Sets) Indard CUDA ViennaCL General Block Stride

. . .

. . .

BLAS/LAPACK

12

Examples of external libraries C/C++, Fortran ecosystem, using StarPU runtime system

https://solverstack.gitlabpages.inria.fr/



Dense and Sparse linear algebra, multi nodes CPUs + GPUs



GEMM directly with CUDA

int main(){

// Perform matrix multiplication C = A*B
// where A, B and C are NxN matrices
int N = 16;
int SIZE = N*N;

// Allocate memory on the host
vector<float> h_A(SIZE);
vector<float> h_B(SIZE);
vector<float> h C(SIZE);

```
// Initialize matrices on the host
for (int i=0; i<N; i++){
    for (int j=0; j<N; j++){
        h_A[i*N+j] = sin(i);
        h_B[i*N+j] = cos(j);
    }
</pre>
```

// Allocate memory on the device
dev_array<float> d_A(SIZE);
dev_array<float> d_B(SIZE);
dev_array<float> d_C(SIZE);

```
// Copy data host to device
d_A.set(&h_A[0], SIZE);
d_B.set(&h_B[0], SIZE);
```

// Compute multiplication
matrixMultiplication(d_A.getData(), d_B.getData(), d_C.getData(), N);
cudaDeviceSynchronize();

// Copy data back to host from device d_C.get(&h_C[0], SIZE); cudaDeviceSynchronize();

GEMM CUDA

- Need to handle host and device arrays
- Need to code a kernel
- Need to handle each GPU one by one
- Need to handle asynchronism



Using external libraries : example Chameleon

```
#include <chameleon.h>
#include <vector>
using namespace std;
int main() {
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    int N = 16;
    int SIZE = N*N;
    // Allocate memory on the host
    vector<float> A(SIZE);
    vector<float> B(SIZE);
    vector<float> C(SIZE);
    // Initialize matrices on the host
    for (int i=0; i<N; i++){</pre>
        for (int j=0; j<N; j++){</pre>
            A[i*N+j] = sin(i);
            B[i*N+j] = cos(j);
    CHAMELEON Init( 32, 4 );
    CHAMELEON sgemm( ChamNoTrans, ChamNoTrans, N, N, N,
                     1.0, A.data(), N,
                           B.data(), N,
                      0.0, C.data(), N );
    CHAMELEON Finalize():
```

```
GEMM Chameleon (StarPU)
```

- Stay focused on traditionnal host arrays
- No need to code the function, just call it, (a data conversion is often necessary)
- Execute on any number of workers (CPUs + GPUs)
- Can call several algorithms without synchronization to let tasks being interleaved = optimization

15

2D Block Cyclic Layout

Using external libraries, Distributed (MPI)

int main() {

// Perform matrix multiplication C = A*B
// where A, B and C are NxN matrices
int N = 16;
int NB = 2;
int SIZE = N*N;

CHAMELEON_Init(32, 4); CHAMELEON_Set(CHAMELEON_TILE_SIZE, NB); int NMPI = CHAMELEON_Comm_size(); int GRID_P = NMPI; int GRID_Q = 1;

struct data_fill fill_args_A = { 1.0 }; CHAMELEON_map_Tile(ChamW, ChamUpperLower, descA, fill_cpu, &fill_args_A);

struct data_fill fill_args_B = { 2.0 }; CHAMELEON_map_Tile(ChamW, ChamUpperLower, descB, fill_cpu, &fill_args_B);

CHAMELEON_sgemm_Tile(ChamNoTrans, ChamNoTrans, 1.0, descA, descB, 0.0, descC);

CHAMELEON_Finalize();

MPI GEMM Chameleon (StarPU)

			Matrix	point	of view	<i>,</i>					Р	roc	cesso	r poir	it of vi	en	'		
0	2	4	0	2	4	0	2	4][
	닏	Ľ		Ē	Ľ		Ē		0	0	0		2	2	2		4	4	4
1	3	5	1	3	5	1	3	5	0	0	0		2	2	2		4	4	4
0	2	4	0	2	4	0	2	4	0	0	0		2	2	2		4	4	4
	3	5		3	5		3	5	0	0	0		2	2	2		4	4	4
	H	H	H	H			\square		0	0	0		2	2	2		4	4	4
0	2	4	0	2	4	0	2	4					_						
1	3	5	1	3	5	1	3	5	1	1	1		3	3	3		5	5	5
0	2	4	0	2	4	0	2	4	1	1	1		3	3	3		5	5	5
1	3	5	1	3	5	1	3	5	1	1	1		3	3	3		5	5	5
0	2	4	0	2	4	0	2	4	1	1	1		3	3	3		5	5	5
																-			

• User can provide a function to fill a given tile, with its specific metadata (some parameters, a file, ...)

struct data_fill {
 float value;
};
static int fill_cpu(const CHAM_desc_t *desc,
 cham_uplo_t uplo, int m, int n,
 CHAM_tile_t *tile, void *user_data)
{
 struct data_fill *data = (struct data_fill *)user_data;
 float *T = (float *)tile->mat;
 /* Get the dimension of the tile */
 int tempnm = (m == (desc->mt-1)) ? (desc->m - m * desc->mb) : desc->mb;
 int tempnn = (n == (desc->nt-1)) ? (desc->n - n * desc->nb) : desc->nb;
 /* fill the tile with the user's value */
 for (int i=0; i<tempnm*tempnn; i++){
 T[i] = data->value;
 }

(void)uplo; return 0;





