PFC OpenCL based machine learning labeling of biomedical datasets

Project goals

- Learn OpenCL
- Test OpenCL
- Improve performance (reduce execution time) of a Medical imaging program

Project scope

OpenCL +



NVIDIA architecture



ATI architecture

Project environment

- Adaboost algorithm for classifying datasets
- Medical imaging techniques to visualize body datasets



PHOTORESEARCHERS

Algorithm

- alfaclass = rc[j1];
- polaritat= gc[j1];
- thresh = bc[j1]*255;
- sumalfa = sumalfa + alfaclass;
- valida = fabs(polaritat)>0.0001;
- positiu=(polaritat>0.5);
- mesgran = i>thresh;
- sum = sum + valida*(positiu*mesgran*alfaclass+ (1-positiu)*(1-mesgran) *alfaclass);

OpenCL: NDRange space

			NDRang	e space
	Work G	roup 0,0		
WI	WI	WI	WI	WI
0,0	1,0	2,0	3,0	0,0
WI	WI	WI 2,1	WI	WI
0,1	1,1		3,1	0,1
WI	WI	WI	WI	WI
0,2	1,2	2,2	3,2	0,2
WI	WI	WI	WI	WI
0,3	1,3	2,3	3,3	0,3

	Work G	roup 1,0	
WI	WI	WI	WI
0,0	1,0	2,0	3,0
WI	WI	WI	WI
0,1	1,1	2,1	3,1
WI	WI	WI	WI
0,2	1,2	2,2	3,2
WI	WI	WI	WI
0,3	1,3	2,3	3,3

1	Work G	roup 0,1	
WI	WI	WI	WI
0,0	1,0	2,0	3,0
WI	WI	WI	WI
0,1	1,1	2,1	3,1
WI	WI	WI	WI
0,2	1,2	2,2	3,2
WI	WI	WI	WI
0,3	1,3	2,3	3,3

	Work G	roup 1,1	
WI	WI	WI	WI
0,0	1,0	2,0	3,0
WI	WI	WI 2,1	WI
0,1	1,1		3,1
WI	WI	WI 2,2	WI
0,2	1,2		3,2
WI 0,3	WI 1,3	WI 2,3	WI 3,3

GPGPU architecture

Local memory	Local memory	
Multi processor	Multi processor	
arallel control unit	Parallel control unit	
roc. unit Proc. unit	Proc. unit Proc. unit	
roc. unit Proc. unit	Proc. unit Proc. unit	
Proc. unit Unit	Proc. unit Proc. unit	
roc. Proc.	Proc. Proc.	

NVIDIA Fermi architecture





NVIDIA Fermi architecture

	SM				Benc	hmark
		Ir	nstructio	on Cach	e	- W S. com
	War	Warp Scheduler		War	Warp Scheduler	
	Dis	spatch U	nit	Dispatch Unit		nit
CUDA Core		+			+	
Dispatch Port		Registe	er File (3	12,768 x	32-bit)	
Operand Collector		-	-	-	LD/ST	-
P Unit INT Unit	Core	Core	Core	Core	LD/ST	
1 1					LD/ST	SFU
Result Queue	Core	Core	Core	Core	LD/ST	
	Corre	Core	Com	Corre	LD/ST	
	Core	Core	Core	Core	LD/ST	SEU
	Core	Core	Core	Core	LD/ST	
					LD/ST	
	Core	Core	Core	Core	LD/ST	
					LD/ST	SFU
	Core	Core	Core	Core	LDIST	
					LDIST	
	Core	Core	Core	Core	LDIST	
					LD/ST	SFU
	Core	Core	Core	Core	LD/ST	
	-	Second Int	erconne	ct Netwo	-	-
			harring blo	monu / I	1 Cache	
		04 NB 34	nance me	anony r c	- Gacile	_
		-	Unitorn	Cache		
	Tex		Tex	Tex		Tex
			Texture	Cache		_
	Verte	x Fetch	Tesse	llator	Viewp	ort
		Attrib	to Setur	Stream	Transf	orm
		- Adding of	o octup		a de ar	

ATI architecture





ATI architecture



OpenCL: code example

}

ATI thread and instruction level parallelism code

kernel void operation(__global float4 input, __global float4 output){

}

output[get_local_id(0)]=
 input[get_local_id(0)]+input[get_local_id(0)];

NVIDIA thread level parallelism code

kernel void operation(__global float input, __global float output){

output[get_local_id(0)]=
 input[get_local_id(0)]+input[get_local_id(0)];

OpenCL medical imaging implementation

- Reducing PCIe usage to 1/64th
- Adding an extra kernel
- First step to test and validate



Gradient Kernel



#define BLOCK_DIMX 32 #define BLOCK_DIMY 4

_kernel void kernelGradient(__global unsigned char* values, __global float* gradients, __local float* local_gradients, int dimx, int dimy, int dimz) {

__local float tile[BLOCK_DIMY + 2][BLOCK_DIMX + 2]; float infront; float behind;

//fill tile and infront

}

for (k = 1 ; k < dimz-1 ; k++)

behind = tile[get_local_id(1)][get_local_id(0)-1]; tile[get_local_id(1)][get_local_id(0)+1] = infront;

// calculate gradients

// copy results from registers
// to local memory (4 transfers per thread 128x4, 1024 lost cycles)

// copy results from local to global memory
// (4 transfers each 16 threads 8x4, 19200 lost cycles)

// we loose 20.224 cycles instead of 307.200 cycles

Gradient Kernel



Classification kernel

_kernel void kernelClassif(__global unsigned char* values,

global float* gradients,

global unsigned char* voxelsOut,

global float* rc,

__global float* gc,

__global float* bc,

___constant int* var,

_local float* sumArr,

_local float* sumalfaArr,

_local float* thisVoxel) {

// Read alfaclass, polaritat, thresh

// Fill thisVoxel first 2 values

for (z=1;z<(var[4]-1);++z){

}

// Fill the next 6 thisVoxel values

// Copy the first 8 thisVoxel values to the last 8 thisVoxel positions

// Calculate two matrix of 240 values and do reduction for each one

voxelsOut[get_group_id(0)+(get_num_groups(0)*get_group_id(1))+(get_num_groups(0)*get_num_groups(1)*(z-1))] = // Compare the two final values to obtain 1 or 0

Classification kernel



Results





OpenCL analysis

Stencil method for wave propagation simulations





OpenCL analysis

Based on Micikevicius GPU algorithm



OpenCL performance

Comparing CUDA with GPU-OpenCL



OpenCL portability

 Testing a working and tuned code for NVIDIA in an ATI card



OpenCL for ATI

Exploring ATI architecture



Architectural trends

- CPU and GPU integration
- Intel Sandy Bridge CPU
- AMD APU (Accelerated Processing Unit)
- Lots of startups: Tilera, Zii, Smooth Stone...
- Low power consumption for HPC (GPU's,ARM,Godson)

Conclusions

- OpenCL for GPU's is a good option for data-parallel
- GPU's for HPC have still a way to walk
- Possible convergence of CPU and GPU architectures
- What is sure is that future looks parallel