Introduction to OpenCL

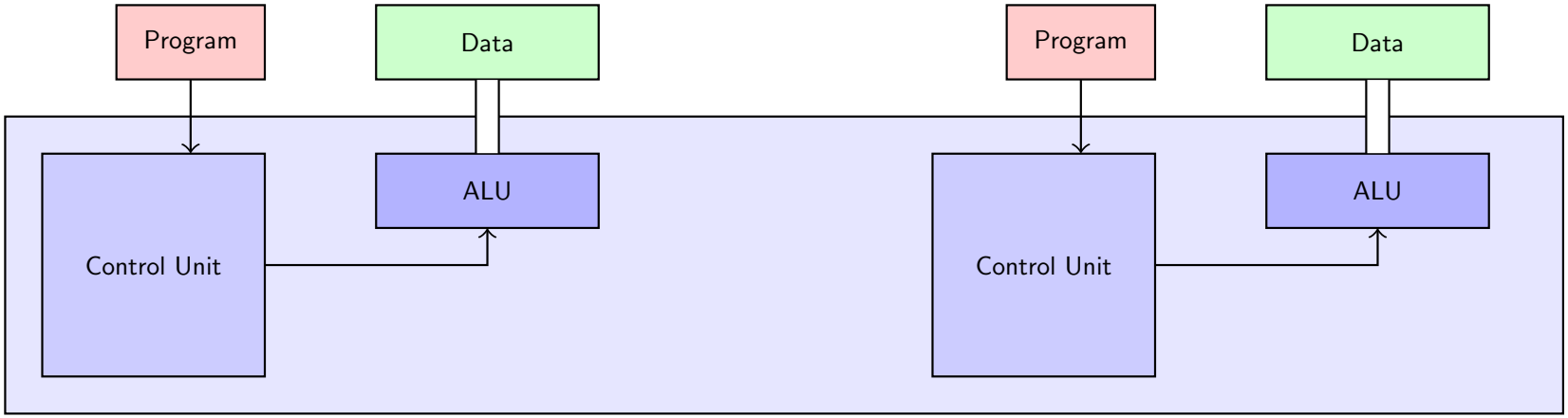
Computing on your Graphics Hardware

Maurice Leclaire

TumFUG
Linux / Unix
get-together

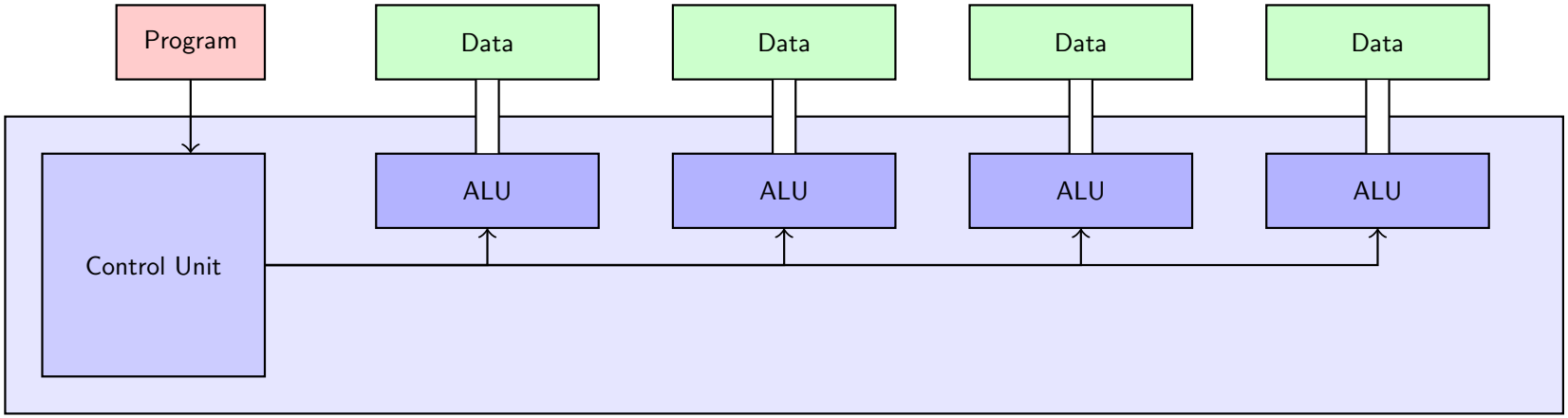
June 22, 2011

Normal CPU Architecture



- Seperate control unit in each core
- Can execute different programs on each core

GPU / SIMD Architecture



- Single instruction multiple data
- Executes the same program on different data

When is this useful?

- The same computition on many data items
- Data items must be independent, e.g. matrix multiplication
- Not useful for totally different programs
- Not useful for dependent data items

OpenCL

- Device independent framework to use SIMD devices
- Runtime device sceduling
- Runtime compilation for different devices
- Host program on CPU executes a kernel on the target device

OpenCL C

- included language for programming OpenCL
- based on C99

Example: Matrix multiplication

$$A, B \in n \times n$$

$$C = A \times B \in n \times n$$

$$c_{i,j} = \sum_{k=0}^{n} a_{i,k} \cdot b_{k,j}$$

Sequential matrix multiplication

```
 1 void matrixmul(int *A, int *B, int *C)
 2 {
 3     for (int i = 0; i < n; i++) {
 4         for (int j = 0; j < n; j++) {
 5             int m = 0;
 6             for (int k = 0; k < n; k++) {
 7                 m += A[i*n+k] * B[k*n+j];
 8             }
 9             C[i*n+j] = m;
10         }
11     }
12 }
```

Time $\mathcal{O}\left(n^3\right)$

Matrix multiplication with OpenCL

```
 1 __kernel void matrixmul(__global int *A,
 2                          __global int *B,
 3                          __global int *C)
 4 {
 5     int n = get_global_size(0);
 6     int i = get_global_id(0);
 7     int j = get_global_id(1);
 8
 9     int m = 0;
10     for (int k = 0; k < n; k++) {
11         m += A[i*n+k] * B[k*n+j];
12     }
13     C[i*n+j] = m;
14 }
```

Time $\mathcal{O}(n)$ *(if enough execution units)*

Executing the kernel

```
 1 int matrixmul(int n, int *A, int*B, int *C)
 2 {
 3     cl_uint numPlatforms = 0;
 4     clGetPlatformIDs(0, NULL, &numPlatforms);
 5     if (numPlatforms == 0)
 6         return -1;
 7     cl_platform_id *platforms = (cl_platform_id *)
 8         malloc(numPlatforms*sizeof(cl_platform_id));
 9     clGetPlatformIDs(numPlatforms, platforms, NULL);
10
11     cl_uint numDevices = 0;
12     clGetDeviceIDs(platforms[0], CL_DEVICE_TYPE_GPU,
13                    0, NULL, &numDevices);
14     if (numDevices == 0)
15         return -1;
16     cl_device_id *devices = (cl_device_id *)
17         malloc(numDevices*sizeof(cl_device_id));
18     clGetDeviceIds(platforms[0], CL_DEVICE_TYPE_GPU,
19                    numDevices, devices, NULL);
```

Executing the kernel

```
20      cl_context context = clCreateContext(
21          NULL, numDevices, devices, NULL, NULL, NULL);
22
23      cl_command_queue cmdQueue = clCreateCommandQueue(
24          context, devices[0], 0, NULL);
25
26      cl_mem a = clCreateBuffer(context,
27          CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
28          n*n*sizeof(int), A, NULL);
29      cl_mem b = clCreateBuffer(context,
30          CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
31          n*n*sizeof(int), B, NULL);
32      cl_mem c = clCreateBuffer(context,
33          CL_MEM_READ_WRITE,
34          n*n*sizeof(int), NULL, NULL);
```

Executing the kernel

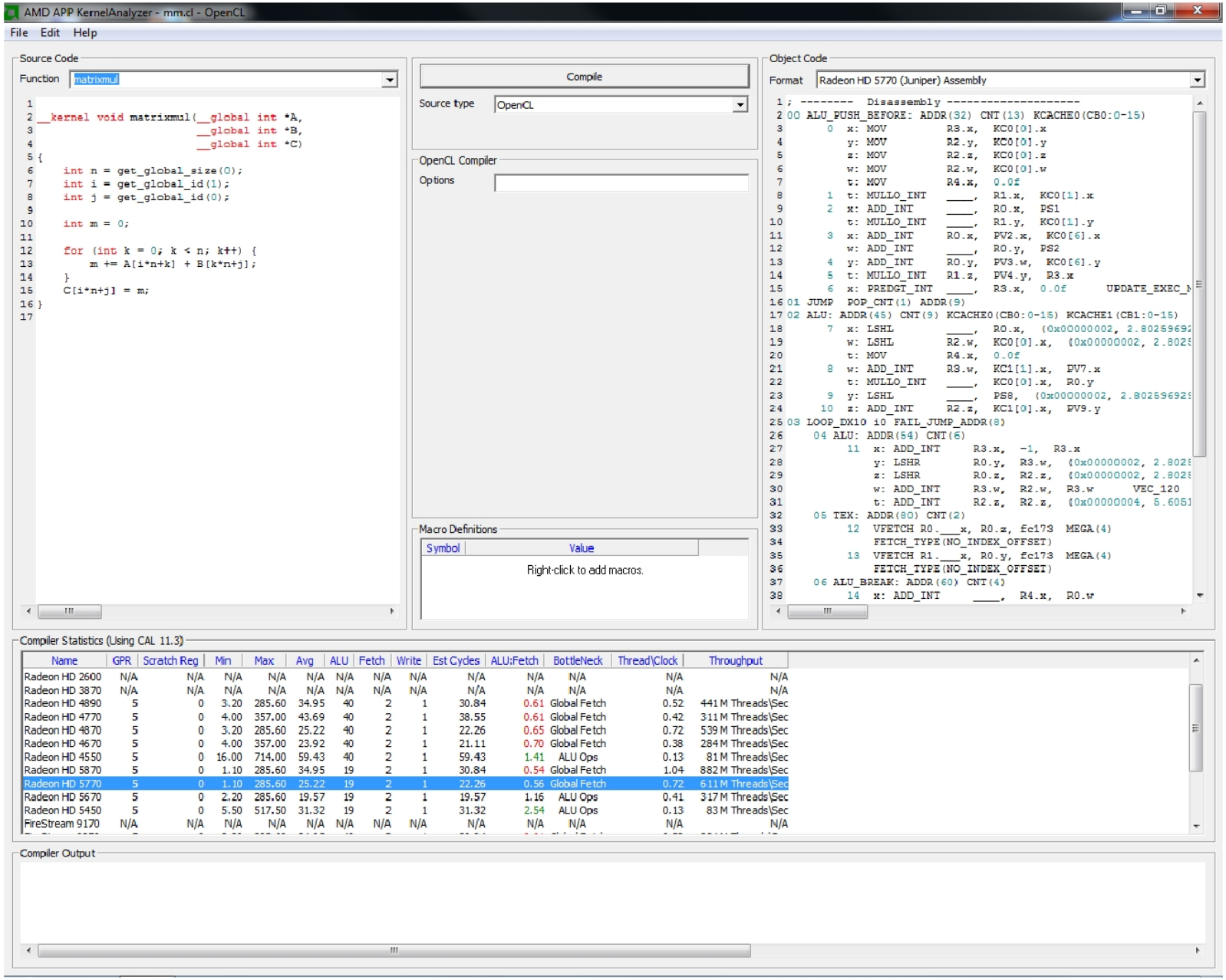
```
35      const char *source; //contains OpenCl source
36
37      program = clCreateProgramWithSource(context, 1,
38          &source, NULL, NULL);
39
40      cl_int buildErr = clBuilProgram(program,
41          numDevices, devices, NULL, NULL, NULL);
42      if (buildErr != CL_SUCCESS)
43          return -1;
44
45      cl_kernel kernel = clCreateKernel(program,
46          "matrixmul", NULL);
```

Executing the kernel

```
47      size_t globalWorkSize[2];
48      globalWorkSize[0] = n;
49      globalWorkSize[1] = n;
50
51      clSetKernelArg(kernel, 0, sizeof(cl_mem), &a);
52      clSetKernelArg(kernel, 1, sizeof(cl_mem), &b);
53      clSetKernelArg(kernel, 2, sizeof(cl_mem), &b);
54
55      clEnqueueNDRangeKernel(cmdQueue, kernel, 2,
56          NULL, globalWorkSize, NULL, 0, NULL, NULL);
57
58      clEnqueueReadBuffer(cmdQueue, c, CL_TRUE, 0,
59          n*n*sizeof(int), C, 0, NULL, NULL);
```

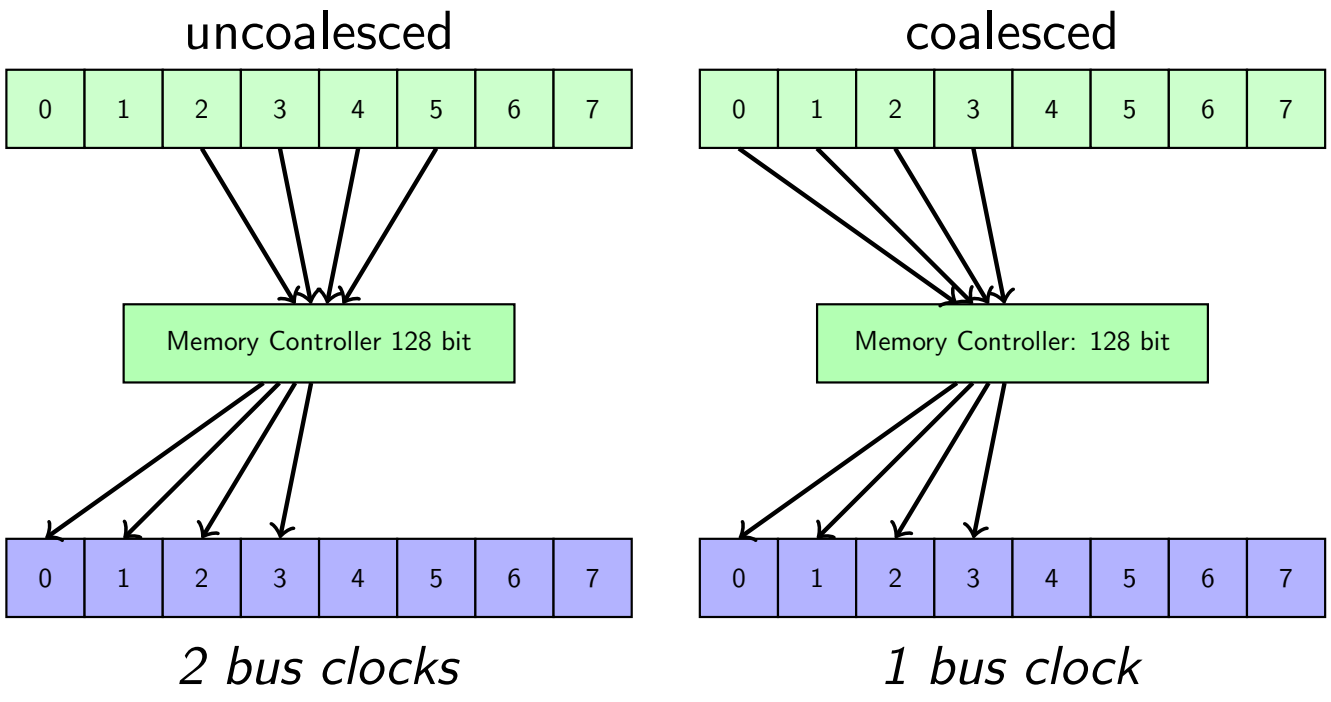
Executing the kernel

```
60      clReleaseKernel(kernel);
61      clReleaseProgram(program);
62      clReleaseCommandQueue(cmdQueue);
63      clReleaseMemObj(a);
64      clReleaseMemObj(b);
65      clReleaseMemObj(c);
66      clReleaseContext(context);
67
68      free(devices);
69      free(platforms);
70
71      return 0;
72  }
```

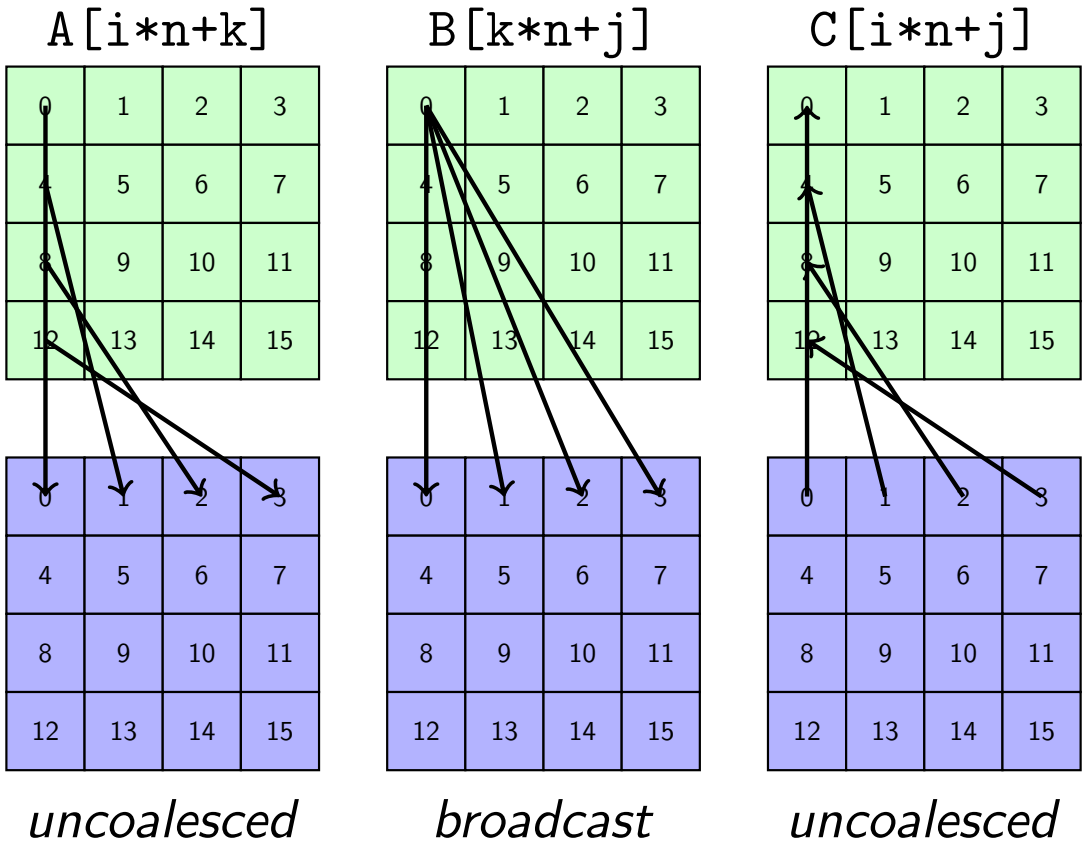

AMD APP KernelAnalyzer - mm.cl - OpenCL
File Edit Help
Source Code
Function matrixmul
__kernel void matrixmul(__global int *A,
__global int *B,
__global int *C)
{
int n = get_global_size(0);
int i = get_global_id(1);
int j = get_global_id(0);
int m = 0;
for (int k = 0; k < n; k++) {
m += A[i*n+k] + B[k*n+j];
}
C[i*n+j] = m;
}
Compile
Source type OpenCL
OpenCL Compiler
Options
Macro Definitions
Symbol Value
Right-click to add macros.
Object Code
Format Radeon HD 5770 (Juniper) Assembly
; -------- Disassembly --------------------
00 ALU_PUSH_BEFORE: ADDR(32) CNT(13) KCACHE0(CB0:0-15)
0 x: MOV R3.x, KC0[0].x
y: MOV R2.y, KC0[0].y
z: MOV R2.z, KC0[0].z
w: MOV R2.w, KC0[0].w
t: MOV R4.x, 0.0f
1 t: MULLO_INT ____, R1.x, KC0[1].x
2 x: ADD_INT ____, R0.x, PS1
t: MULLO_INT ____, R1.y, KC0[1].y
3 x: ADD_INT R0.x, PV2.x, KC0[6].x
w: ADD_INT ____, R0.y, PS2
4 y: ADD_INT R0.y, PV3.w, KC0[6].y
5 t: MULLO_INT R1.z, PV4.y, R3.x
6 x: PREDGT_INT ____, R3.x, 0.0f UPDATE_EXEC_M
01 JUMP POP_CNT(1) ADDR(9)
02 ALU: ADDR(45) CNT(9) KCACHE0(CB0:0-15) KCACHE1(CB1:0-15)
7 x: LSHL ____, R0.x, (0x00000002, 2.8025969
w: LSHL R2.w, KC0[0].x, (0x00000002, 2.802
t: MOV R4.x, 0.0f
8 w: ADD_INT R3.w, KC1[1].x, PV7.x
t: MULLO_INT ____, KC0[0].x, R0.y
9 y: LSHL ____, PS8, (0x00000002, 2.80259692
10 z: ADD_INT R2.z, KC1[0].x, PV9.y
03 LOOP_DX10 i0 FAIL_JUMP_ADDR(8)
04 ALU: ADDR(54) CNT(6)
11 x: ADD_INT R3.x, -1, R3.x
y: LSHR R0.y, R3.w, (0x00000002, 2.802
z: LSHR R0.z, R2.z, (0x00000002, 2.802
w: ADD_INT R3.w, R2.w, R3.w VEC_120
t: ADD_INT R2.z, R2.z, (0x00000004, 5.605
05 TEX: ADDR(80) CNT(2)
12 VFETCH R0.___x, R0.z, fc173 MEGA(4)
FETCH_TYPE(NO_INDEX_OFFSET)
13 VFETCH R1.___x, R0.y, fc173 MEGA(4)
FETCH_TYPE(NO_INDEX_OFFSET)
06 ALU_BREAK: ADDR(60) CNT(4)
14 x: ADD_INT ____, R4.x, R0.w
Compiler Statistics (Using CAL 11.3)
Name GPR Scratch Reg Min Max Avg ALU Fetch Write Est Cycles ALU:Fetch BottleNeck Thread\Clock Throughput
Radeon HD 2600 N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A
Radeon HD 3870 N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A
Radeon HD 4890 5 0 3.20 285.60 34.95 40 2 1 30.84 0.61 Global Fetch 0.52 441 M Threads\Sec
Radeon HD 4770 5 0 4.00 357.00 43.69 40 2 1 38.55 0.61 Global Fetch 0.42 311 M Threads\Sec
Radeon HD 4870 5 0 3.20 285.60 25.22 40 2 1 22.26 0.65 Global Fetch 0.72 539 M Threads\Sec
Radeon HD 4670 5 0 4.00 357.00 23.92 40 2 1 21.11 0.70 Global Fetch 0.38 284 M Threads\Sec
Radeon HD 4550 5 0 16.00 714.00 59.43 40 2 1 59.43 1.41 ALU Ops 0.13 81 M Threads\Sec
Radeon HD 5870 5 0 1.10 285.60 34.95 19 2 1 30.84 0.54 Global Fetch 1.04 882 M Threads\Sec
Radeon HD 5770 5 0 1.10 285.60 25.22 19 2 1 22.26 0.56 Global Fetch 0.72 611 M Threads\Sec
Radeon HD 5670 5 0 2.20 285.60 19.57 19 2 1 19.57 1.16 ALU Ops 0.41 317 M Threads\Sec
Radeon HD 5450 5 0 5.50 517.50 31.32 19 2 1 31.32 2.54 ALU Ops 0.13 83 M Threads\Sec
FireStream 9170 N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A N/A
Compiler Output

Coalescing memory accesses

ATI Radeon HD 5770 / Memory bus with: 128 bit



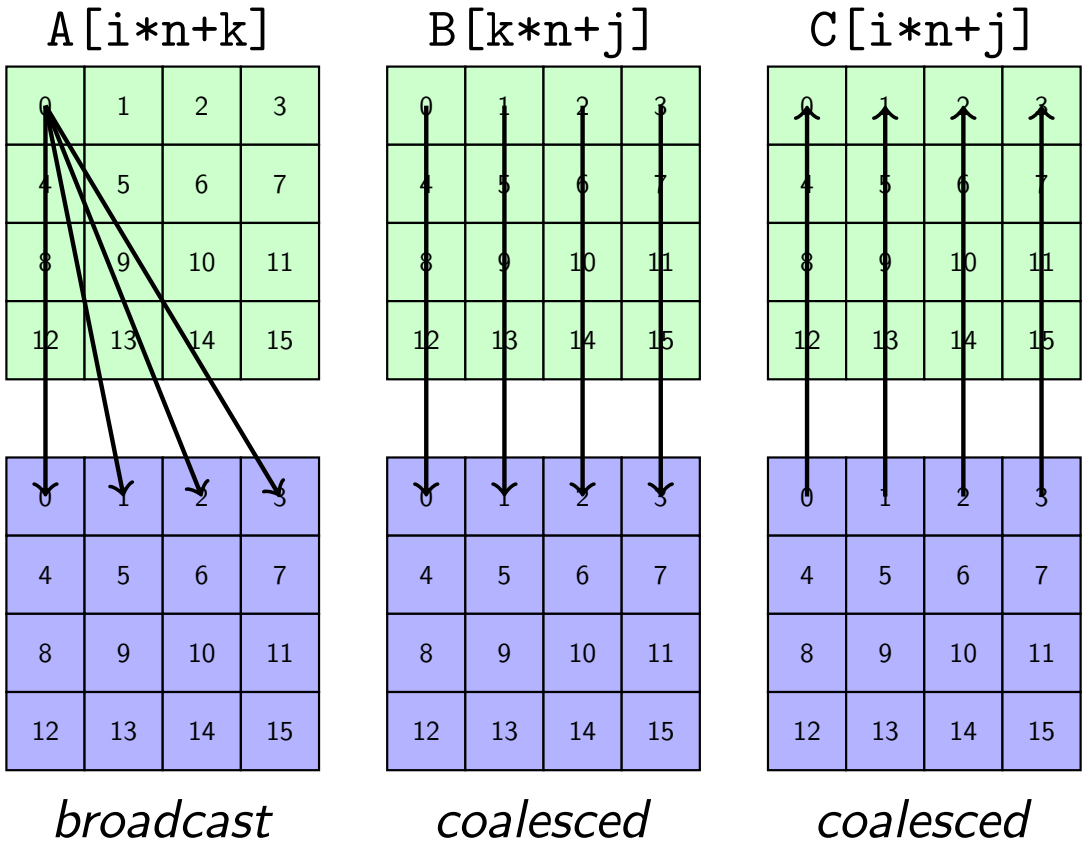
Memory accesses



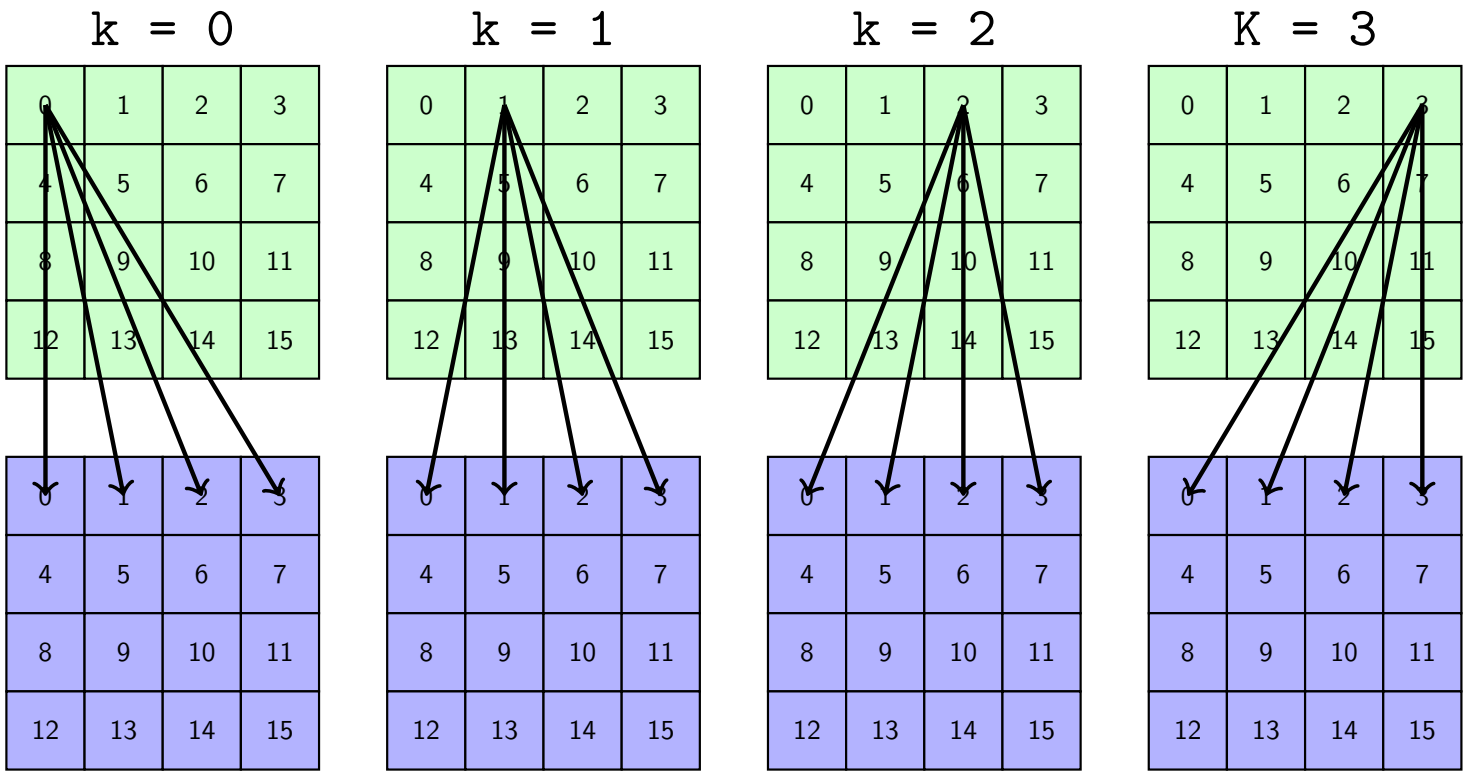
Matrix multiplication with different thread mapping

```
1  __kernel void matrixmul(__global int *A,
2                          __global int *B,
3                          __global int *C)
4  {
5      int n = get_global_size(0);
6      int i = get_global_id(1);
7      int j = get_global_id(0);
8
9      int m = 0;
10     for (int k = 0; k < n; k++) {
11         m += A[i*n+k] * B[k*n+j];
12     }
13     C[i*n+j] = m;
14 }
```

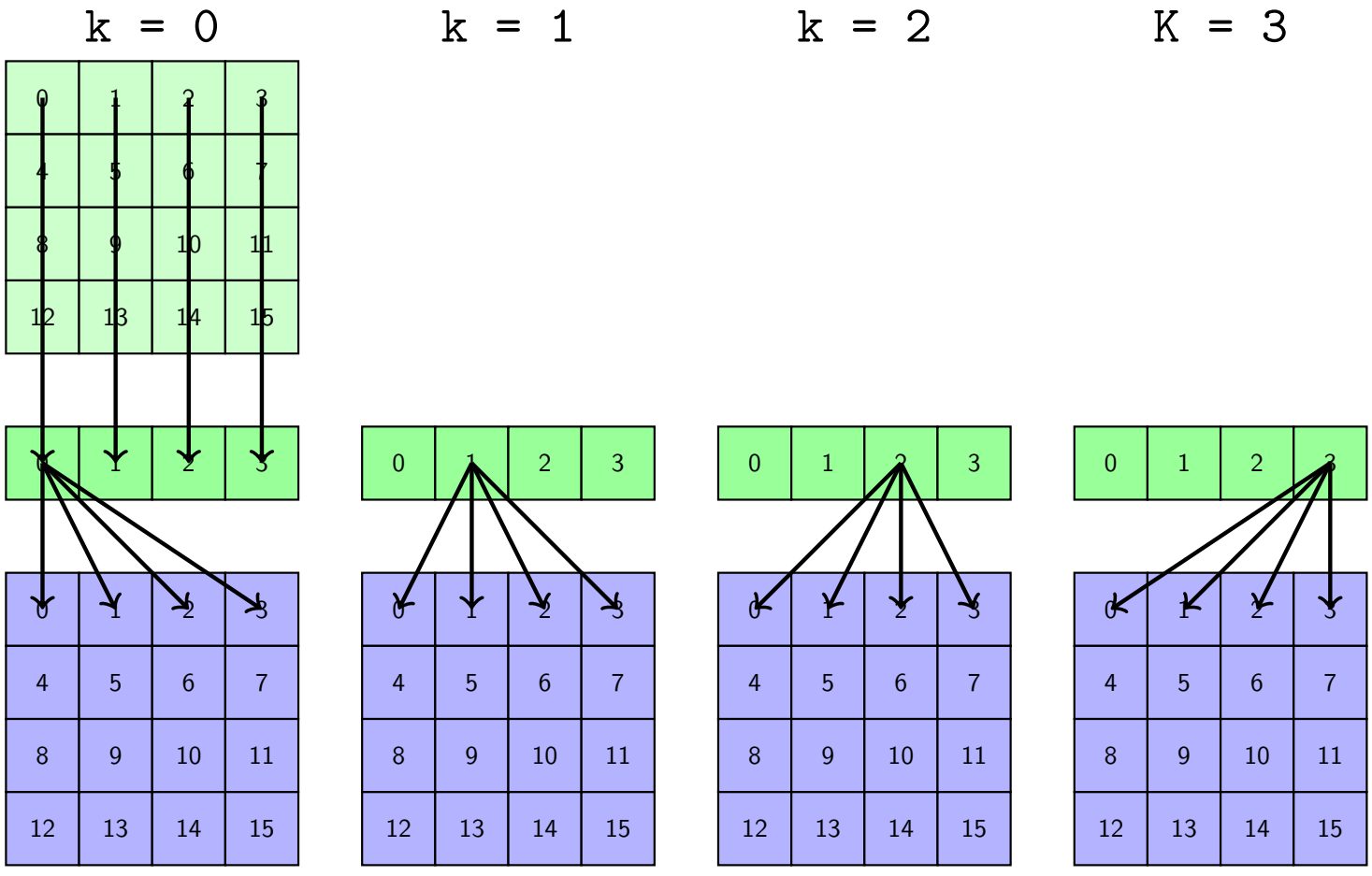
Memory accesses with different thread mapping



Broadcast memory accesses during loop



Memory accesses using local memory



Matrix multiplication using local memory

How can this be done?

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- Dividing the global space into local parts
- Using a local row number of 1
- Local column size should be as big as possible

Matrix multiplication using local memory

```
 1 __kernel void matrixmul(__global int *A,
 2 __global int *B, __global int *C,  __local int *tmp)
 3 {
 4     int n = get_global_size(0);
 5     int i = get_global_id(1);
 6     int j = get_global_id(0);
 7     int nl = get_local_size(0);
 8     int jl = get_local_id(0);
 9     int num = get_num_groups(0);
10
11     int m = 0;
12     for (int k = 0; k < num; k++) {
13         temp[jl] = A[i*n+k*nl+jl];
14         barrier(CLK_LOCAL_MEM_FENCE);
15         for (int l = 0; l < nl; l++) {
16             m += tmp[l] * B[(k*nl + l)*n+j];
17         }
18         barrier(CLK_LOCAL_MEM_FENCE);
19     }
20     C[i*n+j] = m;
21 }
```

Executing the kernel

```
22      size_t globalWorkSize[2];
23      globalWorkSize[0] = n;
24      globalWorkSize[1] = n;
25
26      size_t localWorkSize[2];
27      clGetDeviceInfo(devices[0],
28          CL_DEVICE_MAX_WORK_GROUP_SIZE,
29          sizeof(size_t), localWorkSize, NULL);
30      localWorkSize[1] = 1;
31
32      clSetKernelArg(kernel, 0, sizeof(cl_mem), &a);
33      clSetKernelArg(kernel, 1, sizeof(cl_mem), &b);
34      clSetKernelArg(kernel, 2, sizeof(cl_mem), &b);
35      clSetKernelArg(kernel, 3,
36          localWorkSize[0]*sizeof(int), NULL);
37
38      clEnqueueNDRangeKernel(cmdQueue, kernel, 2,
39          NULL, globalWorkSize, NULL, 0, NULL, NULL);
```

Speed comparison

AMD Phenom II x4 965 vs. ATI Radeon HD 5770

Matrix multiplication with 4096×4096 elements

Environment	Time	Speed factor
CPU (sequential)	4020 sec	1
CPU (OpenMP)	990 sec	4
GPU	100 sec	40
GPU (coalesced)	5,5 sec	700
GPU (local memory)	5 sec	800