C AND OPENCL GENERATION FROM MATLAB

LUÍS REIS, JOÃO BISPO, JOÃO CARDOSO

SPeCS
Special-Purpose Computing
Systems, languages and tools

Faculty of Engineering
University of Porto

Universidade do Porto
FEUP Faculdade de Engenharia

16th of April, 2015
Multicore Software Engineering, Performance, Applications, and Tools (MUSEPAT)
30th ACM/SIGAPP Symposium On Applied Computing (SAC’15)

OUTLINE

- Introduction
- Motivation
- MATISSE OpenCL back-end
- Results
- Conclusions and Future Work

THE AGE OF PARALLELISM – CPU

- Parallelism in CPUs
  - SIMD: **Data parallelism** on a single thread
  - Multicore: Requires **Task parallelism**.
- Both are required for maximum efficiency.

THE AGE OF PARALLELISM – GPU

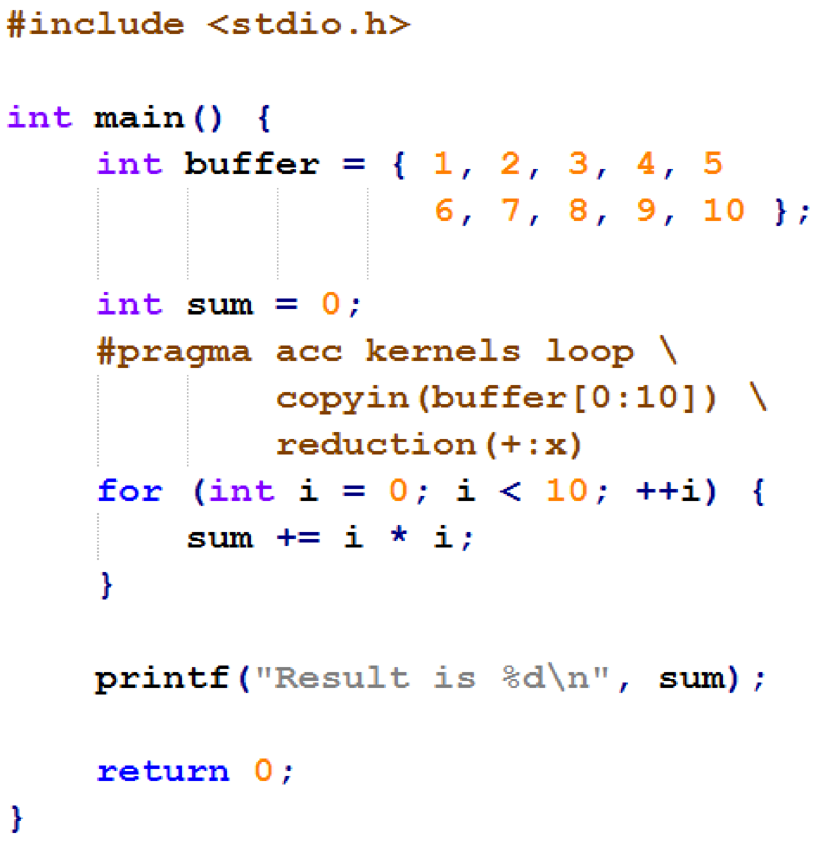
- Initially GPUs mostly used for graphical computing
  - Could be used for other operations, but that was far too much work
  - Usually have their own memory
- GPGPUs: **General-Purpose** Graphics Processing Unit
  - Still focused on graphics, still tend to have a separate memory
  - Easier to program now
- GPGPUs require parallelism:
  - Take longer than CPUs for sequential tasks
  - With parallelism, speedups of 1000x are possible

PROGRAMMING MODELS – DIRECTIVE-DRIVEN

- Some approaches let the programmer specify parallelism declaratively
    - “This part of the code can be made parallel”
    - Acceptable performance with relatively small effort.
    - Code is annotated with “directives” – code that is recognized by the compiler
    - Examples are OpenMP and OpenACC

PROGRAMMING MODELS – OPENACC

- Directive-driven extensions for C, C++ or FORTRAN.
- Compilers automatically generate the GPGPU code and communications.
- Suitable for accelerators, including GPGPUs
    - Data-transfers are explicit, using copyin, copyout, copy and present.



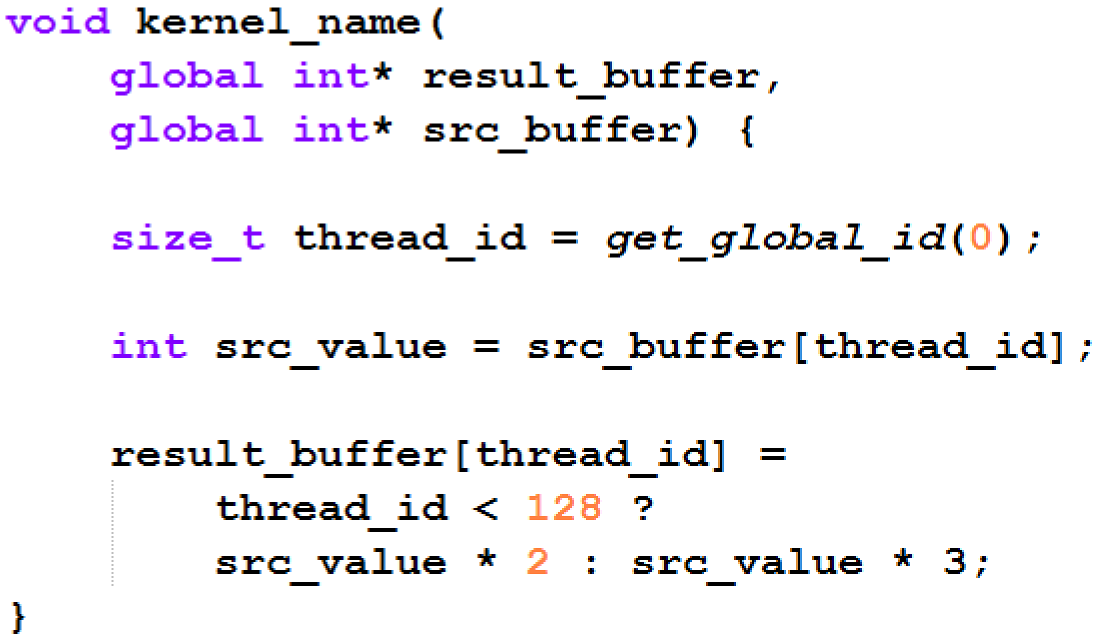
```
#include <stdio.h>

int main() {
    int buffer = { 1, 2, 3, 4, 5
                   6, 7, 8, 9, 10 };

    int sum = 0;
    #pragma acc kernels loop \
            copyin(buffer[0:10]) \
            reduction(+:x)
    for (int i = 0; i < 10; ++i) {
        sum += i * i;
    }

    printf("Result is %d\n", sum);

    return 0;
}
```

PROGRAMMING MODELS – LANGUAGES

- Old days:
  - Shader languages: HLSL, GLSL
- More recently:
  - GPGPU-specific languages: CUDA, OpenCL
  - CUDA is a language by NVIDIA, extends C, C++ or Fortran
  - OpenCL is a standard by Khronos, API + C-based language

PROGRAMMING MODELS – OPENCL

- Programming language and API (C/C++ inspired)
- Initially for GPGPU, currently supports multicore CPUs and even FPGAs
- Supported by Intel, AMD, NVIDIA, ARM, Qualcomm, Apple, ALTERA and Xilinx
- Parallel parts in OpenCL, remaining code in host language (e.g., C)



```
void kernel_name(
    global int* result_buffer,
    global int* src_buffer) {

    size_t thread_id = get_global_id(0);

    int src_value = src_buffer[thread_id];

    result_buffer[thread_id] =
        thread_id < 128 ?
        src_value * 2 : src_value * 3;
}
```

MOTIVATION – MULTI-TARGET CODE

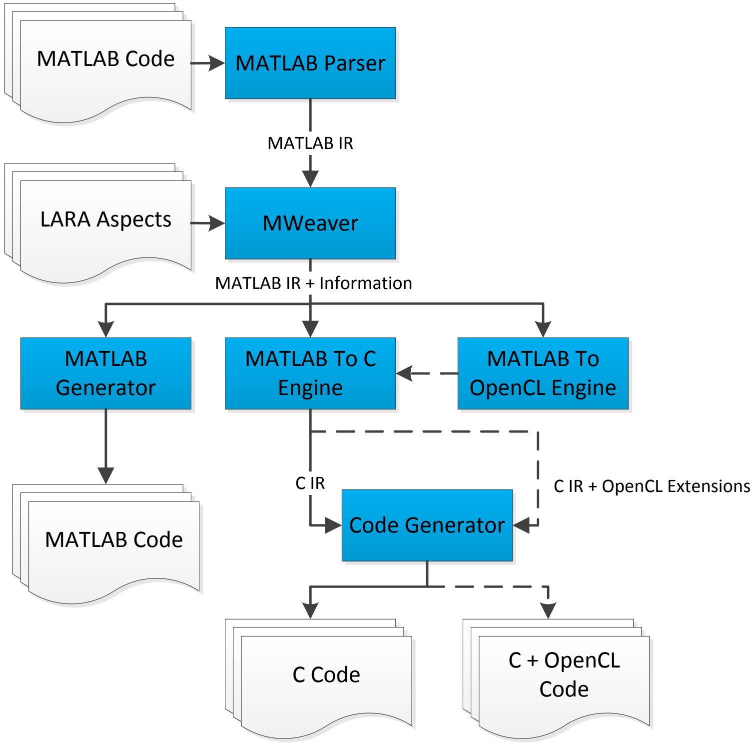
- To get the most performance we need low-level code (C, OpenCL)
- However, low-level code usually is not performance portable
- To maximize performance, different targets require different code
- Additionally, may have special requirements
    - Embedded systems without floating-point HW units, or with units that perform poorly
    - HW synthesis (compliance to different tools)

MOTIVATION – MULTI-TARGET CODE

- Possible Solution:
    - Start from clean implementation, specialize to target
- Problem:
    - Hard to transform low-level code, too many implementation details
- Our approach:
    - High-level description (MATLAB)
    - Augmented with information about implementation (LARA aspects)

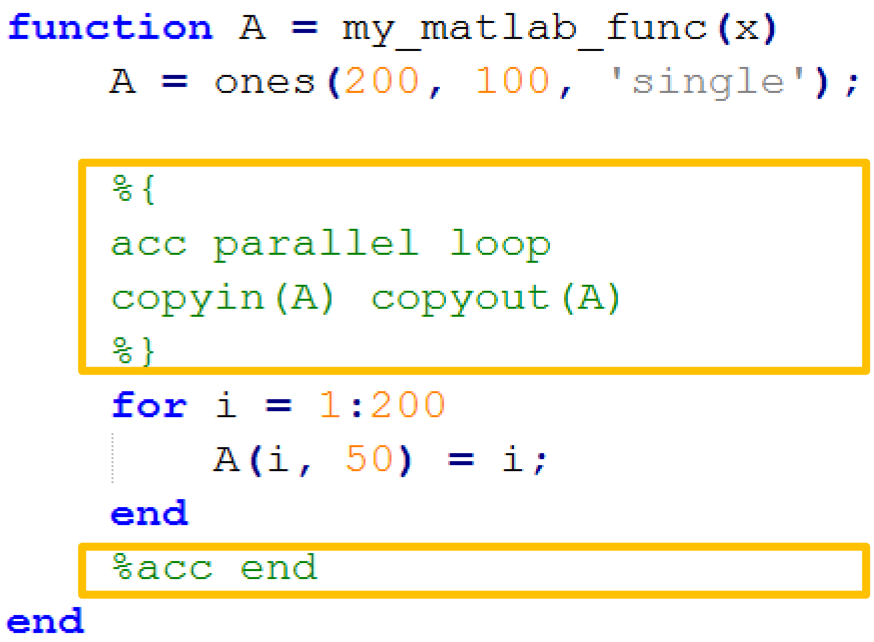
MATISSE – OPENCL BACKEND

- Proof-of-concept OpenCL backend
  - Developed during MSc
- MATLAB compiler that generates C + OpenCL code
- Based on the MATISSE framework
- MATLAB code is extended with OpenACC-based directives



MATISSE – OPENCL BACKEND

- Regions of code are marked as parallel
    - Each loop iteration is independent of the others.
- Copyin: Which variables are copied to the GPU before execution begins.
- Copyout: Which variables are copied out of the GPU after execution ends.
- Other directives are supported

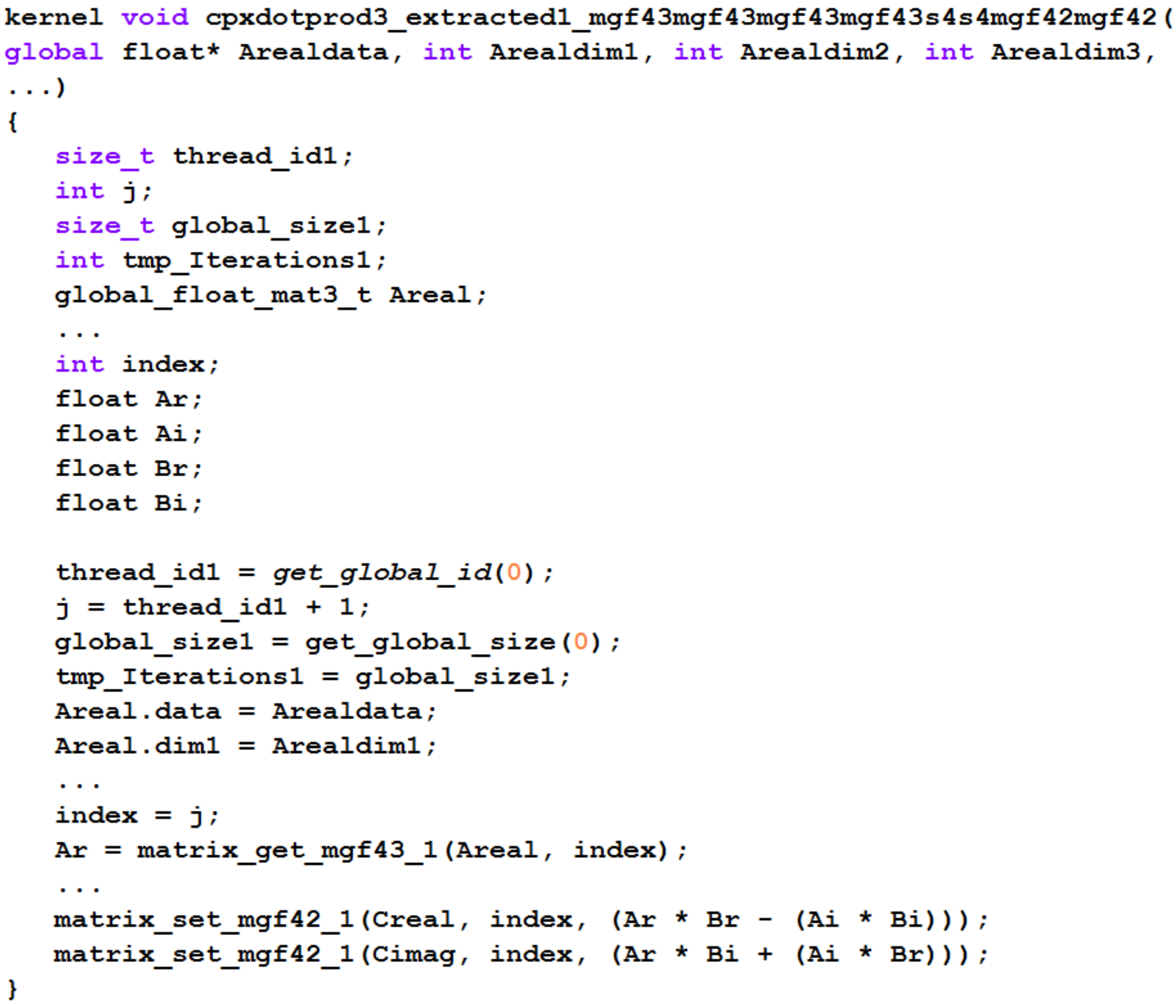


```
function A = my_matlab_func(x)
    A = ones(200, 100, 'single');

    %{
    acc parallel loop
    copyin(A) copyout(A)
    %}
    for i = 1:200
        A(i, 50) = i;
    end
    %acc end
end
```

MATISSE – OPENCL BACKEND

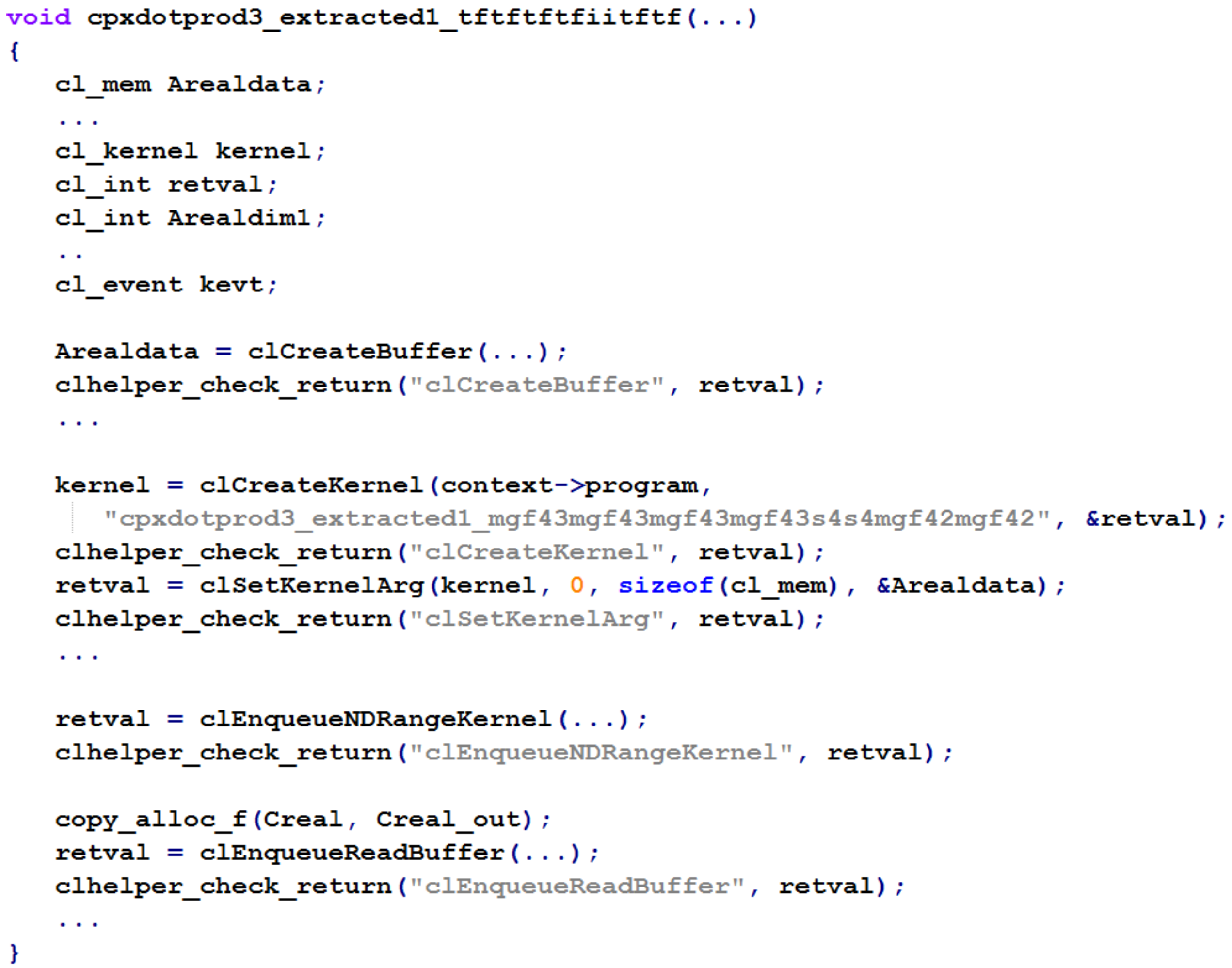
- We reuse and extend the MATISSE IRs:
    - MATLAB AST
    - C IR
- The MATISSE C backend handles sequential code sections.
- MATISSE CL overrides the code generator for the outlined functions.
    - Generates the OpenCL code and the C wrappers.

GENERATED SAMPLE CODE (OPENCL)



```
kernel void cpxdotprod3_extracted1_mgf43mgf43mgf43mgf43s4s4mgf42mgf42(
global float* Arealdata, int Arealdim1, int Arealdim2, int Arealdim3,
...)
{
   size_t thread_id1;
   int j;
   size_t global_size1;
   int tmp_Iterations1;
   global_float_mat3_t Areal;
   ...
   int index;
   float Ar;
   float Ai;
   float Br;
   float Bi;

   thread_id1 = get_global_id(0);
   j = thread_id1 + 1;
   global_size1 = get_global_size(0);
   tmp_Iterations1 = global_size1;
   Areal.data = Arealdata;
   Areal.dim1 = Arealdim1;
   ...
   index = j;
   Ar = matrix_get_mgf43_1(Areal, index);
   ...
   matrix_set_mgf42_1(Creal, index, (Ar * Br - (Ai * Bi)));
   matrix_set_mgf42_1(Cimag, index, (Ar * Bi + (Ai * Br)));
}
```

GENERATED SAMPLE CODE (C WRAPPER)



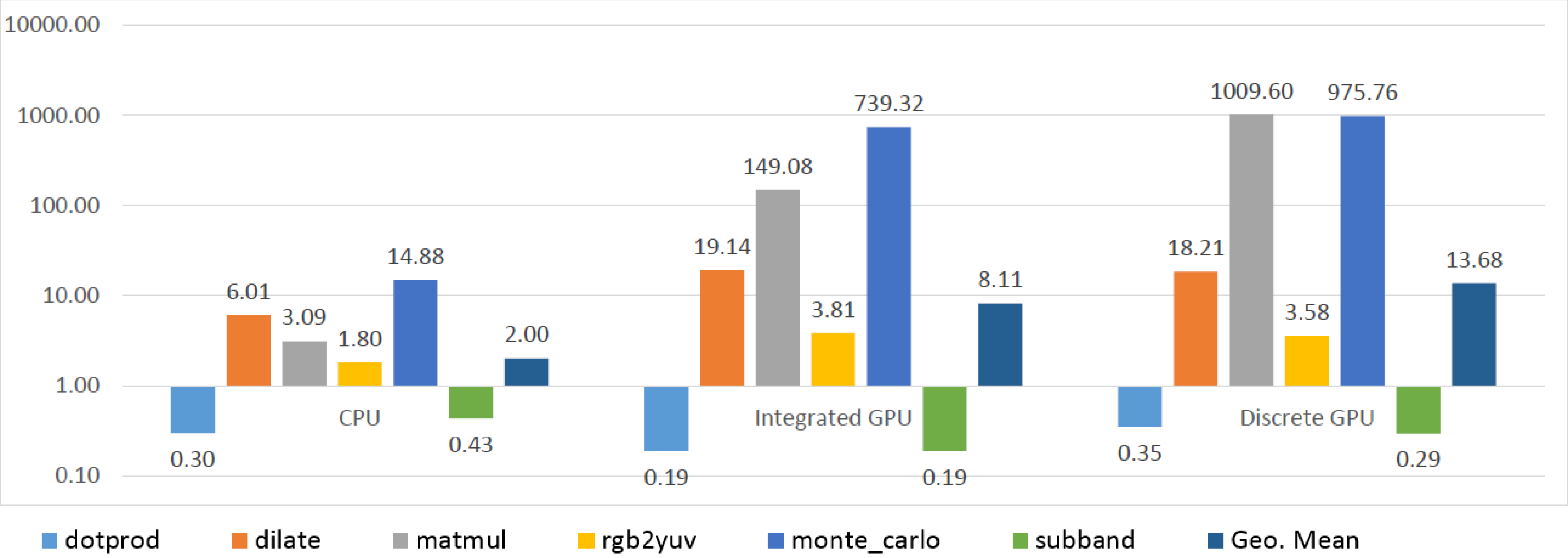
```
void cpxdotprod3_extracted1_tftftftfiitftf(...)
{
   cl_mem Arealdata;
   ...
   cl_kernel kernel;
   cl_int retval;
   cl_int Arealdim1;
   ..
   cl_event kevt;

   Arealdata = clCreateBuffer(...);
   clhelper_check_return("clCreateBuffer", retval);
   ...

   kernel = clCreateKernel(context->program,
      "cpxdotprod3_extracted1_mgf43mgf43mgf43mgf43s4s4mgf42mgf42", &retval);
   clhelper_check_return("clCreateKernel", retval);
   retval = clSetKernelArg(kernel, 0, sizeof(cl_mem), &Arealdata);
   clhelper_check_return("clSetKernelArg", retval);
   ...

   retval = clEnqueueNDRangeKernel(...);
   clhelper_check_return("clEnqueueNDRangeKernel", retval);

   copy_alloc_f(Creal, Creal_out);
   retval = clEnqueueReadBuffer(...);
   clhelper_check_return("clEnqueueReadBuffer", retval);
   ...
}
```

BENCHMARKS

- Benchmarks:
  - Reused some benchmarks already used for MATISSE C
  - Most are from embedded computing
  - Matmul: Naive implementation of matrix multiplication
  - Monte Carlo option pricing: Adapted from a MathWorks example

RESULTS – TOTAL TIME

- CPU: AMD A10-7850K@3.7GHz w/ GPU (integrated), GPU: R7 280X (discrete)
- Includes time spent on data transfers



RESULTS – MATMUL

- Modified matmul (in MATLAB) with optimizations from nVidia exemple
  - Loop Tiling
  - Local Memory

RESULTS – MATMUL

- Modified matmul (in MATLAB) with optimizations from nVidia example
  - Loop Tiling
  - Local Memory

1024×1024	matmul (s)	matmul_nv (s)	Speedup
CPU	9.1	2.0	4.6×
GPU (int.)	0.19	0.062	3.1×
GPU (disc.)	0.028	0.035	0.8×

RESULTS – ODROID

- We were able to compile and run our programs on an Odroid board
  - ARM'sbig.LITTLE configuration and a PowerVR SGX544MP3 GPU
  - Android 4.2.2 (though we bypassed Dalvik entirely)
  - The same processor used by some smartphones.
  - Preliminary results, only
- Sadly, we were rarely able to obtain speedups
  - Only 5% faster in matrix multiplication.
  - 75% slower for the dilate benchmark.
  - Monte Carlo Option Pricing can have statistically insignificant speedups (less than 95% confidence for N = 5000), or significant slowdowns (30% slower for N = 1000)
- We hope to improve these results with future optimizations (such as thread coarsening and use of texture memory)

CONCLUSIONS

- Proof-of-concept OpenCL back-end from MATLAB
  - Based on the MATISSE framework
- Good results on desktop GPUs
- Embedded systems' SOC performance needs more time for experiments and analysis.
- Future Work:
  - Improve MATLAB compatibility (take advantage of idiomatic operations)
  - Specialize code according to target

FUTURE WORK

```
%{
acc parallel loop

copyin(readonly Areal, readonly
Aimag, readonly Breal, readonly
Bimag, numElements)

copyout(Creal, Cimag)
%}
for j=1:numElements
 index=j;
 Ar = Areal(index);
 Ai = Aimag(index);
 Br = Breal(index);
 Bi = Bimag(index);
 Creal(index) = Ar*Br-Ai*Bi;
 Cimag(index) = Ar*Bi+Ai*Br;
end
%acc end
```

```
%!parallel
Creal = Areal.*Breal-Aimag.*Bimag;
Cimag = Areal.*Bimag+Aimag.*Breal;
%!end
```

THE END

THANK YOU!

Questions?

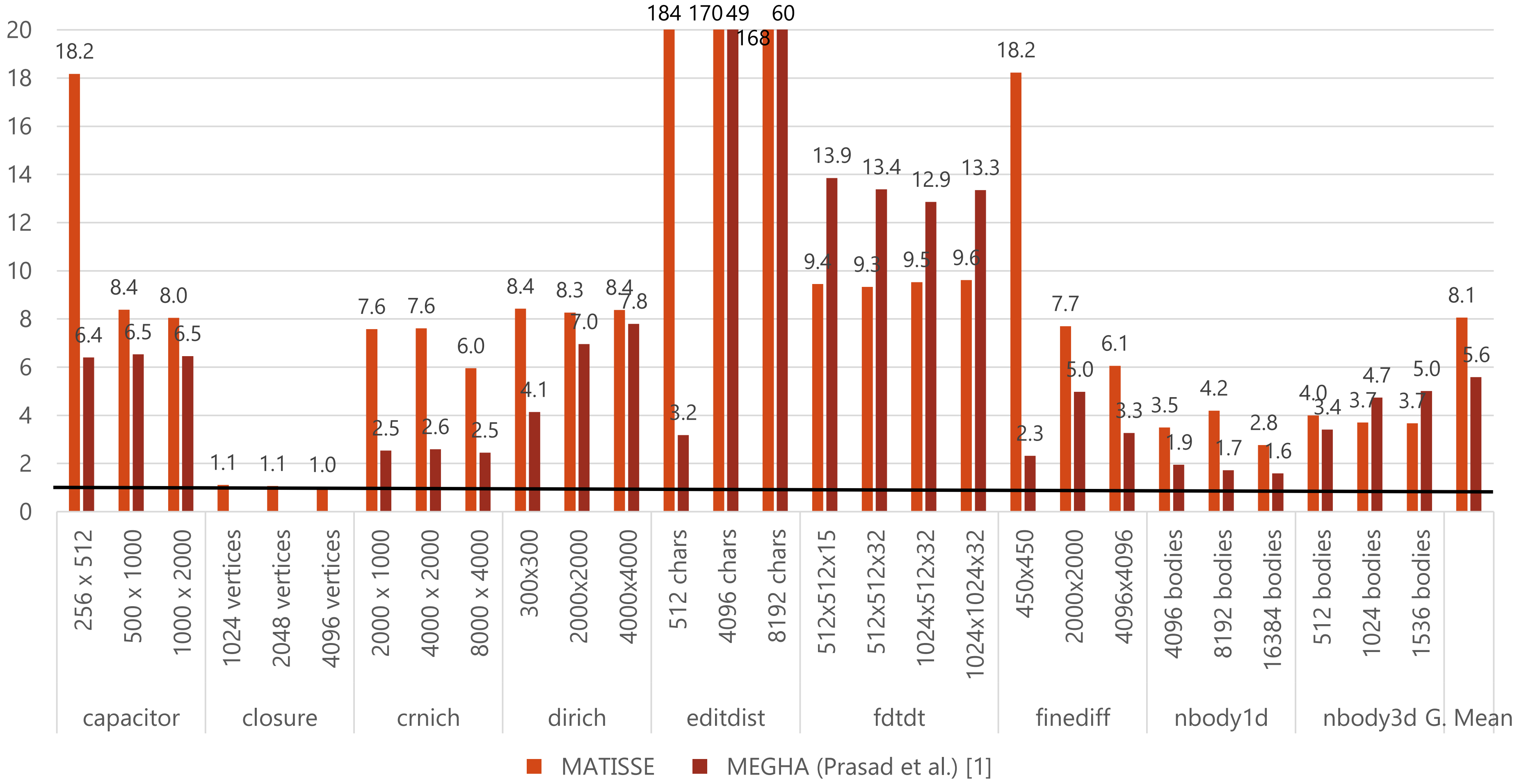
Demo of MATISSE (C only):

http://specs.fe.up.pt/tools/matisse/

RELATED WORK

- MEGHA [Prasad et al, APPLC 2012]:
  - Compiles a subset of MATLAB to CUDA
- HLLC/ParaM
  - Source-to-source [Shei et al, ICS 2011]
    - Outputs MATLAB with GPUmat API calls
  - Alternative approach: [Shei et al, INTERACT 2011]
    - Outputs MATLAB with calls to C++ and CUDA.
- Our approach: MATLAB to C + OpenCL

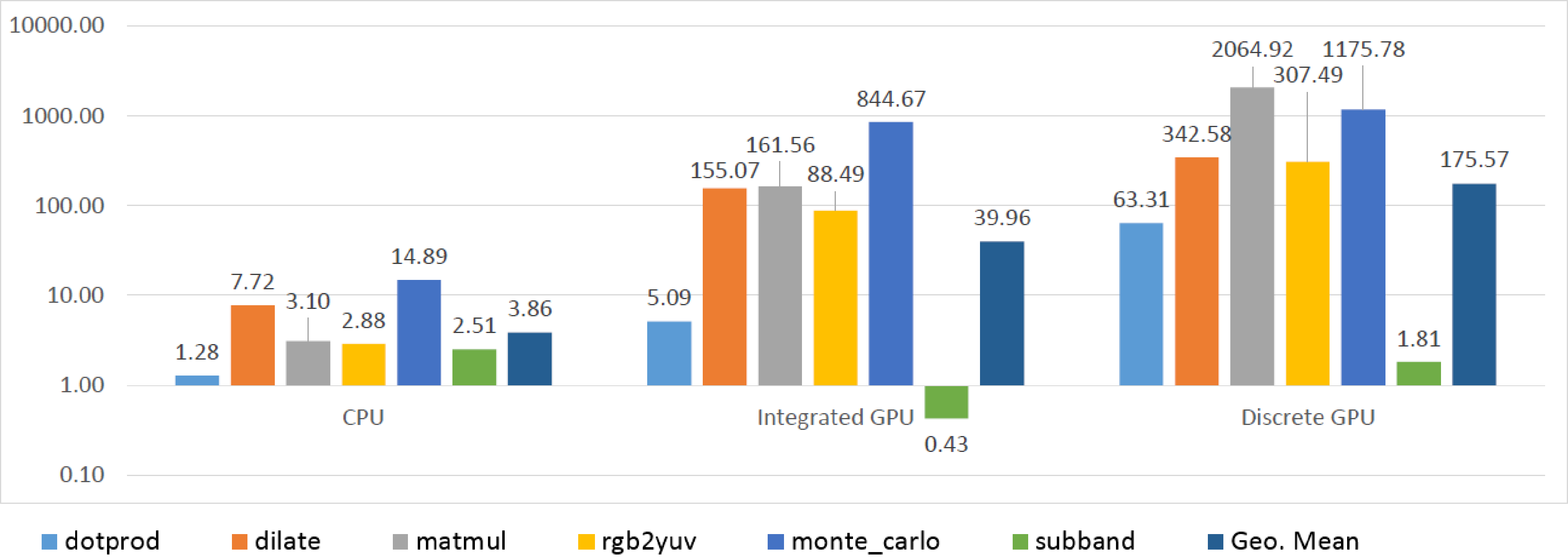
RESULTS – MATISSE C VS MATLAB



[1] A. Prasad, J. Anantpur, and R. Govindarajan, "Automatic compilation of MATLAB programs for synergistic execution on heterogeneous processors," in *ACM Sigplan Notices*, 2011, vol. 46, pp. 152–163.

RESULTS – KERNEL TIME

- Same computer, Kernel time **only** (no data transfers, no C segments)



MATLAB GPU APIS

- MathWorks Parallel Computing Toolbox:
  - CUDA API for MATLAB
  - Official, supported
- GPUmat
  - Open-source
  - CUDA API
  - Open-source, last update on 2012
- Jacket
  - CUDA or OpenCL
  - Discontinued

LIMITATIONS

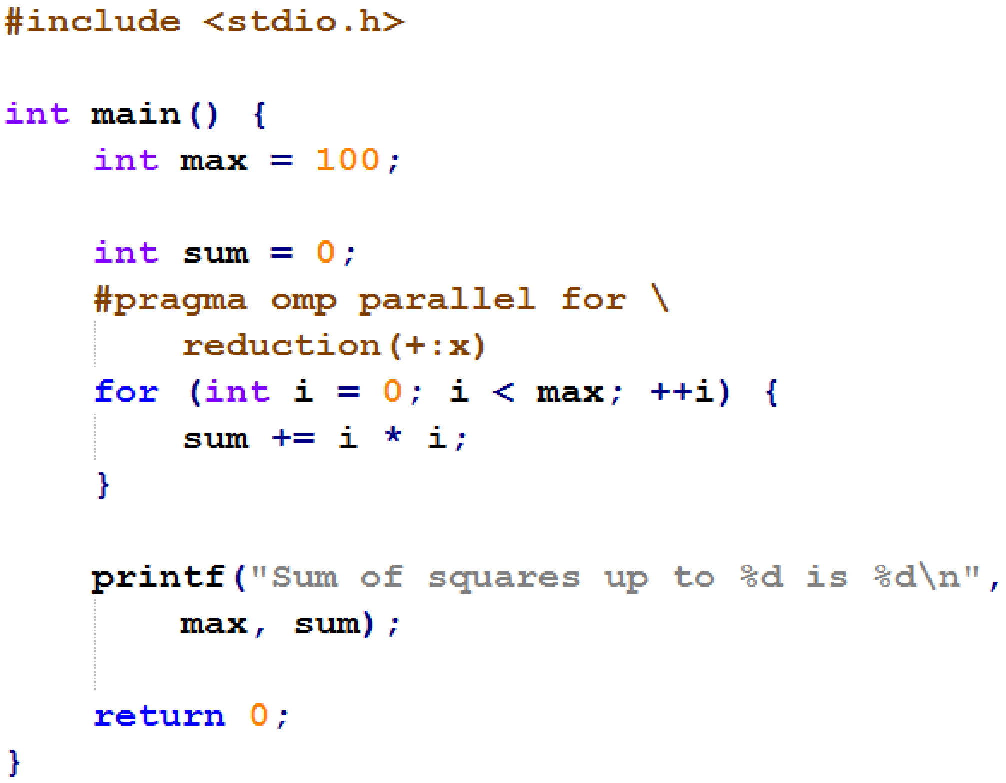
- OpenCL back-end introduced too early in the tool-chain
  - Does not take advantage of current C transformations (e.g., element-wise)
  - Only a small subset of functions are supported within a parallel block
- Odroid performance is poor

MATMUL

- Idiomatic: A = B * C;
- Simple and slow, three nested loops
- Fine-tuned with directives: separate file

OPENMP

- OpenMP: Standard for C, C++ and FORTRAN.
- Very CPU-centric.
- Code is annotated with directives.
- Compilers automatically generate the code to launch threads.



```
#include <stdio.h>

int main() {
    int max = 100;

    int sum = 0;
    #pragma omp parallel for \
        reduction(+:x)
    for (int i = 0; i < max; ++i) {
        sum += i * i;
    }

    printf("Sum of squares up to %d is %d\n",
        max, sum);

    return 0;
}
```

HOW LONG DOES MATLAB TAKE (EXAMPLE)

- Monte Carlo Option Pricing:
  - MATLAB: For 100 iterations, 12 seconds
  - MATLAB: For 1000 iterations, 113 seconds
  - MATISSE C: For 10000 iterations, takes 24 seconds
  - MATISSE OpenCL: For 10000 iterations, takes 0.02 seconds