IPMACC: Translating OpenACC API to OpenCL

University of Victoria

1 Why OpenACC?

Lower development Effort

```
#pragma acc data copy(a[0:LEN*LEN],b[0:LEN*LEN],c[0:LEN*LEN])
#pragma acc kernels
#pragma acc loop independent
for(i=0; i<LEN; ++i){
  #pragma acc loop independent
  for(j=0; j<LEN; ++j){
    float sum=0;
    for(l=0; l<LEN; ++l){
      sum += a[i*LEN+l]*b[l*LEN+j];
    }
    c[i*LEN+j]=sum;
  }
}
```

13 lines **OpenACC matrix-matrix multiplication**

```
int main(){
 ...
 bytes=(LEN*LEN)*sizeof(float);
 d_a=(void*)clCreateBuffer(clctx, CL_MEM_READ_WRITE, …
 d_b=(void*)clCreateBuffer(clctx, CL_MEM_READ_WRITE, …
 d_c=(void*)clCreateBuffer(clctx, CL_MEM_READ_WRITE, …
 clEnqueueWriteBuffer(clcmdqueue, (cl_mem)d_a, …
 clEnqueueWriteBuffer(clcmdqueue, (cl_mem)d_b, …
 const char* kernelSource0 ="__kernel void\\
 matrixMul(__global float * a,__global float * c,__…
 int i=get_global_id(0);\\
 int j=get_global_id(1);\\
 float sum = 0;\\
 for(l = 0; l < LEN; l++) sum += a [i * LEN + l] …
 c [i * LEN + j] = sum;\\
 }";
 clpgm0=clCreateProgramWithSource(clctx, 1, &kernelSource0, …
 clerr=clBuildProgram(clpgm0, 0, NULL, " ", NULL, NULL);
 clkern0=clCreateKernel(clpgm0, "matrixMul", &clerr);
 clerr=clSetKernelArg(clkern0, 0, sizeof(cl_mem), …
 clerr=clSetKernelArg(clkern0, 1, sizeof(cl_mem), …
 clerr=clSetKernelArg(clkern0, 2, sizeof(cl_mem), …
 clerr=clSetKernelArg(clkern0, 3, sizeof(int), …
 size_t clgridDim[2]={LEN,LEN};
 size_t clblockDim[2]={16,16};
 size_t cloffsets[2]={0,0};
 uint clndims=2;
 clerr=clEnqueueNDRangeKernel(clcmdqueue, clkern0, …
  clgridDim, clblockDim, 0, NULL, NULL);
 clEnqueueReadBuffer(clcmdqueue, (cl_mem)d_c, CL_TRUE, …
 ...
}
```

28 lines **OpenCL matrix-matrix multiplication**

Availability

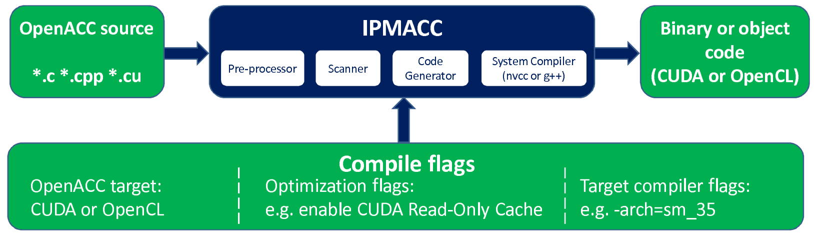
Freely available on Github:

https://github.com/lashgar/ipmacc

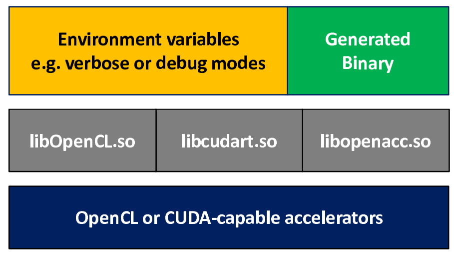
Abstract

In this paper, we introduce IPMACC a framework for executing OpenACC for C applications over OpenCL runtime. We use over framework to compare performance of OpenACC and OpenCL. OpenACC API abstractions remove the low-level control from programmers' hand. To understand the low-level OpenCL optimizations that are not applicable in OpenACC, we compare highly-optimized OpenCL and OpenACC versions of a wide set of benchmarks. We show that under the investigated benchmarks, exploiting scratchpad memory as a fast-communication link is the most important optimization that is not applicable in OpenACC. We also introduce a micro-benchmarking suit to investigate the overhead of various OpenACC operations. We compare our framework to a previous open source OpenACC compiler in various aspects.

2 IPMACC Compile Flow



3 IPMACC Runtime

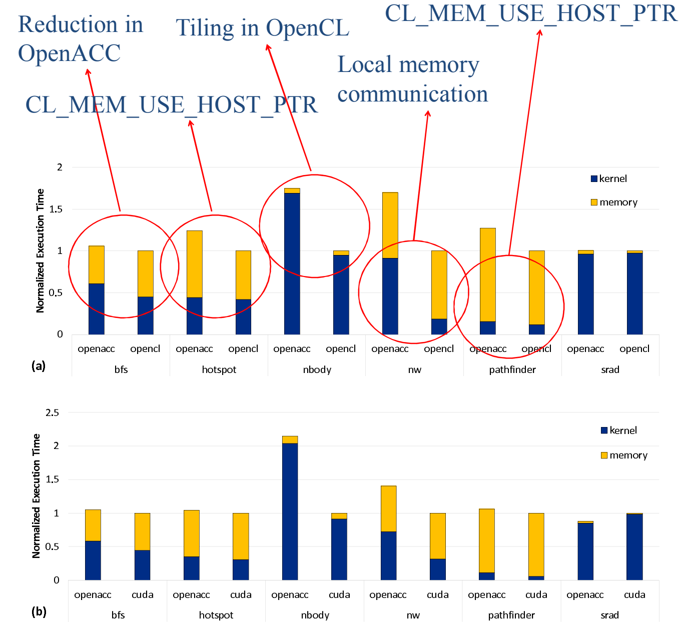


IPMACC features:

- Based on OpenACC 2.0
- User-define data types
- Procedure call in the region
- Supports kernels, loop, data, enter, and exit directives
- Extensible to support more backends

4 OpenACC Performance

OpenACC performs close to OpenCL/CUDA implementations unless OpenCL/CUDA implementations use a different algorithm.

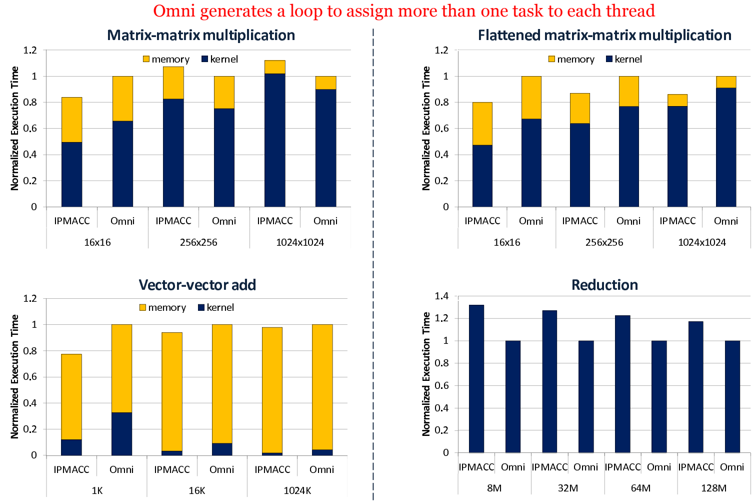


Performance comparison of (a) OpenCL vs OpenACC and (b) CUDA vs OpenACC under Rodinia Benchmark Suite.

5 Framework Efficiency

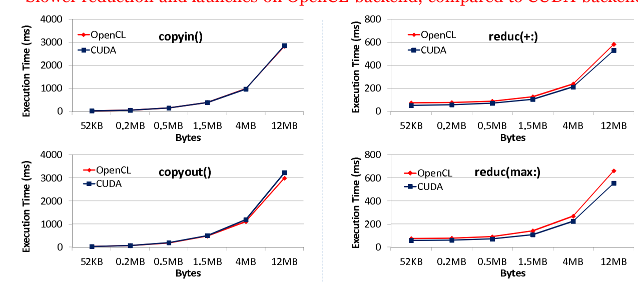
Compare to Omni Compiler

Omni loop-to-thread mapping differs from IPMACC as Omni generates a loop to assign more than one task to each thread



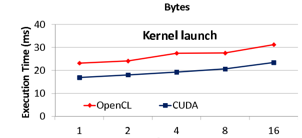
6 Overhead of Operations

Slower reduction and launches on OpenCL-backend, compared to CUDA-backend



Comparing the overhead of copyin, copyout, and reduction clauses and kernel launch.

OpenCL platform: NVIDIA CUDA 5.5
Hardware: NVIDIA Tesla K20c



Authors

Ahmad Lashgar
lashgar@uvic.ca ece.uvic.ca/~lashgar

Alireza Majidi
a.majidi@tamu.edu parasol.tamu.edu/people/alireza/

Amirali Baniasadi
amirali@ece.uvic.ca ece.uvic.ca/~amiralib/