Best GPU Code Practices Combining OpenACC, CUDA, and OmpSs

Pau Farré
Antonio J. Peña

Munich, Oct. 12 2017
Marenosrump 4

• 13.7 PetaFlop/s

• General Purpose Computing
  ▪ 3400 nodes of Xeon, 11 PF/s

• Emerging Technologies
  ▪ Power 9 + Pascal → 1.5 PF/s
  ▪ Knights Landing and Knights Hill → 0.5 PF/s
  ▪ 64bit ARMv8 → 0.5 PF/s
Mission of BSC Scientific Departments

COMPUTER SCIENCES
To influence the way machines are built, programmed and used: programming models, performance tools, Big Data, computer architecture, energy efficiency

EARTH SCIENCES
To develop and implement global and regional state-of-the-art models for short-term air quality forecast and long-term climate applications

LIFE SCIENCES
To understand living organisms by means of theoretical and computational methods (molecular modeling, genomics, proteomics)

CASE
To develop scientific and engineering software to efficiently exploit supercomputing capabilities (biomedical, geophysics, atmospheric, energy, social and economic simulations)
BSC Training on European Level - PATC

PRACE Advanced Training Centers

The PRACE, designated 6 Advanced Training Centers:

• **Barcelona Supercomputing Center (Spain)**
• CINECA Consorzio Interuniversitario (Italy),
• CSC - IT Center for Science Ltd (Finland),
• EPCC at the University of Edinburgh (UK),
• Gauss Centre for Supercomputing (Germany) and Maison de la Simulation (France).

Mission of PATCs

Carry out and coordinate training and education activities that foster the efficient usage of the infrastructure available through PRACE.
BSC & The Global IT Industry 2016

- IBM-BSC Deep Learning Center
- NVIDIA GPU Center of Excellence
- BSC-Microsoft Research Centre
- Intel-BSC Exascale Lab
Projects with the Energy Industry

Repsol-BSC Research Center

Research into advanced technologies for the exploration of hydrocarbons, subterranean and subsea reserve modelling and fluid flows
NVIDIA Award to BSC/UPC (since 2011)

R&D around GPU Computing (currently ~10 core collaborators)
  – Architecture, Programming Models, Libraries, Applications, Porting

Education, Training, Dissemination (free registration)
  – PUMPS Summer School – advanced CUDA mainly
  – PRACE Adv. Training Center courses on Introduction to CUDA & OpenACC
  – Severo Ochoa Seminars on Deep Learning & Image/Video Processing
  – *Always open to research collaborations, internships, advising, hiring*

antonio.pena@bsc.es
Introductions

Pau Farre, Jr. Engineer
– GCoE Core Team
– GPU porting and optimization specialist
– Did most of the hard work for this lab

Antonio J. Peña, Sr. Researcher
– Manager of the GCoE
– Juan de la Cierva Fellow – Prospective Marie Curie Fellow
– Activity Leader “Accelerators and Communications for HPC”
  – The one to blame if anything goes wrong

antonio.pena@bsc.es
pau.farre@bsc.es
Introduction: Programming Models for GPU Computing

CUDA (Compute Unified Device Architecture)
- **Runtime** & Driver APIs (high-level / low-level)
- Specific for NVIDIA GPUs: best performance & control

OpenACC (Open Accelerators)
- Open Standard
- Higher-level, pragma-based
- Aiming at portability – heterogeneous hardware
- For NVIDIA GPUs, implemented on top of CUDA

OpenCL (Open Computing Language)
- Open Standard
- Low-level – similar to CUDA Driver API
- Multi-target, portable*

(Intentionally leaving out weird stuff like CG, OpenGL, …)
Motivation: Coding Productivity & Performance

- **CUDA**
- **OpenACC**
- **OpenACC + CUDA**
- **OmpSs + CUDA**
- **OmpSs + OpenACC**
  - High-level, task-based, pragma-based
  - Target accelerators co-owned by the software developer

Don’t get me wrong: CUDA delivers awesome coding productivity w.r.t., e.g., OpenGL, but I only want to use 3 (easy) colors here. Please interpret colors as relative to each other.

OpenACC may well deliver more than the performance you *need*. However, we have the lowest control on performance w.r.t. the discussed alternatives.
HANDS-ON
LAB CONNECTION INSTRUCTIONS - Part 1

Go to nvlabs.qwiklab.com

Sign in or create an account

Check for Access Codes (each day):
- Click My Account
- Click Credits & Subscriptions

If no Access Codes, ask for paper one from TA.

Please tear in half once used

An Access Code is needed to start the lab

WIFI SSID: GTC_Hands_On
Password: HandsOnGpu
LAB CONNECTION INSTRUCTIONS - Part 2

1. Click Qwiklabs in upper-left

2. Select GTC2017 Class

3. Find lab and click on it

4. Click on Select

5. Click Start Lab

WIFI SSID: GTC_Hands_On
Password: HandsOnGpu
Steps to Parallelize with OpenACC

1. Identify Parallelism
   ○ Using a CPU profiling tool (example: `nvprof --cpu-profiling on`)

2. Express Parallelism
   ○ Declare parallel regions with directives

3. Express Data Locality
   ○ Help OpenACC figure out how to manage data

4. Optimize
   ○ Using `nvprof` & Nvidia visual profiler
● Analyzes physical properties of the subsoil from seismic measures
● Elastic wave propagator + linearly elastic stress-strain relationships
  – Six different stress components
  – Finite differences (FD) method with a Fully Staggered Grid (FSG)

Base code developed by the BSC Repsol Team
FWI Parallelization – OpenACC/CUDA #6: Results

- Our optimized CUDA Kernels have better performance than the OpenACC
OmpSs + CUDA / OpenACC
Sequential control flow
- Defines a single address space
- Executes sequential code that
  - Can spawn/instantiate tasks that will be executed sometime in the future
  - Can stall/wait for tasks

Tasks annotated with directionality clauses
- in, out, inout
- Used
  - To build dependences among tasks
  - For main to wait for data to be produced
- Basis for memory management functionalities (replication, locality, movement, …)
  - Copy clauses

Sequential equivalence (~)
void Cholesky( float *A[NT][NT] ) {
    int i, j, k;
    for (k=0; k<NT; k++) {
        spotrf (A[k*NT+k]) ;
        for (i=k+1; i<NT; i++) {
            strm (A[k][k], A[k][i]);
        }
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++) {
                sgemm( A[k][i], A[k][j], A[j][i]);
            }
            ssyrk (A[k][i], A[i][i]);
        }
    }
}
OmpSs: ... with Directionality Annotations ...

```c
void Cholesky( float *A[NT][NT] ) {
    int i, j, k;
    for (k=0; k<NT; k++) {
        #pragma omp task inout (A[k][k])
        spotrf (A[k][k]) ;
        for (i=k+1; i<NT; i++) {
            #pragma omp task in (A[k][k]) inout (A[k][i])
            strsm (A[k][k], A[k][i]);
        }
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++) {
                #pragma omp task in (A[k][i], A[k][j]) inout (A[j][i])
                sgemm( A[k][i], A[k][j], A[j][i]);
            }
            #pragma omp task in (A[k][i]) inout (A[i][i])
            ssyrk (A[k][i], A[i][i]);
        }
    }
}
```
OmpSs: … that Happens to Execute in Parallel

```c
void Cholesky( float *A[NT][NT] ) {
    int i, j, k;
    for (k=0; k<NT; k++) {
        #pragma omp task inout (A[k][k])
        spotrf (A[k][k]);
        for (i=k+1; i<NT; i++) {
            #pragma omp task in (A[k][k]) inout (A[k][i])
            strsm (A[k][k], A[k][i]);
        }
        for (i=k+1; i<NT; i++) {
            for (j=k+1; j<i; j++) {
                #pragma omp task in (A[k][i], A[k][j]) inout
                sgemm( A[k][i], A[k][j], A[j][i]);
            }
            #pragma omp task in (A[k][i]) inout (A[i][i])
            ssyrk (A[k][i], A[i][i]);
        }
    }
}
```

Decouple how we write/think (sequential) from how it is executed
#include <kernel.h>

int main(int argc, char *argv[]) {
    float a=5, x[N], y[N];
    // Initialize values
    for (int i=0; i<N; ++i)
        x[i] = y[i] = i;
    // Compute saxpy algorithm (1 task)
    saxpy(N, a, x, y);
    #pragma omp taskwait
    // Check results
    for (int i=0; i<N; ++i)
        if (y[i]!=a*i+i) perror("Error\n");
    message("Results are correct\n");
}

void saxpy(int n, float a, float *X, float *Y) {
    for (int i=0; i<n; ++i)
        Y[i] = X[i] * a + Y[i];
}

#pragma omp target device(cuda) copy_deps
#pragma omp task in([n]x) inout([n]y)
void saxpy(int n, float a, float* x, float* y);

void saxpy(int n, float a, float *X, float *Y) {
    for (int i=0; i<n; ++i)
        Y[i] = X[i] * a + Y[i];
}

#pragma omp target device(cuda) copy_deps ndrange(1,n,128)
#pragma omp task in([n]x) inout([n]y)
__global__ void saxpy(int n, float a, float* x, float* y);

__global__ void saxpy(int n, float a, float* x, float* y) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i < n) y[i] = a * x[i] + y[i];
}
OmpSs + OpenACC: General Idea

- Taskify all your application in a data-flow manner
  - Process kernels are just a type of tasks executed inside a GPU
- The OmpSs runtime manages automatically the use of streams & memory transfers
- OpenACC directives are used to generate all GPU kernels that will be treated as a CUDA tasks by OmpSs
- Greatest coding productivity for accelerators!
  - But OpenACC kernels might perform lower than fine-tuned CUDA
#pragma omp target(openacc)
#pragma omp task in(rho, sxptr, syptr, szptr) inout(vptr)
#pragma acc parallel loop deviceptr(rho, sxptr, syptr, szptr, vptr)
for (int y=ny0; y < nyf; y++) {
    for (int x=nx0; x < nxf; x++) {
        for (int z=nz0; z < nzf; z++) {
            ...code...
        }
    }
}

Not released yet
FWI Parallelization – OmpSs/OpenACC - Results

- OmpSs/OpenACC performance is similar to OpenACC

![FWI Speedups Diagram](image-url)
Your Turn!

- Open http://github.com/Hopobcn/FWI
- Follow step-by-step instructions @ GTC2017eu.md
Thank you!

For further information please contact
antonio.pena@bsc.es