OmpCloud: Bridging the Gap between OpenMP and Cloud Computing

Hervé Yviquel, **Marcio Pereira** and Guido Araújo

University of Campinas (UNICAMP), Brazil

UNICAMP

FAPESP

A bit of background

- Guido Araujo, PhD Princeton University
- Marcio Pereira, PhD UNICAMP/UAlberta
- Hervé Yviquel, PhD University of Rennes 1

- Research focus on Compiling Technology
  - Thread-level speculation for loops
  - Loop tiling and vectorization
  - Cloud parallelization techniques for scientific workloads
  - Parallel programming models (MapReduce, OpenMP)
  - Heterogeneous computing (GPUs, DSPs, FPGAs)

My current work

- Compiling and Optimizing OpenMP 4.X Programs to OpenCL and SPIR
  - To be presented in IWOMP on Thursday
  - First to convert OpenMP 4.5 to OpenCL/SPIR
  - Uses loop tiling and vectorization
  - Based on Polyhedral techniques

The Cloud as a Computing Resource

Several cloud providers

Amazon Web Service, Microsoft Azure, etc.

Private cloud infrastructure

Large datacenters

Almost infinite storage

Massively parallel processing capabilities

Flexible usage

Accessible to anyone with internet

Quick availability of the resources

The Cloud as a Solution

Ultimate solution for “The Rising of Big Data”

- Social media (Facebook, Twitter, etc.)
- Multimedia (Netflix, Spotify, etc.)

Useful for other application domains

- Scientific applications (HPC)
- Mobile applications
- Internet-of-Thing (IoT)

BUT… HOW TO PROGRAM THE CLOUD ?

How to program the Cloud?

Application domain

- Small application using cloud services (mobile, IoT,...)

- Big-data
- HPC

Programming model

- Python (or any language) + Cloud provider's SDK

  Easy learning

- Map-Reduce (and Spark)

  High-level
  Fault tolerance

- MPI

  Low-level programming
  Very efficient

HOW ABOUT SOMETHING IN BETWEEN ?

Are you a programming expert ?

Writing parallel programs is complex

- Not so natural...

Integrating the cloud in your application might be complex

- Hybrid execution (running in the cloud and locally)
- Require various programming languages

Let’s make it simpler!

OpenMP

Well-known API for developing parallel application

- Directive-based programming
  - Made to be simple **and** no need to rewrite the code
- **But** assume shared-memory architecture

```
void MatMul(float *A, float *B, float *C) {
  #pragma omp parallel for
  for(int i=0; i<N; ++i)
     for(int j=0; j<N; ++j)
        C[i*N + j] = 0;
        for(int k=0; k<N; ++k)
           C[i*N + j] += A[i*N + k] * B[k*N + j];
}
```

OpenMP Accelerator Model

Extension for programming accelerators (v4.0+)

- Designed for local accelerators (e.g. GPU)
- Host-target architecture model

```
void MatMul(float *A, float *B, float *C) {
  #pragma omp target device(GPU) \
                     map(to: A[:N*N], B[:N*N]) \
                     map(from: C[:N*N])
  #pragma omp parallel for
  for(int i=0; i<N; ++i)
     for(int j=0; j<N; ++j)
        C[i*N + j] = 0;
        for(int k=0; k<N; ++k)
           C[i*N + j] += A[i*N + k] * B[k*N + j];
}
```

The Cloud as an Accelerator

Let's be brave!

- Introduce the cloud as an OpenMP offloading device
- Just another accelerator available in your computer

```
int MatMul(float *A, float *B, float *C) {
  #pragma omp target device(CLOUD) \
                     map(to: A[:N*N], B[:N*N]) \
                     map(from: C[:N*N])
  #pragma omp parallel for
  for(int i=0; i<N; ++i)
     for(int j=0; j<N; ++j)
        C[i*N + j] = 0;
        for(int k=0; k<N; ++k)
           C[i*N + j] += A[i*N + k] * B[k*N + j];
}
```

OpenMP + Cloud = **OmpCloud**

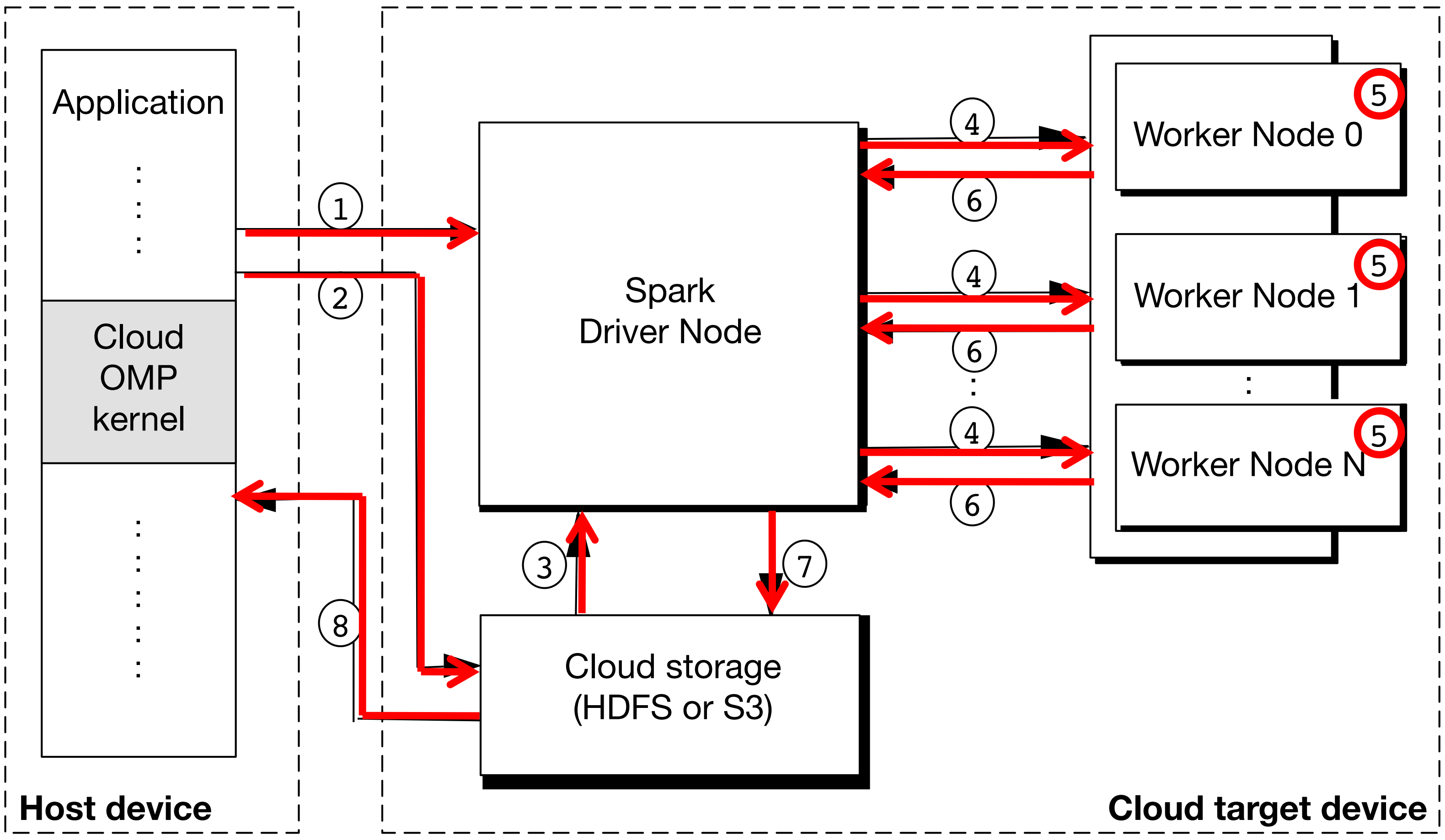
Development environment for cloud offloading

- Open-source (available on Github)
- Rely on **custom LLVM** for *host device*
  - Clang compiler
  - OpenMP library
- Rely on **Apache Spark** for *target device* (cloud)

Cloud Offloading Workflow (1)

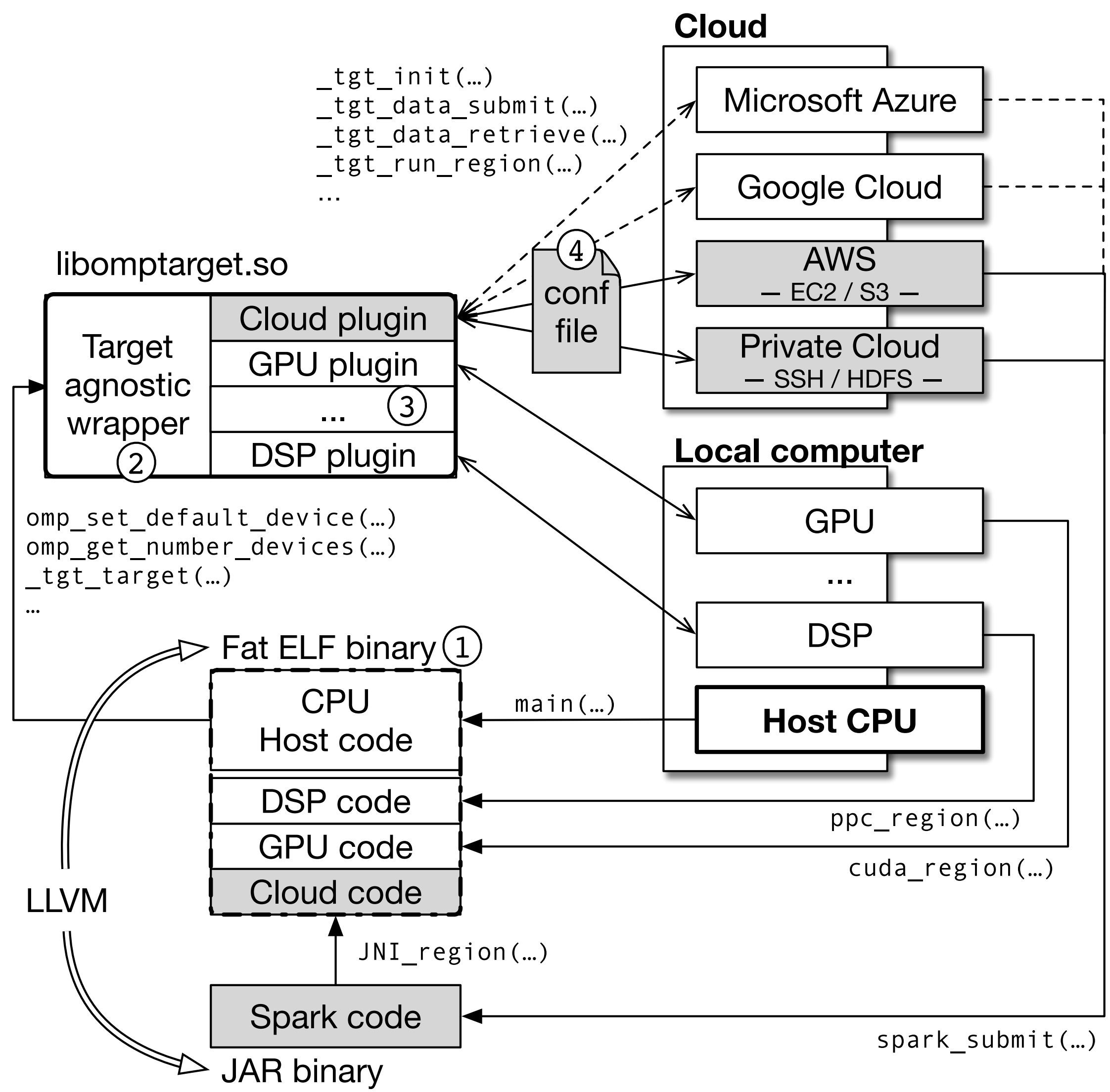
1. Describe the application using OpenMP
2. Compile it with our custom Clang
3. Instantiate a Spark cluster in your favorite cloud provider (e.g. Amazon Web Service)
4. Configure the OmpCloud runtime with the credentials for accessing the cluster in the cloud
5. Run the application !

Cloud Offloading Workflow (2)



Modular host-target implementation

1. *Fat binary generated by LLVM*
2. *Target-agnostic offloading wrapper*
3. *Target-specific offloading plug-ins*
4. *Cloud configuration file*



Cloud Portability

No need to recompile your application. The code is portable for all spark-based cloud device

```
[AzureProvider]
Cluster=clusterName
Container=containerName
StorageAccount=storageName
StorageAccessKey=XXXXX

[Spark]
User=sshuser
WorkingDir=/workspace/
(...)
```

configuration.ini

Provider-specific options

Common options

Data Partitioning

Mapping the data block to the cluster node using it

Essential because…

Reduce communication overhead in distributed systems

But …

Cannot be determined statically in general case

OpenMP does not provide mechanism to describe it

Let’s make it possible!

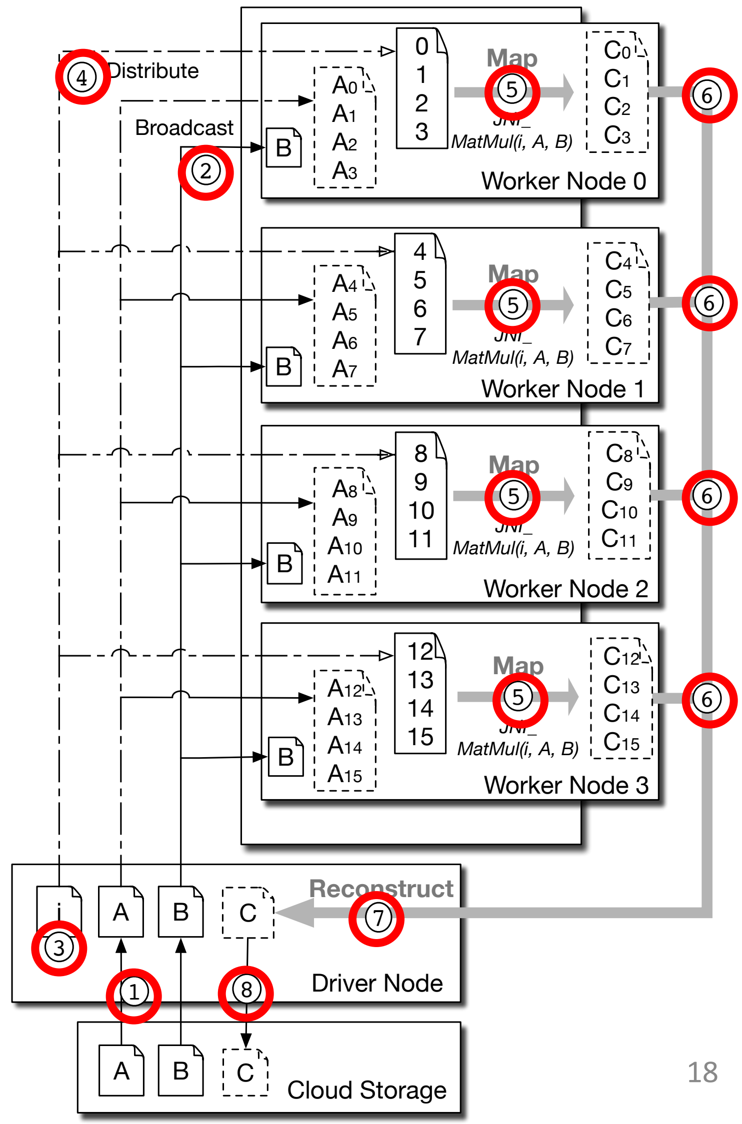
Extending OpenMP for Data Partitioning

Partitions are described using *data map* clauses

```
void MatMul(float *A, float *B, float *C) {
  #pragma omp target device(CLOUD) \
                     map(to: A[:N*N], B[:N*N]) \
                     map(from: C[:N*N])
  #pragma omp parallel for
  for(int i=0; i<N; ++i)
     #pragma omp data map(to: A[i*N:(i+1)*N]) \
                      map(from: C[i*N:(i+1)*N])
     for(int j=0; j<N; ++j)
        C[i*N + j] = 0;
        for(int k=0; k<N; ++k)
           C[i*N + j] += A[i*N + k] * B[k*N + j];
}
```

Matching Spark Execution Model

1. Read inputs (A and B) from the cloud storage
2. Broadcast unpartitioned B
3. Generate the set of all values taken by the loop index
4. Distribute A and i
5. Map loop body function to the values of the loop index
6. Send back parts of C
7. Reconstruct final version of C
8. Write C to the cloud storage



Wanna see the generated Spark (pseudo)code ?

```
// Read inputs as Array[Byte]
val A = DecompressFromStorage(0)
val B = DecompressFromStorage(1)

// Generate distributed list of tiled-loop index values
val indexes = (0 to N-1).toRDD

// Partition data and distribute loop iterations
val results = indexes.map{ i => (i,
    JNI_loopbody(i, A.slice(i*N*4, ((i+1)*N*4), B)) }

// Reconstruct the output
val C = new Array[Byte](N*N)
results.foreach{(i,Ci) =>
    Ci.copyToArray(C, i*N*4, (i+1)*N*4)}

// Write the result back
CompressToStorage(3, C)
```

```
* Please note that 4 = sizeof(float)
```

Optimizing the Granularity

- Large overhead possible when
  - **Number of iterations “N” >> Number of cores “C”**
  - Because of JNI calls and data partitioning
- Loop tiling optimization
  - Blocking size ⌊N/C⌋ defined at runtime (parameter)
  - User-partitioning automatically adjusted

```
// Tiled parallel for

for ii=0 to N-1 by ⌊N/C⌋ do
  for i=ii to min(ii+⌊N/C⌋-1,N-1) do
    // loop body
  end for
end for
```

Experiments

- Realistic test case
  - Host → A laptop connected from UNICAMP, Brazil
  - Target → AWS datacenter in US (North Virginia)
- Spark Cluster of 1 driver and 16 worker nodes
  - EC2 instances of type *c3.8xlarge* (16 cores - 60GB of RAM)
  - Ubuntu 14.04 with Spark 2.1.0
- Using a set of well-known benchmarks

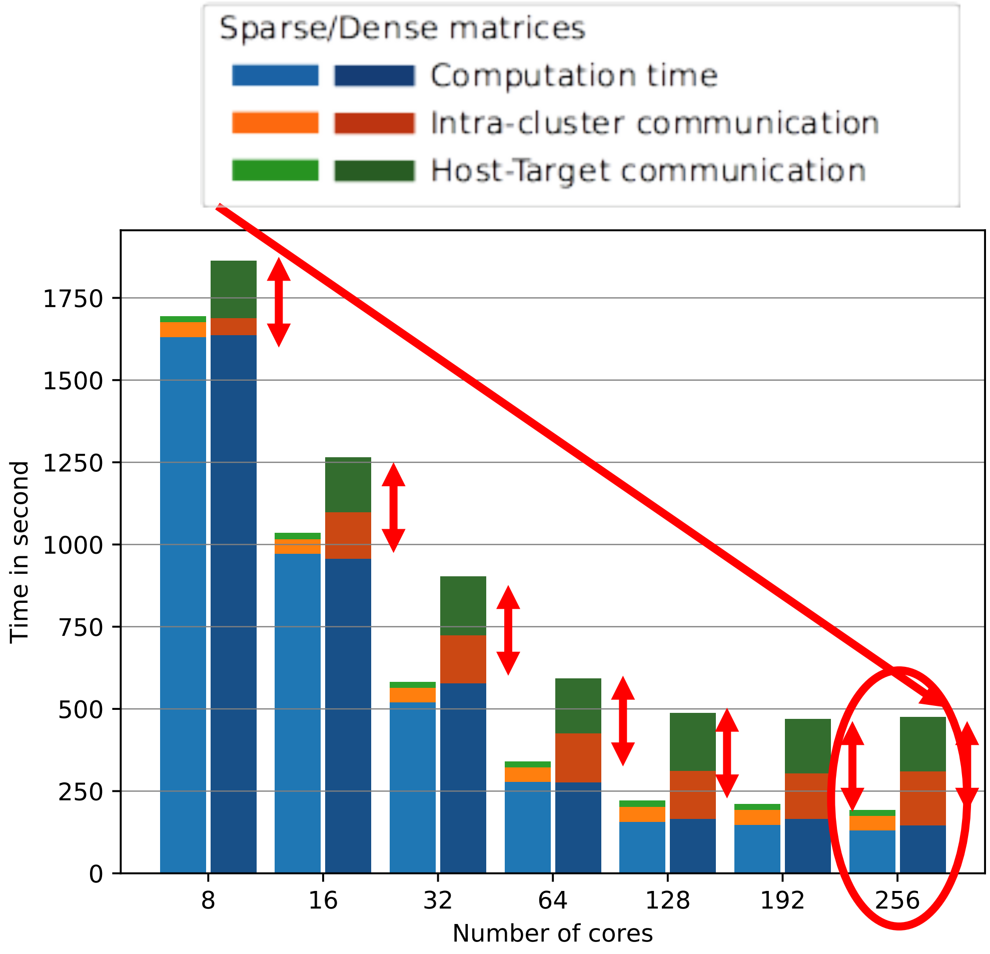
Matrix Multiplication

Matrices 16000x16000
- 1GB / floating-point

Execution time
- Sequential = 3.5h
- 256 cores = 3-8min

Increasing speedups
- 27x/68x on 256 cores

Communication overhead
- Data-type matter



Limitations of the Programming Model

Code regions offloaded to the cloud

do support

- *parallel for* with nested loops
- *reduction clause*

do not support

- *atomic*, *flush*, *barrier*, *critical*, or *master*

will support

- blocks of sequential code
- *parallel for* inside a sequential loop

Cluster programming made easy!

Sometimes, cloud offloading is **not adapted**

- No need to run from local computer
- Host-Target communications are expensive

One can run the app **directly** from the Spark driver node

- Connect with SSH; transfer your app; configure OmpCloud runtime; and run it !!
- Communications between the binary and Spark are handled seamlessly using local file

Easy way to program cluster from C/C++

Conclusion (1)

Simple parallel programming model

- C/C++ and OpenMP directives
- No need to rewrite your code

New development environment

- Offload computation to the cloud
- Integrate the cloud in local application
- Program clusters
- Support any cloud provider

Conclusion (2)

Early experiments

- Demonstrate viability on benchmarks
- Already showed promising performance

Future works

- Offload *Blender* rendering to cloud cluster
- Machine learning / Face recognition

Thanks!
Obrigado!
Merci!

Any questions ?

Check our website at ompcloud.org
Contact: herve.yviquel@ic.unicamp.br

UNICAMP FAPESP