# HamaPipes

**Hama Pipes** is equivalent to **Hadoop Pipes** and offers the possibility to use Hama with C/C++.

## Installation

You can compile Hama Pipes by executing the following commands:

```
cd $HAMA_HOME/c++/utils
./configure
make install

cd $HAMA_HOME/c++/pipes
./configure
make install
```

## Interface

Hama Pipes provides the following methods for C/C++ integration: (similar to the BSPModel)

| Function | Description |
|---|---|
| `void sendMessage(const string& peerName, const string& msg)` | Send a message to another peer. Messages sent by this method are not guaranteed to be received in a sent order. |
| `string& getCurrentMessage()` | Returns a message from the peer's received messages queue (a FIFO). |
| `int getNumCurrentMessages()` | Returns the number of messages in the peer's received messages queue. |
| `void sync()` | Starts the barrier synchronization and sends all the messages in the outgoing message queues to the corresponding remote peers. |
| `long getSuperstepCount()` | Returns the count of current super-step. |
| `string& getPeerName()` | Returns the name of this peer in the format "hostname:port". |
| `string& getPeerName(int index)` | Returns the name of n-th peer from sorted array by name. |
| `int getPeerIndex()` | Returns the index of this peer from sorted array by name. |
| `vector<string> getAllPeerNames()` | Returns the names of all the peers executing tasks from the same job (including this peer). |
| `int getNumPeers()` | Returns the number of peers. |
| `void clear()` | Clears all queues entries. |
| `void write(const string& key, const string& value)` | Writes a key/value pair to the output collector. |
| `bool readNext(string& key, string& value)` | Deserializes the next input key value into the given objects. |
| `void reopenInput()` | Closes the input and opens it right away, so that the file pointer is at the beginning again. |

The following additional methods support access to SequenceFiles under C/C++:

| Function | Description |
|---|---|
| `sequenceFileOpen(const string& path, const string& option, const string& keyType, const string& valueType)` | Opens a SequenceFile with option "r" or "w", key/value type and returns the corresponding fileID. |
| `bool sequenceFileReadNext(int fileID, string& key, string& value)` | Reads the next key/value pair from the SequenceFile. |
| `bool sequenceFileAppend(int fileID, const string& key, const string& value)` | Appends the next key/value pair to the SequenceFile. |
| `bool sequenceFileClose(int fileID)` | Closes a SequenceFile. |

## C++ BSP example

Finally here is the Pi Estimator example implemented with Hama Pipes:

```
#include "hama/Pipes.hh"
#include "hama/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"
```

```
#include <time.h>
#include <math.h>
#include <string>
#include <iostream>
#include <cstdlib>

using std::string;
using std::cout;

using HamaPipes::BSP;
using HamaPipes::BSPContext;
using namespace HadoopUtils;

class PiCalculationBSP: public BSP {
private:
    string masterTask;
    int iterations;
public:
  PiCalculationBSP(BSPContext& context) {
      iterations = 10000;
  }

  inline double closed_interval_rand(double x0, double x1) {
    return x0 + (x1 - x0) * rand() / ((double) RAND_MAX);
  }

  void bsp(BSPContext& context) {

    // initialize random seed
    srand(time(NULL));

    int in = 0;
    for (int i = 0; i < iterations; i++) {
      //rand() -> greater than or equal to 0.0 and less than 1.0.
      double x = 2.0 * closed_interval_rand(0, 1) - 1.0;
      double y = 2.0 * closed_interval_rand(0, 1) - 1.0;
      if (sqrt(x * x + y * y) < 1.0) {
        in++;
      }
    }

    double data = 4.0 * in / iterations;

    context.sendMessage(masterTask, toString(data));
    context.sync();
  }

  void setup(BSPContext& context) {
    // Choose one as a master
    masterTask = context.getPeerName(context.getNumPeers() / 2);
  }

  void cleanup(BSPContext& context) {
    if (context.getPeerName().compare(masterTask)==0) {
      double pi = 0.0;
      int msgCount = context.getNumCurrentMessages();
      string received;
      for (int i=0; i<msgCount; i++) {
        string received = context.getCurrentMessage();
        pi += toDouble(received);
      }

      pi = pi / msgCount; //msgCount = numPeers
      context.write("Estimated value of PI is", toString(pi));
    }
  }
};

int main(int argc, char *argv[]) {
  return HamaPipes::runTask(HamaPipes::TemplateFactory<PiCalculationBSP>());
}
```

Makefile for this example:

```
CC = g++
CPPFLAGS = -m64 -I$(HAMA_HOME)/c++/install/include

PiCalculation: PiCalculation.cc
	$(CC) $(CPPFLAGS) $< -L$(HAMA_HOME)/c++/install/lib -lhamapipes -lhadooputils -lcrypto -lpthread -g -O2
-o $@

clean:
	rm -f PiCalculation
```

The corresponding job configuration `PiCalculation_job.xml` looks like that:

```
<?xml version="1.0"?>
<configuration>
  <property>
    // Set the binary path on DFS
    <name>hama.pipes.executable</name>
    <value>bin/PiCalculation</value>
  </property>
  <property>
    <name>hama.pipes.java.recordreader</name>
    <value>true</value>
  </property>
  <property>
    <name>hama.pipes.java.recordwriter</name>
    <value>true</value>
  </property>
  <property>
    <name>bsp.input.format.class</name>
    <value>org.apache.hama.bsp.NullInputFormat</value>
  </property>
  <property>
    <name>bsp.output.format.class</name>
    <value>org.apache.hama.bsp.TextOutputFormat</value>
  </property>
  <property>
    <name>hama.pipes.logging</name>
    <value>false</value>
  </property>
  <property>
    <name>bsp.peers.num</name>
    <value>10</value>
  </property>
</configuration>
```

Finally the PiCalculation example can be submitted with these commands:

```
# delete output dir
hadoop dfs -rmr output/PiCalculation
# copy piCalculation binary to HDFS
hadoop dfs -rmr bin/PiCalculation
hadoop dfs -put PiCalculation bin/PiCalculation
# submit job
hama pipes -conf PiCalculation_job.xml -output output/PiCalculation
```