imglib2 network transfer

Vladimír Ulman

CSBD and MPI-CBG

10th Oct, 2017
DAIS wp 1.3, internal meeting, Dresden

The document is valid as of commit
1564092410fe50dded97a1d8e4df4847c388371b (3-Oct-2017)

The Goal of DAIS wp 1.3

- Direct transmission of the imglib2 data between two peers
- Peers are assumed to be two independent processes
- Processes may:
  - live on the same machine
  - live on machines connected over TCP/IP network
  - any app, e.g., FIJI, KNIME, wrapped C/C++/Python library
- Inspired by the BlueTooth principles
- Uses proprietary/own protocol:
  - text header and binary voxel data
  - *similar to ICS (Image Cytometry Standard)*
  - default TCP port 54545
  - *no collision in /etc/services*
- Networking is done via ZeroMQ package.

The API of DAIS wp 1.3

How to use it from Java:

import de.mpicbg.ulman.imgtransfer.ImgPacker;
[**import** de.mpicbg.ulman.imgtransfer.ProgressCallback;]

ImgPlus<?> imgP = ...;

A_sendsTo_B:
A: ImgPacker.sendImage((ImgPlus) imgP, "tcp://"+remoteURL, timeoutTime[, logger]);
B: imgP = ImgPacker.receiveImage(portNo, timeoutTime[, logger]);

– or –

A_downloadsFrom_B:
B: ImgPacker.serveImage((ImgPlus) imgP, portNo, timeoutTime[, logger]);
A: imgP = ImgPacker.requestImage("tcp://"+remoteURL, timeoutTime[, logger]);

- TimeoutTime is given in seconds as int, max waiting time 68 yrs
- Logger is optional
- Logger must implement ProgressCallback.info(String) and ProgressCallback.setProgress(0≤float≤1)

The Principle of DAIS wp 1.3

Why two communication models:

- Network stand point:
- A and B on the same local network:
  → A_sendsTo_B, B_sendsTo_A, A_downloadsFrom_B, B_downloadsFrom_A
- A on local network (behind firewall/NAT),
- B on remote network with public IP:
  → A_sendsTo_B, A_downloadsFrom_B
- A on local network (behind firewall/NAT),
- B on remote network (behind firewall/NAT):

  → only with tunneling through public IP SSH server, or alike

- PLANNED, local network discovery mode:
  - A broadcasts "who is around"
  - B replies "B is at this IP"
  - A collects replies, presents them to user
  - A sends to...

The Principle of DAIS wp 1.3

Why two communication models:

- Flow control stand point:
- A is the main program that asks someone to process data
- A_sendsTo_B, B does some work, A_downloadsFrom_B
- A knows B because it asks B to do some work for A
- B does not need to know identification of A
  - B: imgP = ImgPacker.receiveImage(portNo, timeoutTime);
  - B: process(imgP);
  - B: ImgPacker.serveImage((ImgPlus) imgP, portNo, timeoutTime);

The Principle of DAIS wp 1.3

How to batch execute foreign code/binary (not tested yet):

- B: construct the foreign binary as:
  - receive image, process image, serve image
  - *only portNo is required*
- A: construct the Java caller as:
  - pre-process image,
  - send the image in thread Q & exec() the foreign binary in thread W,
  - receive image in thread Q & wait for thread W,
  - post-process image
  - *connecting address of B is required*
- Foreign code can be any executable
  e.g. Fiji-headless-plugin or wrapped C++ library

The API of DAIS wp 1.3

How to use it from Java (reminder):

import de.mpicbg.ulman.imgtransfer.ImgPacker;
[**import** de.mpicbg.ulman.imgtransfer.ProgressCallback;]

ImgPlus<?> imgP = ...;

A_sendsTo_B:
A: ImgPacker.sendImage((ImgPlus) imgP, "tcp://"+remoteURL, timeoutTime[, logger]);
B: imgP = ImgPacker.receiveImage(portNo, timeoutTime[, logger]);

– or –

A_downloadsFrom_B:
B: ImgPacker.serveImage((ImgPlus) imgP, portNo, timeoutTime[, logger]);
A: imgP = ImgPacker.requestImage("tcp://"+remoteURL, timeoutTime[, logger]);

- TimeoutTime is given in seconds as int, max waiting time 68 yrs
- Logger is optional
- Logger must implement ProgressCallback.info(String) and ProgressCallback.setProgress(0≤float≤1)

The Control Flow of API of DAIS wp 1.3

- All 4 functions should finish either with:
  - success after image was for sure sent or received
  - exception due to timeout (no functional connection was established)
  - exception due to other error (e.g. socket issues, protocol error)

- Currently supported:
  - only the image imgP.getName() metadata
  - ImpPacker.SUPPORTED_VOXEL_CLASSES list supported voxel imglib2 types (ByteType, UnsignedByteType, ShortType, UnsignedShortType, FloatType, DoubleType)
  - ArrayImg, PlanarImg supported; CellImg, CachedCellImg not yet

The Control Flow of API of DAIS wp 1.3

Technical details, A sends image to B:

- A calls ImgPacker.sendImage()
- A→ sends "v1 dimNumber 2 256 256 Float PlanarImg" (example): ImgPacker.packAndSend(); ImgPacker.packAndSendHeader()

- B calls ImgPacker.receiveImage()
- B← periodically (non-blocking) reads its port until timeout: ImgPacker.receiveImage()
- B← reads header, tries to create empty image: ImgPacker.receiveAndUnpack()
- B→ sends "ready": ImgPacker.receiveAndUnpack()

- A← periodically (non-blocking) reads its port until timeout: ImgPacker.packAndSendHeader()
- A← needs to read "ready", else complain: ImgPacker.packAndSendHeader()

- A→ sends "metadataQQimagenameQQsomeStringWithSpacesQQendmetadata": ImgPacker.packAndSendPlusData()
- B← reads metadata, parses them with the separator QQ (ImgPacker.mdMsgSep): ImgPacker.receiveAndUnpackPlusData()

- A→ keeps sending PlanarImg voxel data: ImgPacker.packAndSendPlanarImg()
- B← keeps reading PlanarImg voxel data: ImgPacker.receiveAndUnpackPlanarImg()

- B→ sends "done" after it finishes reading: ImgPacker.receiveAndUnpack()
- B finishes

The Control Flow of API of DAIS wp 1.3

Technical details, A sends image to B:

- A→ keeps sending PlanarImg voxel data
- B← keeps reading PlanarImg voxel data

- ArrayImg is sent in one "chunk"
- PlanarImg is sent plane-wise in multiple "chunks", one per plane
- (CellImg will be sent cell-wise)

- PlanarImg:

```
for (int slice = 0; slice < img.numSlices(); ++slice)
{
  final Object data = img.getPlane(slice).getCurrentStorageArray();
  ArraySender.sendArray(data, socket)
  // ArrayReceiver.receiveArray(data, socket);
}
```

The Control Flow of API of DAIS wp 1.3

- ArraySender.sendArray() & ArrayReceiver.receiveArray()
  - send/receive one "chunk" of data
  - figure voxel type from Object data
  - call sending/receiving routines specific for that voxel type

- Example specific for float:
- void sendFloats(final float[] data, final ZMQ.Socket socket)
  - data is length-limited array of 4B elements
  - but ZeroMQ consumes ByteBuffer: length-limited array of 1B elements
  - solution: send as 4 messages
  - one "chunk" is sent as 4 ZeroMQ messages for FloatType

- Summary:
  - ArrayImg is sent as a single plane
  - PlanarImg is sent plane-wise
  - each plane is sent as multiple ZeroMQ messages, depends on voxel type