BlockLayout-btAMReX

August 11, 2023

0.1 Summary

Analysis of block layout for deforming bubble problem in two dimensions with pseudo Uniform Grid (UG) and Adaptive Mesh Refinement (AMR) for configurations involving AMReX (in native and bittree mode) and Paramesh libraries

[1]:

```python
# Import standard libraries
import os
import sys
import itertools
from distutils.sysconfig import get_python_version
get_python_version()
```

[1]:

```
'3.11'
```

[2]:

```python
# Start by setting local path to Performance repo
PROJECT_HOME=os.path.join(os.getcwd(),'..')
```

[3]:

```python
# Import libraries to manage arrays, plotting, and Flash-X datasets
import numpy
import matplotlib.pyplot as pyplot
import boxkit
```

[4]:

```python
# Import log dictionaries
from bin.logDict import FlashLogDict, AmrexLogDict
```

[5]:

```python
# Start with defining simulation datasets we wish to study
dataFileDict = {
    "AMReX Bittree" :
    "simulation/Bittree/DeformingBubble2D/amrexBittree/jobnode.archive/
  ↪sedona-blocks64ranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0200",

    "AMReX Native" :
    "simulation/Bittree/DeformingBubble2D/amrexNative/jobnode.archive/
  ↪sedona-blocks64ranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0200",

    "Paramesh Bittree" :
```

```
    "simulation/Bittree/DeformingBubble2D/parameshBittree/jobnode.archive/
 ↪sedona-blocks64ranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0200",
}
```

[6]:

```
def getDatasetDict(dataFileDict):
    """
    Arguments
    ---------
    dataFileDict : list of simulaton files

    Returns
    -------
    datasetDict : dataset dictionary
    """
    datasetDict = {}

    for simKey, simFile in dataFileDict.items():
        dataset = boxkit.read.Dataset(PROJECT_HOME + os.sep + simFile,␣
 ↪source="flash")
        datasetDict.update({simKey : dataset})

    return datasetDict
```

[7]:

```
datasetDict = getDatasetDict(dataFileDict)
```

[8]:

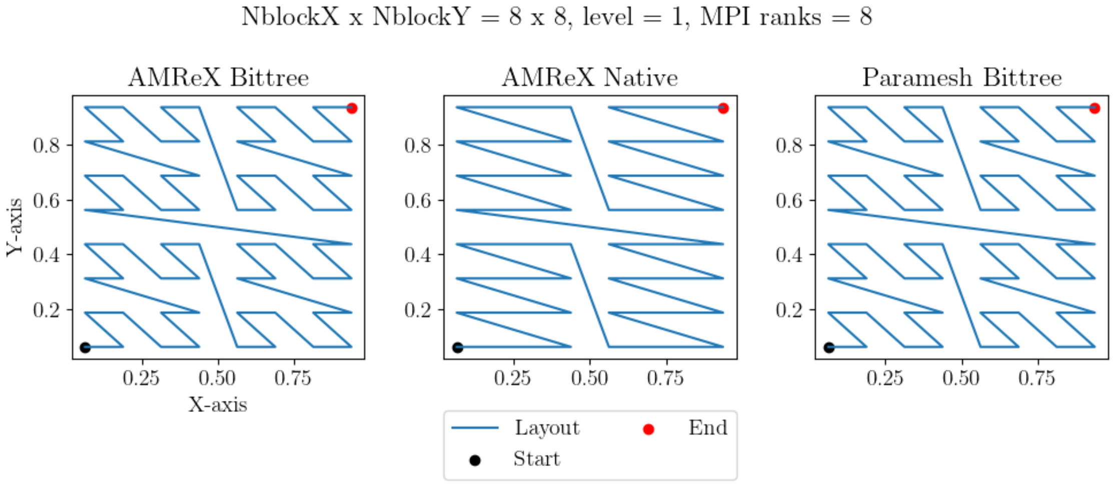
```
def plotBlockLayout(datasetDict, title, minLevel=1, maxLevel=1):
    """
    Arguments
    ---------
    datasetDict : dataset dictionary
    title : plot title
    """
    pyplot.rc("font", family="serif", size=14, weight="bold")
    pyplot.rc("axes", labelweight="bold", titleweight="bold")
    pyplot.rc("text", usetex=True)
    figure = pyplot.figure(figsize=(10, 5), dpi=100)
    pyplot.suptitle(title)
    axList= figure.subplots(1, len(datasetDict))
    figure.subplots_adjust(top=0.8)

    for title, dataset, ax in zip(list(datasetDict.keys()), list(datasetDict.
 ↪values()), axList):
        # Create an empty array to store bounds from blocklist
        blockXBnd = numpy.array([])
        blockYBnd = numpy.array([])

        # block counter
        for block in dataset.blocklist:
            if block.level >= minLevel and block.level <= maxLevel:
                blockXBnd = numpy.append(blockXBnd, block.xcenter)
                blockYBnd = numpy.append(blockYBnd, block.ycenter)

        ax.plot(blockXBnd, blockYBnd)
        ax.scatter(blockXBnd[0], blockYBnd[0], color="k")
        ax.scatter(blockXBnd[-1], blockYBnd[-1], color="r")
        ax.set_title(title)

    axList[int(len(datasetDict)/2)].legend([r"Layout", r"Start", r"End"],
                                           ncol=2, loc="lower center",␣
 ↪bbox_to_anchor=(0.5, -0.5))

    axList[0].set_xlabel("X-axis")
    axList[0].set_ylabel("Y-axis")

    pyplot.tight_layout()
    pyplot.show()
```

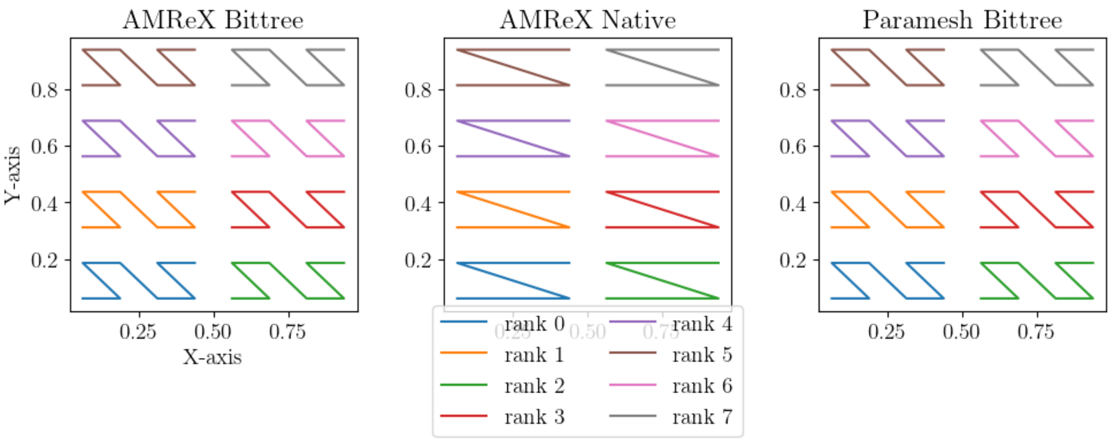
[9]:

```
def plotBlockLayoutByProc(datasetDict, minLevel=1, maxLevel=1):
    """
    Arguments
    ---------
    datasetDict : dataset dictionary
    title : plot title
    """
    pyplot.rc("font", family="serif", size=14, weight="bold")
    pyplot.rc("axes", labelweight="bold", titleweight="bold")
    pyplot.rc("text", usetex=True)
    figure = pyplot.figure(figsize=(10, 4.5), dpi=100)
    axList= figure.subplots(1, len(datasetDict))
    figure.subplots_adjust(top=0.8)

    for title, dataset, ax in zip(list(datasetDict.keys()), list(datasetDict.
 ↪values()), axList):
        # proclist
        procList = [*set([block.inputproc for block in dataset.blocklist])]

        # Create an empty list of array to store bounds from blocklist
        blockXBndList = [numpy.array([])]*len(procList)
        blockYBndList = [numpy.array([])]*len(procList)

        # block counter
        for block in dataset.blocklist:
            if block.level >= minLevel and block.level <= maxLevel:
                blockXBndList[block.inputproc] = numpy.
↪append(blockXBndList[block.inputproc], block.xcenter)
                blockYBndList[block.inputproc] = numpy.
↪append(blockYBndList[block.inputproc], block.ycenter)

        for blockXBnd, blockYBnd in zip(blockXBndList, blockYBndList):
            ax.plot(blockXBnd, blockYBnd)
            ax.set_title(title)

    axList[int(len(datasetDict)/2)].legend([f"rank {proc}" for proc in␣
↪procList],
                                           ncol=2, loc="lower center",␣
↪bbox_to_anchor=(0.5, -0.5))
    axList[0].set_xlabel("X-axis")
    axList[0].set_ylabel("Y-axis")

    pyplot.tight_layout()
    pyplot.show()
```

[10]:

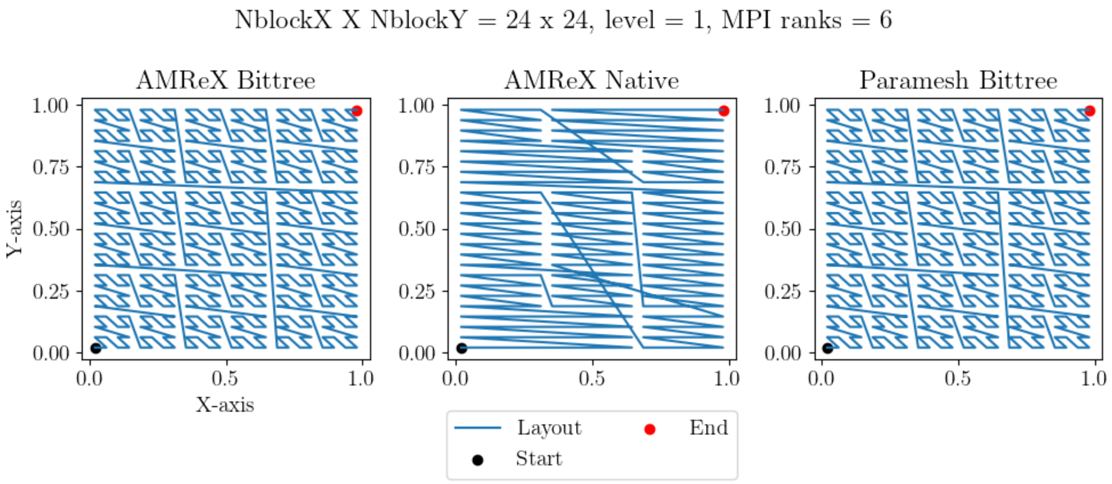
```
plotBlockLayout(datasetDict, "NblockX x NblockY = 8 x 8, level = 1, MPI ranks =␣
↪8")
plotBlockLayoutByProc(datasetDict)
```

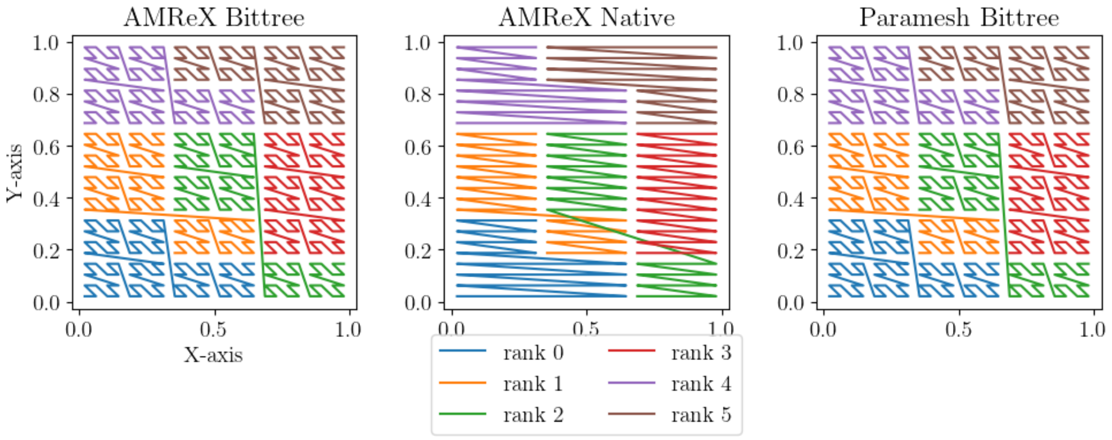




[11]:

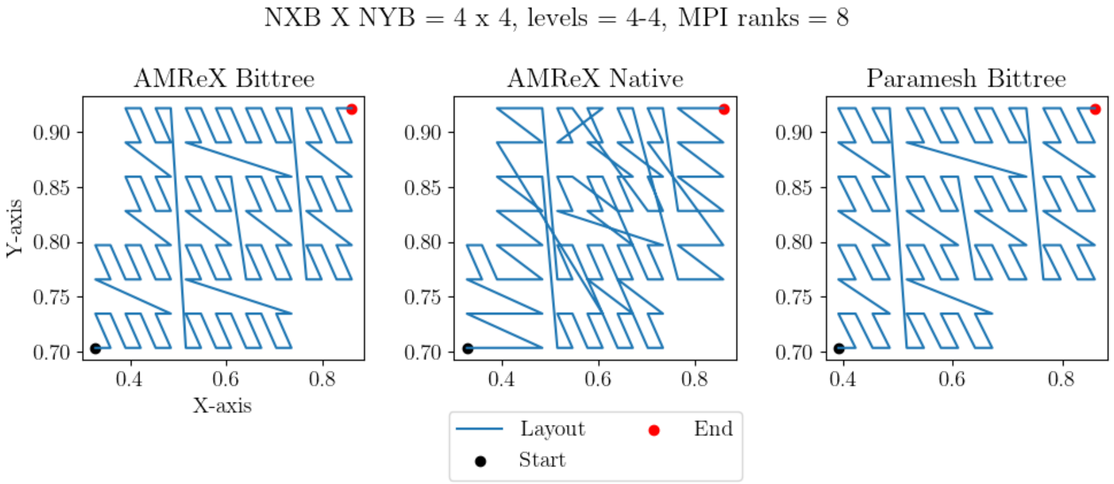
```
# Dataset locations
dataFileDict = {
    "AMReX Bittree" :
    "simulation/Bittree/DeformingBubble2D/amrexBittree/jobnode.archive/
 ↪sedona-blocks576ranks6/INS_Deforming_Bubble_hdf5_plt_cnt_0200",

    "AMReX Native" :
    "simulation/Bittree/DeformingBubble2D/amrexNative/jobnode.archive/
 ↪sedona-blocks576ranks6/INS_Deforming_Bubble_hdf5_plt_cnt_0200",

    "Paramesh Bittree" :
    "simulation/Bittree/DeformingBubble2D/parameshBittree/jobnode.archive/
 ↪sedona-blocks576ranks6/INS_Deforming_Bubble_hdf5_plt_cnt_0200",
}

datasetDict = getDatasetDict(dataFileDict)
plotBlockLayout(datasetDict, "NblockX X NblockY = 24 x 24, level = 1, MPI ranks␣
 ↪= 6")
plotBlockLayoutByProc(datasetDict)
```

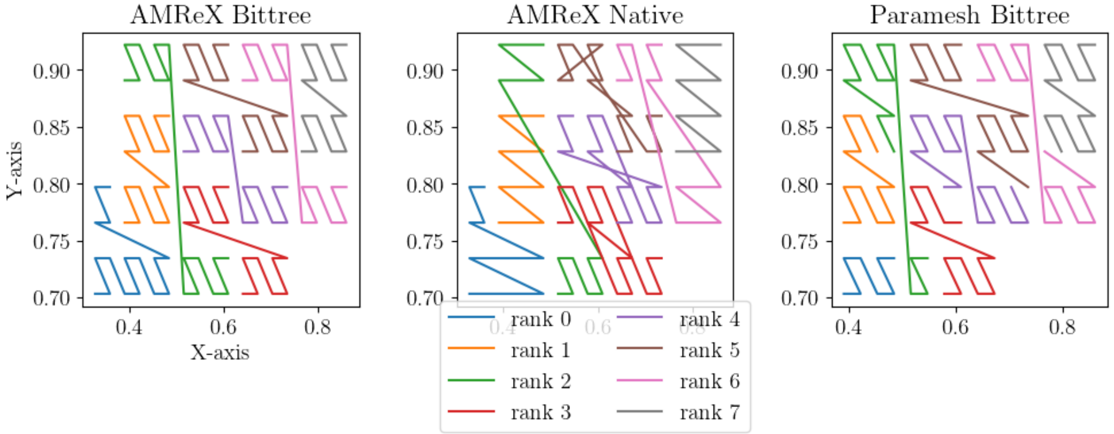




```
[12]: # Dataset locations
dataFileDict = {
    "AMReX Bittree" :
    "simulation/Bittree/DeformingBubble2D/amrexBittree/jobnode.archive/
 ↪sedona-AMRranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0025",

    "AMReX Native" :
    "simulation/Bittree/DeformingBubble2D/amrexNative/jobnode.archive/
 ↪sedona-AMRranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0025",

    "Paramesh Bittree" :
    "simulation/Bittree/DeformingBubble2D/parameshBittree/jobnode.archive/
 ↪sedona-AMRranks8/INS_Deforming_Bubble_hdf5_plt_cnt_0025",
}
```

```
minLevel, maxLevel = 4, 4
datasetDict = getDatasetDict(dataFileDict)
plotBlockLayout(datasetDict, f"NXB X NYB = 4 x 4, levels =␣
 ↪{minLevel}-{maxLevel}, MPI ranks = 8", minLevel=minLevel, maxLevel=maxLevel)
plotBlockLayoutByProc(datasetDict, minLevel=minLevel, maxLevel=maxLevel)
```





[13]:

```
from IPython.display import Image
Image(url="./assets/deforming_bubble.gif")
```

[13]:

```
<IPython.core.display.Image object>
```

[14]:

```
logFileDict = {

    "AMReX Bittree" : {
        "nodeList" : [4, 64],
        "procList" : [4*42, 64*42],
        "logFiles" : ["simulation/Bittree/DeformingBubble3D/amrexBittree/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     "simulation/Bittree/DeformingBubble3D/amrexBittree/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     ],
        "outFiles" : ["simulation/Bittree/DeformingBubble3D/amrexBittree/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/o2535277.am_bittree",
                     "simulation/Bittree/DeformingBubble3D/amrexBittree/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/o2535315.am_bittree",
                     ]
    },

    "AMReX Native" : {
        "nodeList" : [4, 64, 512],
        "procList" : [4*42, 64*42, 512*42],
        "logFiles" : ["simulation/Bittree/DeformingBubble3D/amrexNative/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     "simulation/Bittree/DeformingBubble3D/amrexNative/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     "simulation/Bittree/DeformingBubble3D/amrexNative/
↪node_0512/jobnode.archive/summit-gcc-2022-11-06/INS_Deforming_Bubble.log",
                     ],
        "outFiles" : ["simulation/Bittree/DeformingBubble3D/amrexNative/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/o2535278.am_native",
                     "simulation/Bittree/DeformingBubble3D/amrexNative/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/o2535316.am_native",
                     ]
    },

    "Paramesh Bittree" : {
        "nodeList" : [4, 64, 512],
        "procList" : [4*42, 64*42, 512*42],
        "logFiles" : ["simulation/Bittree/DeformingBubble3D/parameshBittree/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     "simulation/Bittree/DeformingBubble3D/parameshBittree/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/INS_Deforming_Bubble.log",
                     "simulation/Bittree/DeformingBubble3D/parameshBittree/
↪node_0512/jobnode.archive/summit-gcc-2022-11-06/INS_Deforming_Bubble.log",
                     ],
        "outFiles" : ["simulation/Bittree/DeformingBubble3D/parameshBittree/
↪node_0004/jobnode.archive/summit-gcc-2022-11-04/o2535280.pm_bittree",
                     "simulation/Bittree/DeformingBubble3D/parameshBittree/
↪node_0064/jobnode.archive/summit-gcc-2022-11-04/o2535317.pm_bittree",
                     ]
    },
}
```

[15]:

```
def getFlashLog(logFileDict):
    """
    Argument
    --------
    logFileDict : Dictionary of log files to parse

    """
    logDict = {}

    for logKey, logInfo in logFileDict.items():

        tempDict = {}

        if 'nodeList' in logInfo.keys():
            tempDict.update({'nodeList' : logInfo['nodeList']})
        if 'procList' in logInfo.keys():
            tempDict.update({'procList' : logInfo['procList']})

        logList = []

        for logFile in logInfo['logFiles']:
            logList.append(FlashLogDict(PROJECT_HOME + os.sep + logFile))

        tempDict.update({'logList' : logList})
        logDict.update({logKey : tempDict})

    return logDict
```

[16]:

```
flashLog=getFlashLog(logFileDict)
```

[17]:

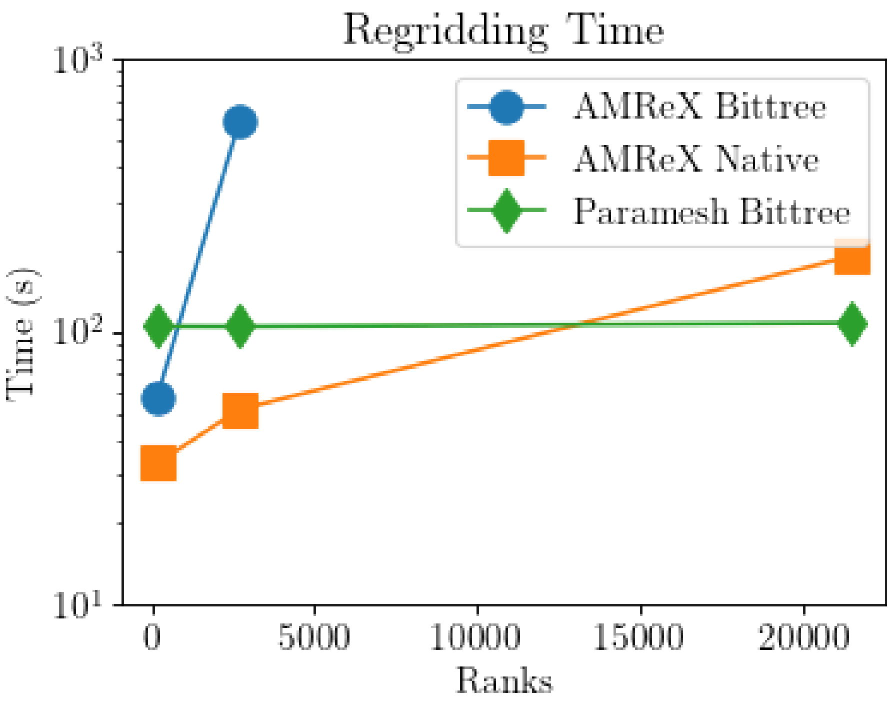
```
def getAmrexLog(logFileDict):
    """
    Argument
    --------
    logFileDict : Dictionary of log files to parse

    """
    logDict = {}

    for logKey, logInfo in logFileDict.items():

        tempDict = {}

        if 'nodeList' in logInfo.keys():
            tempDict.update({'nodeList' : logInfo['nodeList']})
        if 'procList' in logInfo.keys():
            tempDict.update({'procList' : logInfo['procList']})


        logList = []

        for logFile in logInfo['outFiles']:
            logList.append(AmrexLogDict(PROJECT_HOME + os.sep + logFile))

        tempDict.update({'logList' : logList})
        logDict.update({logKey : tempDict})

    return logDict
```

[18]:

```
amrexLog=getAmrexLog(logFileDict)
```

[19]:

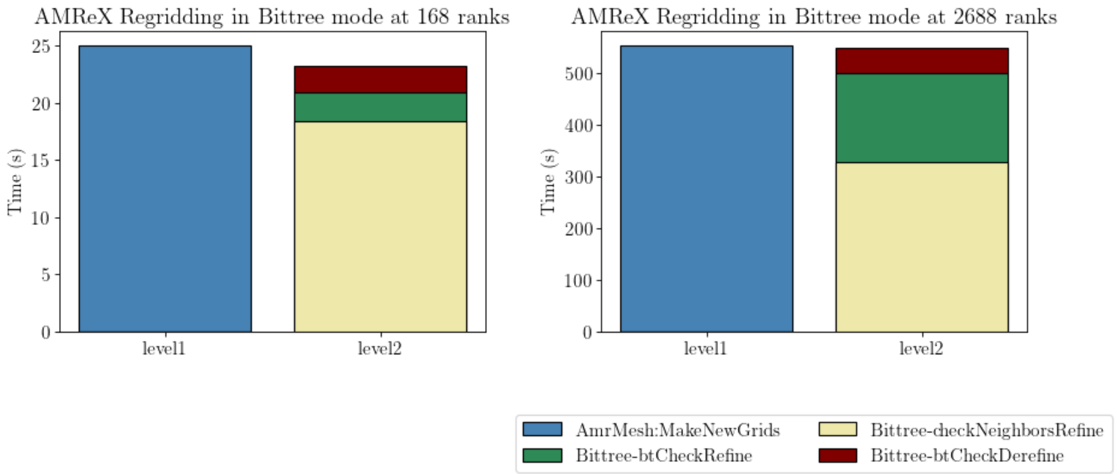
```
# Plot figure
pyplot.rc("font", family="serif", size=14, weight="bold")
pyplot.rc("axes", labelweight="bold", titleweight="bold")
pyplot.rc("text", usetex=True)
figure = pyplot.figure(figsize=(5, 4), dpi=80)

# Create subplots
ax1 = figure.subplots(1,1)

ax1.plot(flashLog["AMReX Bittree"]["procList"],
          [log['Grid_updateRefinement']['avg/proc'] for log in flashLog['AMReX␣
 ↪Bittree']['logList']],
         marker="o", markersize=12)
ax1.plot(flashLog["AMReX Native"]["procList"],
          [log['Grid_updateRefinement']['avg/proc'] for log in flashLog['AMReX␣
 ↪Native']['logList']],
         marker="s", markersize=12)
ax1.plot(flashLog["Paramesh Bittree"]["procList"],
          [log['tree']['avg/proc'] for log in flashLog['Paramesh␣
 ↪Bittree']['logList']],
         marker="d", markersize=12)

ax1.set_title("Regridding Time")
ax1.legend(list(flashLog.keys()))
ax1.set_xlabel(r"Ranks")
ax1.set_ylabel(r"Time (s)")
ax1.set_yscale("log")
ax1.set_yticks([10, 100, 1000])
pyplot.tight_layout()
```



[20]:

```
# Plot figure
pyplot.rc("font", family="serif", size=14, weight="bold")
pyplot.rc("axes", labelweight="bold", titleweight="bold")
pyplot.rc("text", usetex=True)
figure = pyplot.figure(figsize=(12, 6), dpi=80)

axList = figure.subplots(1,2)

for index, ax in enumerate(axList):

    log = amrexLog["AMReX Bittree"]["logList"][index]
    ranks = amrexLog["AMReX Bittree"]["procList"][index]

    labels = ["level1", "level2"]

    layer1 = [log["AmrMesh::MakeNewGrids"]["Incl. Avg"], 0]
    layer2 = [0, log["Bittree-btCheckRefine"]["Incl. Avg"]]
    layer3 = [0, log["Bittree-checkNeighborsRefine"]["Incl. Avg"]]
    layer4 = [0, log["Bittree-btCheckDerefine"]["Incl. Avg"]]

    bar1 = ax.bar(labels, layer1, color=['steelblue', 'black'],␣
 ↪edgecolor="black")
    bar2 = ax.bar(labels, layer2, color=['black', 'seagreen'],␣
 ↪edgecolor="black")
```

```
    bar3 = ax.bar(labels, layer3, color=['black', 'palegoldenrod'],␣
 ↪edgecolor="black")
    bar4 = ax.bar(labels, layer4, bottom=layer2, color=["black", "maroon"],␣
 ↪edgecolor="black")

    ax.set_title(f"AMReX Regridding in Bittree mode at {ranks} ranks")
    ax.set_ylabel(r"Time (s)")

ax.legend([bar1[0],bar2[1], bar3[1],bar4[1]],["AmrMesh:MakeNewGrids",␣
 ↪"Bittree-btCheckRefine",
                                              "Bittree-checkNeighborsRefine",␣
 ↪"Bittree-btCheckDerefine"],
          ncol=2,loc="lower center",bbox_to_anchor=(0.5, -0.5),)

pyplot.tight_layout()
```



0.2 Code Block

```
bool btUnit::checkNeighborsRefine(BittreeAmr* const mesh, MortonTree::Block b) {

    BL_PROFILE("Bittree-checkNeighborsRefine");

    auto tree0 = mesh->getTree();
    auto tree1 = mesh->getTree(true);
    int nIdx[3], cIdx[3];
    unsigned childCoord_u[AMREX_SPACEDIM];

    // Loop over neighbors
    for(nIdx[2]= -1*K3D; nIdx[2]<= K3D; ++nIdx[2]) {
    for(nIdx[1]= -1*K2D; nIdx[1]<= K2D; ++nIdx[1]) {
```

```
    for(nIdx[0]= -1*K1D; nIdx[0]<= K1D; ++nIdx[0]) {
        std::vector<int> nCoord = neighIntCoords(mesh, b.level, b.coord, nIdx);

        // If neighbor is outside domain or otherwise invalid, continue.
        if(AMREX_D_TERM(nCoord[0]<0, || nCoord[1]<0, || nCoord[2]<0 )) {
            continue;
        }

        // Identify neighbor from Bittree.
        unsigned neighCoord_u[AMREX_SPACEDIM];
        for(unsigned d=0; d<AMREX_SPACEDIM; ++d) {
            neighCoord_u[d] = static_cast<unsigned>(nCoord[d]);
        }
        auto n = tree0->identify(b.level, neighCoord_u);
        if(b.level==n.level && n.is_parent) {
            // Loop over children of neighbor.
            for(cIdx[2]= 0; cIdx[2]<= K3D; ++cIdx[2]) {
            for(cIdx[1]= 0; cIdx[1]<= K2D; ++cIdx[1]) {
            for(cIdx[0]= 0; cIdx[0]<= K1D; ++cIdx[0]) {

                // Only check adjacent children
                if (( ((1-nIdx[0])/2)==cIdx[0] || nIdx[0] == 0 ) &&
                    ( ((1-nIdx[1])/2)==cIdx[1] || nIdx[1] == 0 ) &&
                    ( ((1-nIdx[2])/2)==cIdx[2] || nIdx[2] == 0 )) {

                    // Identify child on updated tree
                    for(unsigned d=0; d<AMREX_SPACEDIM; ++d) {
                      childCoord_u[d] = neighCoord_u[d]*2 + static_cast<unsigned>(cIdx[d]);
                    }
                    auto c = tree1->identify(n.level+1, childCoord_u);

                    // If child WILL be parent, return true
                    if( c.level==(b.level+1) && c.is_parent) {
                        return true;
                    }
                }
            }}}
        }
    }}}

    // Return false otherwise
    return false;
}
```