Eikonal Parallel Solver on Android

- Eikonal equation
- C++ Implementation on serial and multi threaded CPUs
- Code optimization aspects and results
- G++, clang++, and Intel compiler
- C++ parallel programming on Android using NDK
- Android Programming
Eikonal equation

- Mathematical description.
- Definition of the local solver.
- Algorithm description
Mathematical description

- Special case of Hamilton-Jacobi PDEs.
- Compute time of excitation $\varphi(x) : \mathbb{R}^3 \rightarrow \mathbb{R}$ for every point $x \in \Omega$ of the mesh discretized by tetrahedral elements

$$H(x, \nabla \varphi) = \sqrt{(\nabla \varphi)^T M(\nabla \varphi)} = 1 \ \forall x \in \Omega \subset \mathbb{R}^3$$

with $M(x)$ denoting the velocity information of the domain (a.k.a. velocity map).

Velocity map example  
Solution example
Definition of the local solver.

$\Phi_{i,j} = \Phi_j - \Phi_i$  Travel time  
$e_{i,j} = x_j - x_i$  Edge vector  
$X_i = (x, y, z)^T$ where $x, y, z \in \mathbb{R}$  

Upwind scheme to compute $\Phi_4$.

Speed function is constant so the travel time to $x_4$ is determined by the time associated with the segment along the wave front normal direction that minimizes $\Phi_4$.

Denote the travel time from $x_5$ to $x_4$ given by Fermat’s principle:

$\Phi_{5,4} = \Phi_4 - \Phi_5 = \sqrt{e_{5,4}^T M e_{5,4}}$
Definition of the local solver.

- **The goal** is to find the location of $x_5$ that minimizes $\Phi_4$

$$x_5 = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$

$$\Phi_5 = \lambda_1 \Phi_1 + \lambda_2 \Phi_2 + \lambda_3 \Phi_3$$

$$1 = \lambda_1 + \lambda_2 + \lambda_3$$

$$\Phi_4 = \lambda_1 \Phi_1 + \lambda_2 \Phi_2 + (1 - \lambda_1 - \lambda_2) \Phi_3 + \sqrt{e_{5,4}^T M e_{5,4}}$$

Take partial derivatives w.r.t $\lambda_1$, $\lambda_2$ and equate them to 0:

$$\begin{cases}
\Phi_{1,3} \sqrt{\lambda^T M' \lambda} = \lambda^T \alpha \\
\Phi_{2,3} \lambda^T \alpha = \Phi_{1,3} \lambda^T \beta
\end{cases}$$

$$M' = \begin{bmatrix}
e_{1,3}^T M e_{13}, e_{2,3}^T M e_{13}, e_{3,4}^T M e_{13} \\
e_{1,3}^T M e_{23}, e_{2,3}^T M e_{23}, e_{3,4}^T M e_{23} \\
e_{1,3}^T M e_{34}, e_{2,3}^T M e_{34}, e_{3,4}^T M e_{34}
\end{bmatrix}^T$$

$$\lambda = \begin{bmatrix}\lambda_1 & \lambda_2 & 1\end{bmatrix}^T$$

Apply the 2D local solver to the faces $\Delta_{124}$, $\Delta_{134}$, and $\Delta_{234}$ in the following cases:

- No root.
- Bad tetrahedron
- $\lambda_i \not\in [0, 1]$
Algorithm description – FIM method

- Iterate over elements in "Active-List"
- Add not converged neighboring elements
- Remove converged elements
Algorithm description – FIM method

- **Active List Update Scheme**
- Combined with the local solver for tetrahedral meshes
- Data structure: “Active list” set
- The algorithm is iterative
- During each iteration active vertices can be updated in parallel
- Convergence of the algorithm is proven on [1]

```plaintext
comment: 1. Initialization (X : set of grid points, L : active list)
for each x ∈ X
  do { if x is source then U_x ← 0
       else U_x ← ∞
       }
for each x ∈ X
  do { if any neighbor of x is source then add x to L
       }

comment: 2. Update points in L
while L is not empty
  for each x ∈ L
    do { p ← U_x
          q ← g(U_x)
          U_x ← q
          if |p - q| < ε
          do { for each 1-neighbor x_nb of x
                do { if x_nb is not in L
                      then { p ← U_x_nb
                             q ← g(U_x_nb)
                             if p > q
                             then { U_x_nb ← q
                                     add x_nb to L
                                 }
                           }
                     remove x from L
                 }
             }
         }
     }
```

Algorithm (1)
• Iterative C++ Implementation

• Update method
  - **Active list** initialized
  - **Set** used as the data structure for the active list.
  - List converted to vector for each iteration to avoid iterator invalidation during updates (insert & delete)
  - Iteratively **process** each node in the list
  - Return the solution vector **U**.
  - Generate .vtk file

```cpp
vector<float> updateScheme::update()
{
  initialize_list();

  while(!activeList.empty()){
    cout << "ActiveList size: " << activeList.size() << endl;
    const vector<int> vec(activeList.cbegin(), activeList.cend());
    const size_t nvec = vec.size();

    for(size_t node = 0; node < nvec; ++node)
      process(vec[node]);
  }

  return getU();
}
```
Iterative C++ Implementation

- Process method:
  - Calculate new solution and update the old value
  - Check convergence
  - Add neighboring elements
  - Avoid iterator invalidation:
    - Iteration done on the vector
    - Updates done on the set
  - Remove converged element
OpenMP parallelization

- Because the serial algorithm does not depend significantly on the ordering of updates, the extension to multiple processors is immediate.

- We simply divide the active list arbitrarily into N sublists, assign the sublists to the N threads, and let each thread use an asynchronous update for the vertices within the sublist.

- These updates are done by applying the updating step in Algorithm (1) to each sub-active list.

```cpp
class updateScheme {  
public:  
    void update() {  
        initialize_list();  
        while (!activeList.empty()) {  
            int maxNrThread = 8;  // define the number of threads and also the number of the sublists  
            int nr = activeList.size() / maxNrThread;  // the number each sublist should contain.  
            vector<set<int>> vectorSet; // every sublist is going to be contained in a vector  
            for (int i = 0; i < maxNrThread; i++) {  // divide the big set equally for n-1 subsets.  
                set<int> tmp;  
                for (int j = 0; j < maxNrThread; j++) {  // each thread is assigned a subset to process  
                    tmp.insert((vectorSet[j]).cbegin(), (vectorSet[j]).cend());  
                }  
                // clear the big list before merging all the updated sublists into one after the iteration.  
                activeList.clear();  
                for (int i = 0; i < maxNrThread; i++) {  
                    activeList.insert((vectorSet[i]).cbegin(), (vectorSet[i]).cend());  
                }  
            }  
            return getU();  
        }  
    }  
};
```
```cpp
void updateScheme::process(const int & list_iter, set<int>& activeList)
{
    float p, d, q;

    p = getU()[list_iter];
    q = minSolutionOneRingTetra(list_iter, p); // calculate the solution of the point x of the list by the local solver on every one nbh tetra and pick the min

    if(p>q){
        #pragma omp critical // this block should be critical because two threads may read the same value from the solution vector before one of them modifies it
        // meaning that the condition p>q is as a result not valid since the comparison is done in non consistent data and therefore may happen that 
        // one thread updates teh solution even if it is not less than the current value

        {
            if(q<getU()[list_iter])

                #pragma omp atomic write
                getU()[list_iter] = q;
            
        }

    }

    if(abs(p-q) < epsilon*1*(abs(p)+abs(q))/2){ // if converges then

        for(int i=getN2nRow()[list_iter];i<getN2nRow()[list_iter+1];i++) { // for each nbh if it is downwind than solve and remove the element.
            const int idx = getN2nCol()[i];

            if(idx != list_iter){

                if(activeList.find(getN2nCol()[i]) == activeList.cend()){


```
OpenMP parallelization

```c
/*
#pragma omp critical
{
    p = getU[idx];

    d = getU[list_iter];
    //

    if (p > d) { // add only down wind neighbors
        q = minSolutionOneRingTetra(idx, p); // calculate the solution of the "nbh of x" of the list by the local solver on every one nbh tetra and pick the min
        if (p > q) { // if the new value computed is less than the current one for the nbh than the nbh should be added to the list.
            #pragma omp critical
            {
                if (q < getU[idx])
                    #pragma omp atomic write
                    getU[idx] = q;
            }
            activeList.insert(idx); // insert the new element before the *list_iter elem
        }
    }
}
activeList.erase(list_iter); // remove x from L
```
Code optimizations & Results

- Adjacency matrix method.
- `#pragma omp critical` optimization.
- `const` keyword.
Connection matrix (e2n- Sparse)

- $n2e = e2n^\top$ - node to element matrix

- $n2n=n2e*e2n$ – node to node matrix

- Matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position according to whether and are adjacent or not.

- For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal.

- For an undirected graph, the adjacency matrix is symmetric.
Connection matrix (e2n- Sparse)

- n2e = e2n\textsuperscript{T} - node to element matrix

- n2n = n2e * e2n – node to node matrix

- Matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position according to whether and are adjacent or not.

- For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal.

- For an undirected graph, the adjacency matrix is symmetric.
HyperCube1000.0 example

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tt 0 1 2 4 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tt 2 1 3 7 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tt 1 4 5 7 1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tt 1 2 4 7 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tt 4 2 6 7 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ n2e = e2n' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \]

n2eCol = [0 0 1 2 3 0 1 3 4 1 0 2 3 4 2 4 1 2 3 4]
n2eRow = [1 4 4 1 4 1 1 4]
N2eDsp = [0 1 5 9 10 14 15 16 20]

```cpp
void updateScheme::getNeighborsOf(const int& x, vector<int>& nbhs){
    for(int i=getN2nRow()[x]; i<getN2nRow()[x+1]; i++){
        if(getN2nCol()[i] != x) nbhs.push_back(getN2nCol()[i]);
    }
}
```
Results

- slab file:
  240000 tetrahedrons
  52521 vertices

- Iterative update time (no optimization)
  250 sec

- Iterative update time optimized – 15 sec
void updateScheme::process(const int & list_iter, set<int>& activeList)
{
    float p, d, q;
    #pragma omp critical
    {
        p = getU()[list_iter];
        q = minSolutionOneRingTetra(list_iter, p); // calculate the solution
        if(p>q){
            getU()[list_iter] = q;
        }
    }
    if(abs(p-q) < epsilon*(1+(abs(p)+abs(q))/2)) { // if converges then
        getU()[list_iter] = q;
    }

    if(idx != list_iter){
        if(activeList.find(getN2nCol()[i]) == activeList.cend()){
            #pragma omp critical
            {
                p = getU()[idx];
                d = getU()[list_iter];
                if(p > d) { // add only down wind neighbors
                    q = minSolutionOneRingTetra(idx, p); // calculate the solution
                    if(p>q){ // if the new value computed is less than
                        if(q<getU()[idx])
                            getU()[idx] = q;
                        activeList.insert(idx); // insert the new element
                    }
                }
            }
        }
    }
    activeList.erase(list_iter); // remove x from L
- No need to use `pragma atomic` or `critical` for read access anymore!
- Increased performance!
Results

- TbunnyC_i file:
  3073529 tetrahedrons
  547680 vertices

- Iterative update time
  60.8668 sec

- Parallel update time (no optimization)
  103.433 sec

- Parallel update time optimized
  16.8859 sec
Intel Hyper-Threading Technology

- In practice we take full advantage of hyper-Threading tech.
- We choose N to be twice the number of CPU cores.
- Number of threads is equals to the number of sublists.
- To each thread a sublist is assigned for processing.

Results for TbunnyC_i:
- Number of threads equal to the number of sublists and processor cores (4). Update time – 25.9162 sec
- Number of threads equal to the number of sublists twice the number of processor cores (4). Update time – 18.1209 sec
- G++, clang++ and Intel compiler

- G++
  - g++ -I"/home/daniel/git/eeikonalc/src/headers" -std=c++11 -O3 -Wall -c -g -pedantic -Wextra -Wshadow -Wconversion -Weffc+ + -fopenmp EEikonalC++.cpp

- Clang++-3.6
Results

- **G++**

  1. [-Wshadow] warings

     ```
     void setPts(vector<vector<float>>& pts) {
         // headers/updateScheme.h:66:43: warning: declaration of 'pts' shadows a member of 'this' [-Wshadow]
         vector<vector<float>>& pts; // ^
     }
     ```

  2. [-Wconversion] Warnings

     ```
     const float epsilon = 1e-5;
     // headers/updateScheme.h:51:24: warning: implicit conversion loses floating-point precision: 'double' to 'const float' [-Wconversion]
     ```

- **Clang++-3.6**

  1. [-Wshadow] warings

     ```
     void setPts(vector<vector<float>>& pts) {
         // headers/updateScheme.h:66:38: warning: declaration shadows a field of 'updateScheme' [-Wshadow]
         vector<vector<float>>& pts; // ^
     }
     ```

  2. [-Wconversion] Warnings

     ```
     const float epsilon = 1e-5;
     // headers/updateScheme.h:51:24: warning: implicit conversion loses floating-point precision: 'double' to 'const float' [-Wconversion]
     ```
Reasons why to use Clang++

1. More Helpful Diagnostics Messages

2. Reporting more warnings for the same flag

3. Thread Safety Analysis (clang++-3.7)
   1. Supports pthreads
   2. Mutex lock and unlock
Data race analysis

- Intel Inspector

```cpp
if(p>q){ // if the new value computed is less than
    #pragma omp critical
    {
        if(q<getU()[idx])
            //pragma omp atomic write
            getU()[idx]=q;
    }
    activeList.insert(idx); // insert the new eleme
}
```

- Clang++-3.6
  Compiled with -Wthread-safety flag!

- No result!
Android NDK

NDK is a toolset that allows you to implement parts of your app using *native-code (jni)* languages such as C and C++

Use the NDK if your app is truly processor bound—never because you simply prefer to program in C/C++

<table>
<thead>
<tr>
<th>Native Code CPU Architecture Used</th>
<th>Compatible Android Platform(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM, ARM-NEON</td>
<td>Android 1.5 (API Level 3) and higher</td>
</tr>
<tr>
<td>x86</td>
<td>Android 2.3 (API Level 9) and higher</td>
</tr>
<tr>
<td>MIPS</td>
<td>Android 2.3 (API Level 9) and higher</td>
</tr>
</tbody>
</table>

To ensure compatibility, an application using NDK must declare a `<uses-sdk>` element in its manifest.xml file `android:minSdkVersion` attribute value of "3" or higher. For example:

```xml
<manifest>
  <uses-sdk android:minSdkVersion="3" />
  ...
</manifest>
```
Getting Started with the NDK

1. Download and Install NDK package.
   - `ndk$ chmod a+x android-ndk-r10c-darwin-x86_64.bin`
   - `ndk$ ./android-ndk-r10c-darwin-x86_64.bin`
   - When uncompressed the NDK files are contained in a directory called `android-ndk-<version>` ready to use.

- Place your native sources under `<project>/jni/...`
- Create `<project>/jni/Android.mk` to describe your native sources to the NDK build system.
- Build your native code by running the `ndk-build` script from your project's directory. It is located in the top-level NDK directory.
  - `Cd <project>`
  - Execute `ndk-build` script
  - The build tools copy the stripped, shared libraries needed by your application to the proper location in the application's project directory.

- Finally, compile your application using the SDK tools in the usual way. The SDK build tools will package the shared libraries in the application's deployable .apk file.
Local C++ vs NDK C++ project
Jni Driver function

- The 'extern "c"' command tells the compiler to give the method "C linkage"
- Native C++ function
- Specific function name pattern "Java_package_class_method"
- Takes a Java object string and the JNI environment parameters
- Converts the Java String to C++ String
- Then it calls an ordinary C++ function

```c
extern "C" { /*C++ function names get mangled by the
    compiler (to support overloading and other things) -- unless
    you specify extern "C". If you forget the extern
    declaration, JNI will be unable to find the function
    implementations to match your Java "native" declarations*
    Important: Only for C++ code no need for C*/

    //"JNIEXPORT void JNICALL" defines the return type of the method as a Java-compatible string.
    JNIEXPORT void JNICALL
    Java_com_opengl_hyperCube_mesh_JniRunnable_solveEikonalOn(JNIEnv * env, jobject obj, jstring logThis) {
        jboolean isCopy;
        const char * szLogThis = (env)->GetStringUTFChars(logThis, &isCopy);
        string elemFile = szLogThis;
        JNIEngine j;
        j.setFromJNI2Native(elemFile);
        (env)->ReleaseStringUTFChars(logThis, szLogThis);
    }
```
Android.mk and Application.mk

Local module to be loaded in the Java part.

Recommended to use android logging feature -llog (/system/lib/liblog.so).

As of revision 9, the Android NDK comes with support for OpenMP.

Important you need LDFLAGS also for the linker.

The default compiler is GCC 4.6. The NDK_TOOLCHAIN_VERSION key specifies the 4.9 value to define the toolchain to use GCC 4.9 as the compiler.

The default configuration of the NDK build system generates the machine code for the armeabi; that is, the ARMv5TE-based CPU. The APP_ABI value allows you to select different ABIs.

The following line enables the supported C++11 features for all your modules

GNU STL implementation for STL features.

The following line specifies the path to GNU STL implementation headers included in the NDK folders. Because of the specified toolchain use GCC 4.9, make sure the headers path corresponds to version 4.9.
Android Project Eclipse

- Good reference how to create a Android project in Eclipse:

- JniRunnable Thread class for handling the dialog job.

- Mesh: GlSurfaceView class for drawing tetrahedras using OpenGl.

- Run: the activity class.

- Tetrahedron class for creating tetrahedra objects to be drawn.

- FileUtilities class handling the file load from /sdcard.
<xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.opengl.hyperCube.mesh"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="18"
        android:targetSdkVersion="22"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_SETTINGS" />

    <application android:icon="@drawable/icon" android:label="@string/app_name"
        android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
        android:allowBackup="false"
        android:allowClearUserData="true"
        android:hardwareAccelerated="true"
        android:largeHeap="true">
        <activity android:hardwareAccelerated="true" android:name="com.opengl.hyperCube.mesh.Run"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

<xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Eikonal Solver</string>
    <string name="author">Daniel Ganellari</string>
    <string name="dialog_title">Open Data</string>
    <string-array name="mesh">
        <item>hyperCube100.0</item>
        <item>slab</item>
        <item>TBunnyC2.i</item>
        <item>TBunnyC.i</item>
    </string-array>
    <color name="blueAndroid">#0065B3</color>
</resources>

<xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/blueAndroid"
    android:id="@+id/linearlayout1">
    <com.opengl.hyperCube.mesh.mesh
        android:id="@+id/glSurfaceViewID"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="0.23"/>
</LinearLayout>
Create the Activity Class

- The class should extend `Activity` class
- Declare the openGL view
- Declare the JniRunnable and start the new Thread.
  - Not mandatory! Just for needs of this app.
- `SetContentView` sets the layout declared in the `eikonal.xml` file and sets it as the view.
- Start the thread!
- An activity class should contain also `onPause()` and `onResume()` but they are automatically created after extension as override methods.

```java
public class Run extends Activity {
    /** The OpenGL View */
    private mesh mes;

    public static JniRunnable meshRun = new JniRunnable(false);

    /**
     * Initiate the OpenGL View and set our own
     * Render (see Lesson05.java)
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.eikonal);

        mes = (mesh) findViewById(R.id.glSurfaceViewID);

        if (meshRun.exists == false) { // run the thread solver only
            meshRun.setContext(this);
            meshRun.start();
            meshRun.exists = true;
        }
    }
}
```
- For a class to be a runnable instance it should inherit from Thread class or implement Runnable interface.

- Native function declaration same name as in the NDK C++ impl.

- Call the native NDK function `solveEikonalOn` inside the `solve` function.

- Load the “Eikonal” Library as declared into the Android.mk NDK file.

- The Thread runs during all the lifecycle and executes `solve` function whenever a file is chosen and `solve` variable is set to `true`.

- After the solving has finished the rendering of the mesh is done and the progress Dialog is dismissed.

- **Important:** The native function declaration, library load and call should be done by the same Thread.
Create an item list menu by using an alert dialog which reads the R.array.mesh created in the string.xml file.

OnClick function the dialog is closed, a file name and index position chosen.

The path is then built.

A progress dialog is built and shown which is then closed when the Eikonal equation is solved for the chosen file.

Properties for the JniRunnable static object are set which are read from the thread and upon solve=true it starts solving the file chosen.

Then a requestRender() is invoked to start drawing the tetrahedra.
Results on Android

- **slab file:**
  
  240000 tetrahedrons
  52521 vertices

- Parallel update time optimized
  8.599 sec

- **TBunnyC2_i file:**
  
  266846 tetrahedrons
  59292 vertices

- Parallel update time optimized
  29.657 sec

- **TBunnyC_i file:**
  
  3073529 tetrahedrons
  547680 vertices

- Parallel update time optimized
  257.141 sec

- *Running the tests with 4 sublists 4 threads.*
Conclusions

- Local C++ code giving very good results.

- Solving on parallel gives better results on solution values than iteratively since more solutions are found on Tetrahedrons than on Triangle surface sides or edges of the Tetrahedrons.

- Solving on Android gives satisfactory results.

- There is still room for performance improvement.

- Thinking on memory layout and vectorization.
References

