ttgLib — A Middleware For Dynamic Software Adaptation To Heterogeneous Architectures

Maxim Krivov[1,2], Sergey Grizan[2], Mikhail Pritula[1,2], Pavel Ivanov[1,2]
{ M_Krivov, S_Grizan, M_Pritula, P_Ivanov }@ttgLabs.com

[1] M.V. Lomonosov Moscow State University (Russia), [2] ttgLabs, LLC

Moscow State University

ttgLabs, LLC

Overview

What is ttgLib?

ttgLib is a C++ middleware library that provides a set of optimization primitives to solve the problem of software performance optimization on heterogeneous systems. ttgLib enables to tailor an application to the hardware platform it is running on as well as to the processed data during the runtime.

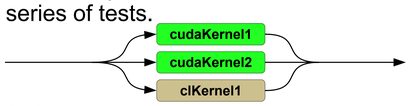
Problems addressed

- Magic constant determination. Most of the so called magic constants depend on data size. Therefore, at the compilation time, their optimal values remain unknown.

```
const int threshold = 1024;        #define GRAIN 32
#define BLOCK_SIZE 256             long tile_length = 384;
```

- Wise load balancing. A simultaneous usage of CPU and GPUs can either accelerate or slow down the application. That's why an optimal strategy of load distribution cannot be implemented statically in the application code.

- Best kernel selection. Hardware-specific optimizations spawn numerous implementations of the same algorithm. Usually, the best version can be chosen only after a series of tests.



The solution

- Dynamic parameters. Any magic constant that affects application performance can be replaced by dynamic parameter the best value of which is defined automatically.

```
int grain_size = 32;                    Parameter<int> grain_size = 32;
double threshold = 0.1;                 Parameter<double> threshold = 0.1;

if (n < grain_size)                     if (n < grain_size)
//...                                   //...
```

- Hybrid programming primitives. Hybrid implementations of popular parallel primitives like HybridFor or HybridTask efficiently distribute the load between all available computing devices.

```
if (data.size() > 512)                  HybridTask htask;
   ProcessOnGPU(data);                  htask.CUDA() += ProcessOnGPU;
else                                    htask.Serial() += ProcessOnCPU;
   ProcessOnGPU(data);
                                        htask.Execute(data);
```

- Multiple optimization strategies. Taking account of application specificity the user can choose the best optimization strategy for a particular computing pattern.

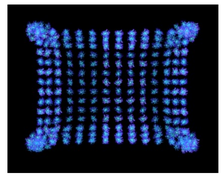
Testing

Heat-Mass-Transfer

Solution for Laplace equation with Neumann boundary conditions on a 2D grid.

NBody

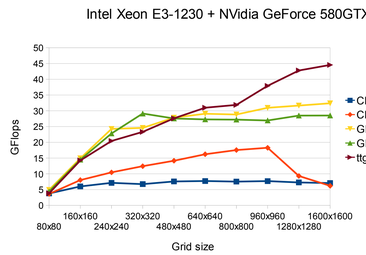
Simulation of the system of *N* bodies interacting with each other

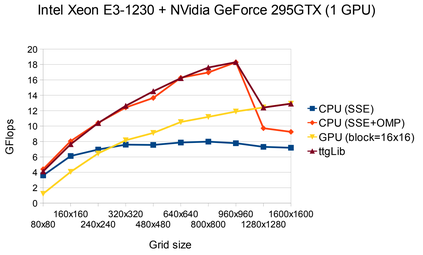


Video-Processing

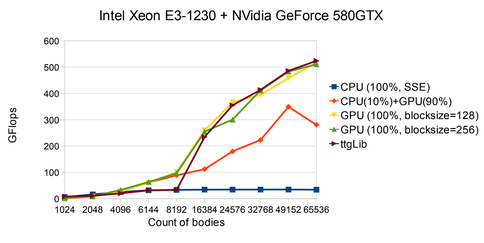
Each frame is converted to HSV, edited and then converted back to RGB

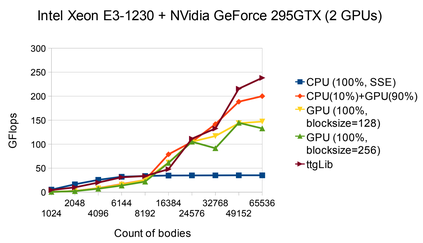
Heat-Mass-Transfer sample



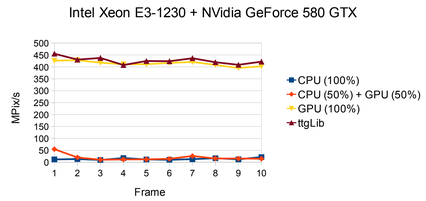


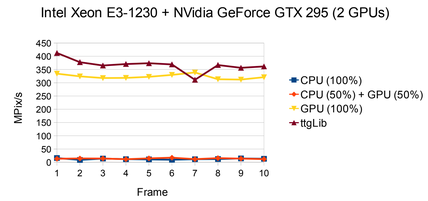
N-Body sample





Video-Processing sample





The source code of these samples is available in ttgLib SDK

How To

- **Step 0.** Install ttgLib SDK and add it to the INCLUDE and LIBRARY paths.
- **Step 1.** Replace performance-critical constants by dynamic parameters

```
dim3 threads(128);                      ttgLib::CudaGrid1D grid(count);
dim3 blocks(count / 128);
kernel<<<blocks,                        kernel<<<grid.GetBlocks(),
       threads>>>();                            grid.GetThreads()>>>();
```

- **Step 2.** Use hybrid primitives

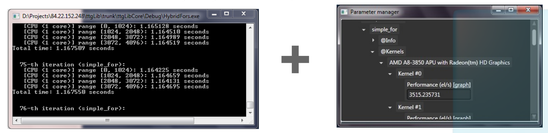
```
ttgLib::HybridFor1D hfor;               ttgLib::HybridTask htask;
hfor.CUDA() += cudaKernel1;             htask.CUDA() += cudaKernel;
hfor.CUDA() += cudaKernel2;             htask.OpenCL() += clKernel;
hfor.OpenCL() += clKernel;              htask.Serial() += cppKernel;

hfor.Process(data, 1024);               htask.Execute();
```

- **Step 3.** Define iterations and enable optimization

```
                                        ttgLib::OptimizationSession os;
while (residual > 1e-10)                while (residual > 1e-10)
{                                       {
                                           os.StartIteration();
   //...                                  //...
                                           os.FinishIteration();
}                                       }
```

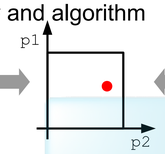
- **Step 4** (Optional). Use external ttgLib utilities to change the optimization strategy, the device priorities or even the value of any dynamic parameter. Just IN RUNTIME!



Approaches

- All primitives and parameters are mapped into hypercube $[0.0, 1.0]^n$
- Thus, the global optimization problem is reduced to the minimization of time function in a hypercube
- At each iteration, the execution time as a function of particular parameters' values is measured
- Optimization is done based on user-selected optimization strategy and algorithm

```
Parameter<double> p1;
Parameter<int> p2;
```



$$J(p1, p2) \to min$$
$$p1 \in [0.0, 1.0]$$
$$p2 \in [0.0, 1.0]$$