Master Educational Program
“Information technology in applications”

Mathematical computations with GPUs

Adaptation for GPU

Alexey A. Romanenko
arom@ccfit.nsu.ru
Novosibirsk State University

Features of GPU

* GPU — processor with SIMD architecture
* Huge number of threads (~1000)
* «simple» operations
* Recursion* is absent

Well suited tasks

* Tasks with data-parallel computations
* Examples of adopted tasks: www.nvidia.com/object/cuda_home.html

Bad-suited tasks

* Task which is not well-suited to SIMD parallelization
  * Task-parallel computations
* Task which is naturally sequential:
  * Some data compression algorithms
  * IIR-filters
  * Algorithms with recursion

Approach

* Analyze program to find pieces of code which could be executed in SIMD mode.
* Arrange the code as a function.
* Select way to distribute data.
* Implement kernel for GPU.
* Replace function with sequence of operations: H2D, kernel launch, D2H.
* Remove unnecessary data transfer operation.
* Optimization.

Bio-informatics

* Motives search
* Molecular dynamic

Motives search

* **Task**: Array of RNA sequences is given. For each motives count all RNA sequence matching the motive.
* Length of sequence – 50..64 chars. ABC is “ATGC”.
* Motive length – 8 chars. ABC from 15 chars.
* Data distribution approaches:
  * One thread process one motive and one RNA sequence
  * One thread process one motive and all RNA sequences
  * One thread process a set of motives and all RNA sequences

Motives search: Results

* Origin
  * 1 core (Pentium 4)– 14-15 days
  * 32 node (dual CPU) cluster – 8-10 hours
* Algorithm was changed
  * Pentium 4 – 7 day
  * GPU (Tesla C1060) – 6 hours
  * FPGA – 20 minutes

Molecular dynamics (SIESTA)

* **Task:** Move SIESTA package to GPU having minimal code modifications

* Result:
  * BLAS library -> cuBLAS
  * LAPACK library -> Magma
  * FFT -> cuFFT
  * No more than two dozens line of code were added
  * Up to 3x speed-up single GPU vs. 8 CPU cores

Geo-science

* Seismic data processing
* Tsunami wave modeling

Seismic data processing

* **Task:** Implement algorithm of 3D-wave package decomposition on GPU.

* Features:
  * Terabytes of data on non-regular grid
  * FFT -> interpolation -> scaling -> IFFT

Problems

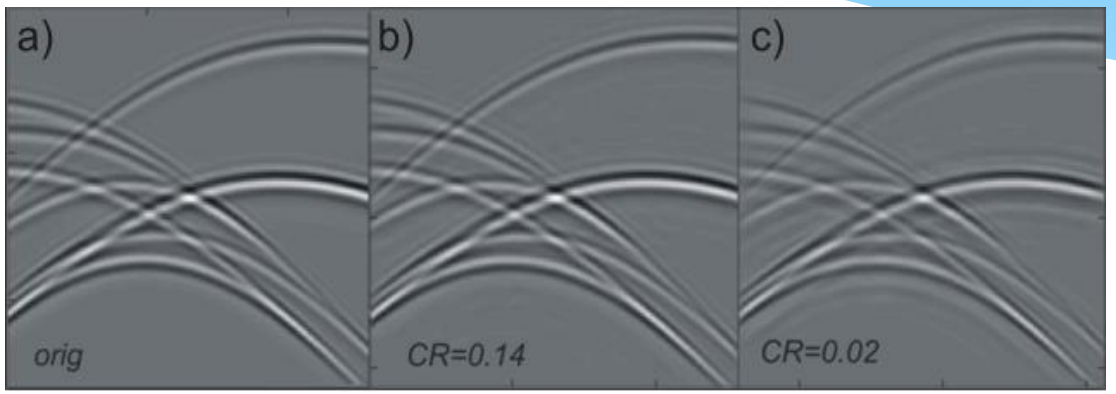
Seismic data processing

* Problem size – 256^3

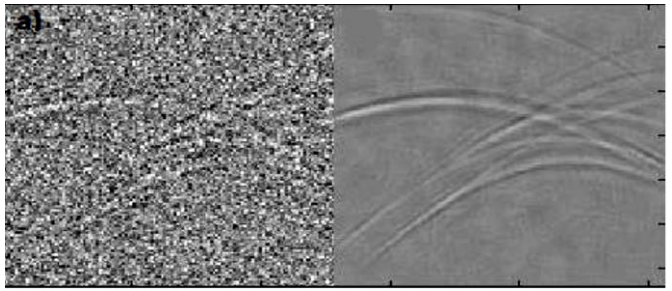
Platform	Direct transf. (sec)	Inverse transf. (sec)	Speed up
Tesla C2010	65,51	89,48	45x / 35x
GeForce Quadro 4000	88,05	119,06	33x / 26x
GeForce 9800 GX2	368,98	464,87	8x / 7x
4 x Dual-Core AMD Opteron(tm) 2218 HE	1123,79	1150,13	2,6x / 2,7x
Intel core i7	2947,50	3137,04	1x / 1x

* Now we can work with bigger data sets. We have close to linear scaling with GPU numbers.

Seismic data processing



* Compression

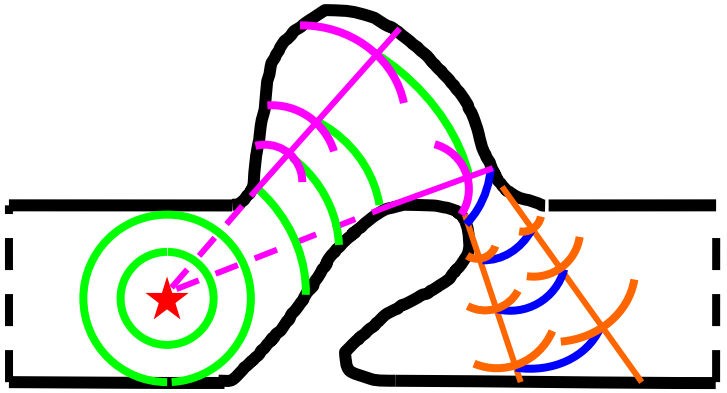


* Noise suppression

* Filling gaps

Modeling of Seismic Waves

* Feasible source wave field in terms of cascade diffraction



$$F = G + D^{(1)} + \ \ldots$$

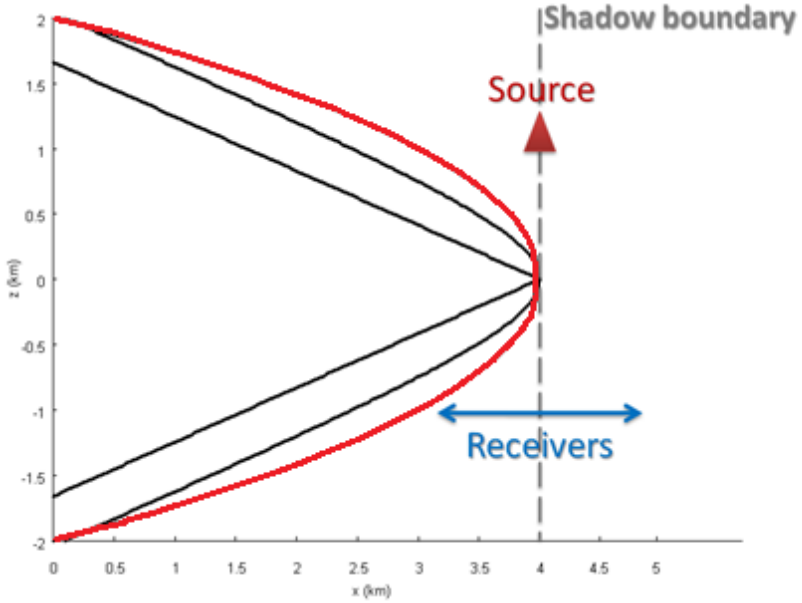
$$D^{(1)} = PAG_s$$

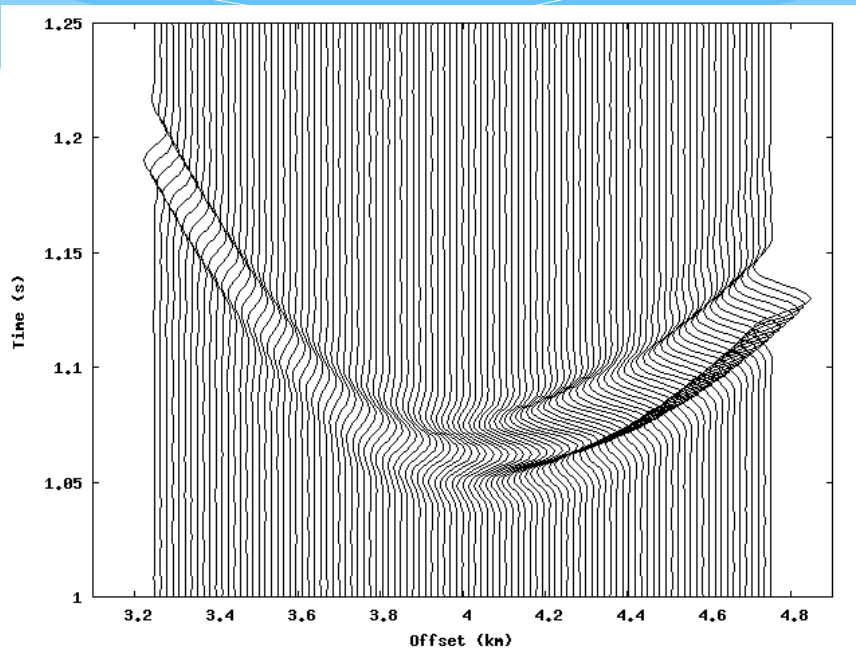
$$\mathrm{P}\left(\mathrm{x}, \mathrm{s}'\right)\left\langle \ldots \right\rangle = \iint_S G\left(\mathrm{x}, \mathrm{s}'\right) \mathrm{N}_{\mathrm{s}'} \left\langle \ldots \right\rangle dS(\mathrm{s}')$$

$$\mathrm{A}\left(\mathrm{s}, \mathrm{s}'\right)\left\langle \ldots \right\rangle = \iint_S h\left(\mathrm{s}, \mathrm{s}'\right) \mathrm{G}\left(\mathrm{s}, \mathrm{s}'\right) \mathrm{N}_{\mathrm{s}'} \left\langle \ldots \right\rangle dS\left(\mathrm{s}'\right)$$

Problems

Results





* Initial program – 27 hours
* Optimized (Core i7 + Intel MKL) – 2 hours
* GPU (Quadro FX 4000) - 23 min

Searching for wave patterns

* Seismic records
  float src[N_components][N_stations][s_length];
  N_components = 5
  N_stations = 3
  s_length = 4320000
* Pattern
  float ptrn[N_components][N_stations][p_length];
  p_length = 400
* **Task:**
  for each j=[0.. s_length-p_length] calculate F_j

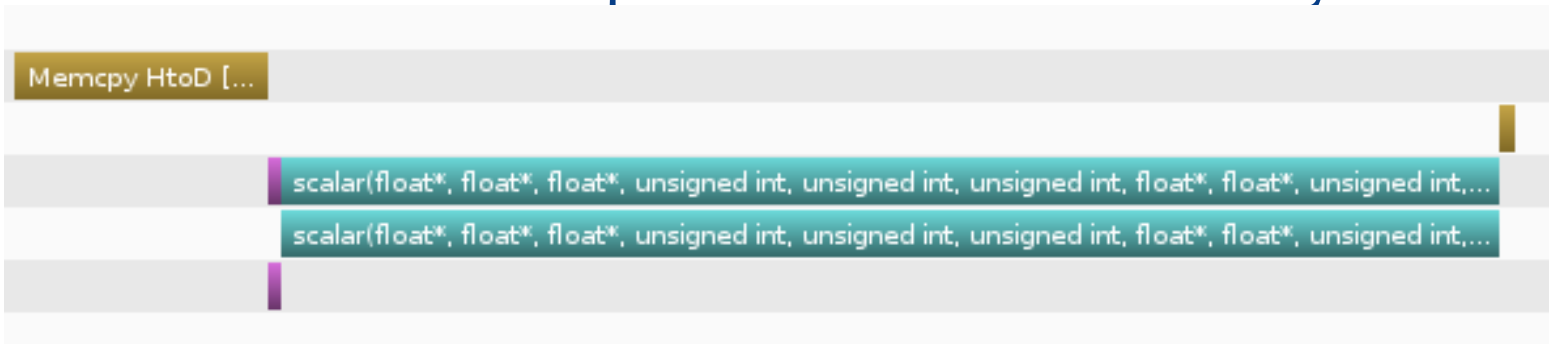
$$F_j = \sum_{k=0}^{N\_comp} \sum_{m=0}^{N\_stat} \frac{\sum_{p=0}^{p\_length} src_{km(j+p)} * ptrn_{kmp}}{\sqrt{\sum ptrn_p^2 \sum src_{j+p}^2}}$$

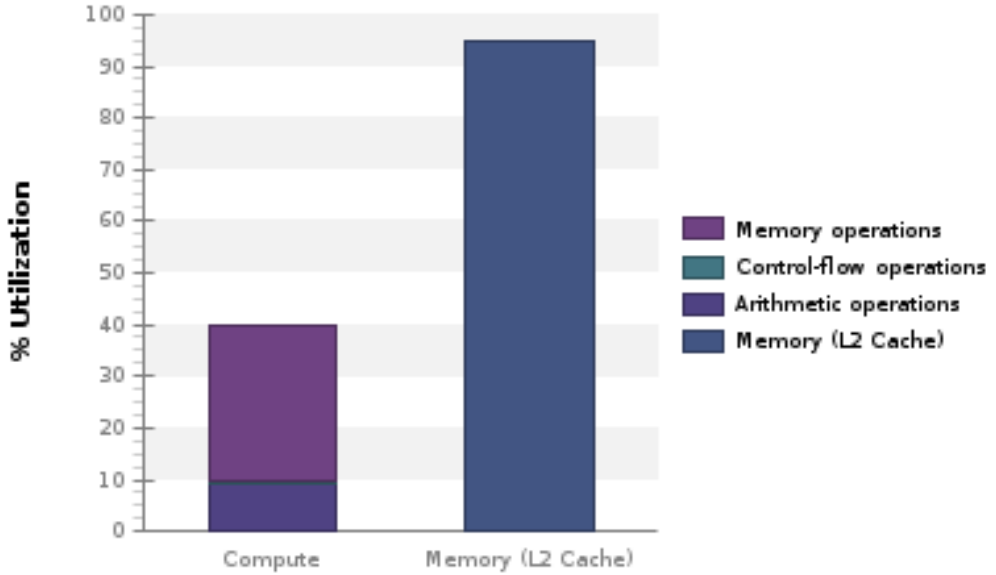
Searching for wave patterns

```
dim3 BS(1024);
dim3 GS(ceil(ncorr/(float)BS.x)); dim3 GS1(ceil((src_len*nsta*ncomp)/(float)BS.x));

complete = readPtrn(hptrn, ptrn_len, nsta, ncomp, pname);
for(int i=0; i< nsta*ncomp; i++){
        float res = 0.0;
        for(int j=0; j<ptrn_len; j++){
            res += hptrn[i*ptrn_len + j]*hptrn[i*ptrn_len + j];
        }
        hptrn2[i] = res;
}

complete = readSrc(hsrc, src_len, nsta, ncomp, dd, dname, fname);
// move data to GPU
…
sqr<<<GS1,BS>>>(dsrc, dsrc2, src_len*nsta*ncomp); // dsrc2[i] = dsrc[i]*dsrc[i];
scalar<<<GS,BS>>> (dsrc, dsrc2, dres, src_len, nsta, ncomp, dptrn, dptrn2, …. );
// download result
…
saveRes(hres, ncorr, fname);
```

Results

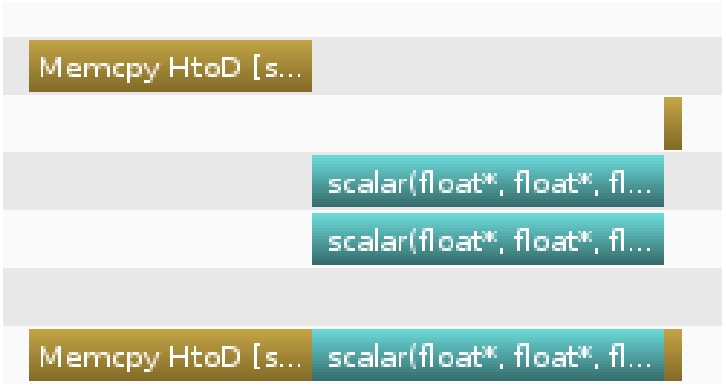
* Scalar kernel performance is bound by memory

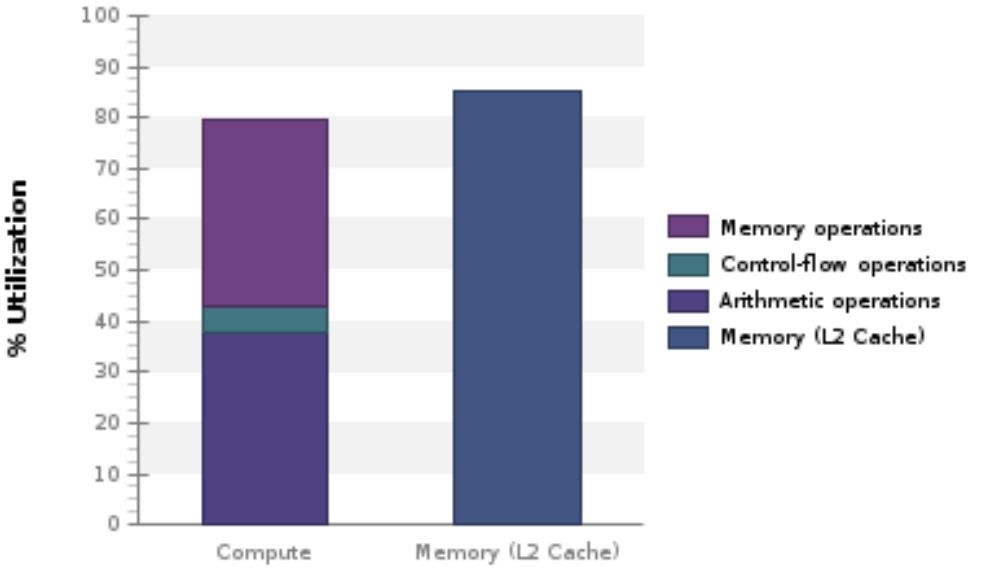




Results

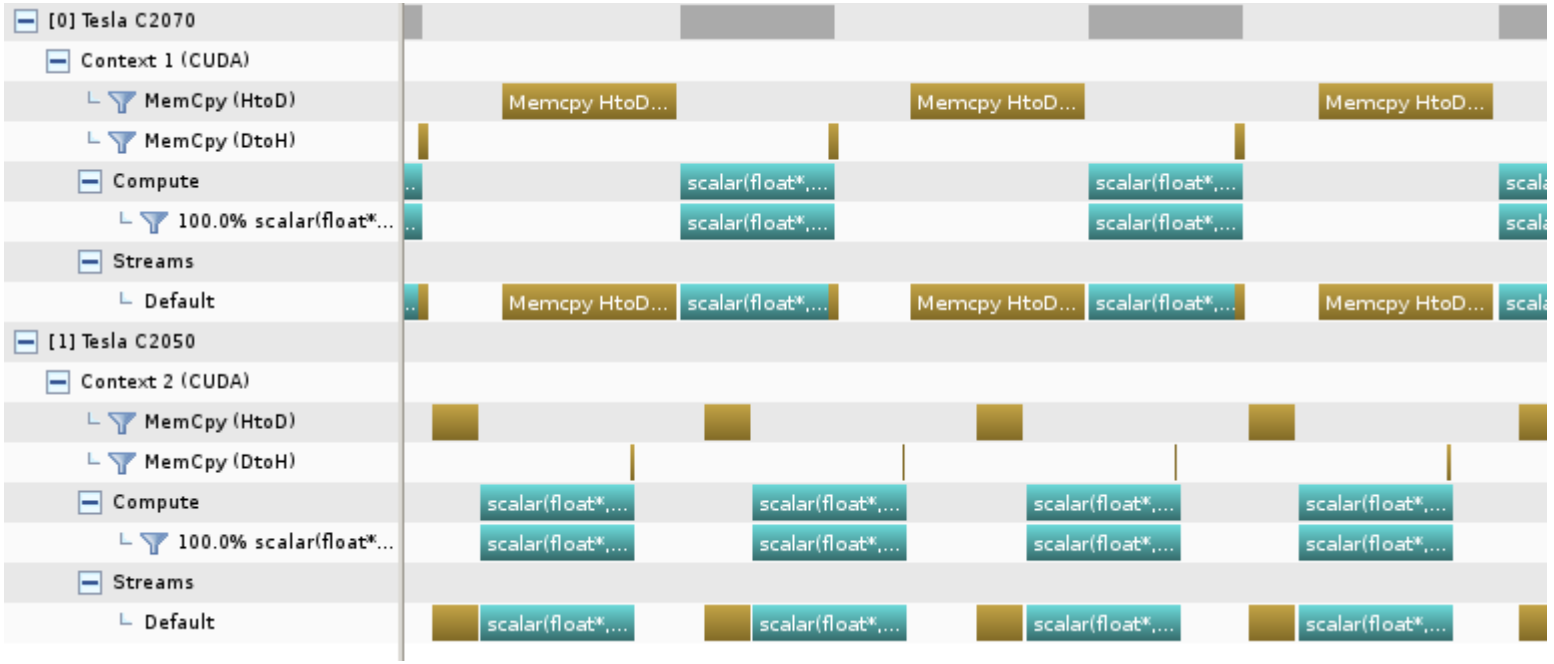
* Kernel SQR was removed
* Customer processed 2.5 years database during 30 hours (8GPUs). It was estimated that it would take 150 days on 4*12 CPU cores





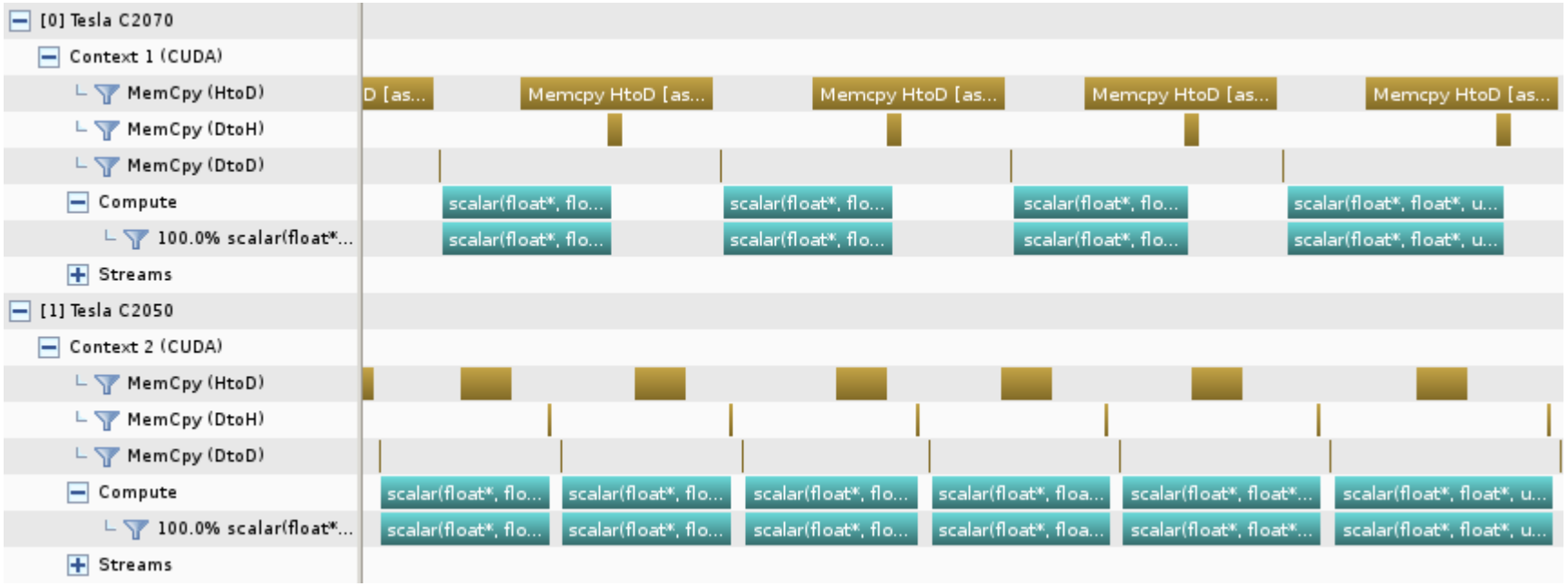
Results

* Multiple-GPU with OpenMP (70% speedup)
* Shared memory for patterns (30% speedup)
* Shared mem – 16K, L1 Cache – 64K



Results

* Asynchronous operations (35% speedup)
* Original – 9 sec per one series
* Optimized – 2.6 sec per 20 series



Tsunami waves

* MOST package (used in PMEL NOAA)
  * Generation (earthquake)
  * **Propagation** (transoceanic)
  * Inundation of dry land
* Problem size
  * 4’ grid (2600 x 2900 points)
  * ~8600 time steps (12 hours)
  * 7 hours on Pentium 4 2.8GHz

Example 3. Tsunami propagation

* Tsunami wave propagation is described with the following system:

$$H_t + (uD)_x + (vD)_y = Q,$$

$$u_t + uu_x + vu_y + gH_x = 0,$$

$$v_t + uv_x + vv_y + gH_y = 0,$$

* Where $H(x, y, t) = Q(x, y, t) + D(x, y, t)$; Q – wave height; D – depth profile, $u(x, y, t)$, $v(x, y, t)$ – velocities along x and y *axis*; g – gravity constant.

Main calculation loop

```
for(n=0; n<mmax; n++){ // for each time step
    for(j = 0; j < n1; j ++) {  // calculate along Y
        for(i=0; i<n2; i++) {
            qw[i] = u[j*n2 + i] - 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
            uw[i] = u[j*n2 + i] + 2.0f * sqrtf( 9.8f*q[j*n2 + i]);
            vw[i] = v[j*n2 + i];
        } //for i
        swater(uw, qw, vw, &d[j*n2], u1, q1, v1, &n2, h2, &grnd, &t);
        for(i=0; i<n2; i++){
            q[j*n2 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
            u[j*n2 + i] = (u1[i] + q1[i]) * .5f;
            v[j*n2 + i] = v1[i];
        } //for i
    } //  for j
    transpose(q, n2, n1); transpose(u, n2, n1); transpose(v, n2, n1);
    for(j=0; j<n2; j++){  // calculate along X
        for(i=0; i<n1; i++){
            qw[i] = v[j*n1 + i] - 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
            uw[i] = v[j*n1 + i] + 2.0f * sqrtf( 9.8f*q[j*n1 + i]);
            vw[i] = u[j*n1 + i];
        } //for i
        swater(uw, qw, vw, &dt[j*n1], u1, q1, v1, &n1, h1, &grnd, &t);
        for(i=0; i<n1; i++){
            q[j*n1 + i] = (u1[i] - q1[i]) * (u1[i] - q1[i]) /(9.8f*16.0f);
            v[j*n1 + i] = (u1[i] + q1[i]) * .5;
            u[j*n1 + i] = v1[i];
        } //for i
    } //  for j
    transpose(q, n1, n2);   transpose(u, n1, n2);  transpose(v, n1, n2);
    // getting data from "sensors" and save result
}
```

Move to GPU

```
for(int i=0; i<cfg.steps; i++){
      // calculate along X
       Invariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
       SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
       RInvariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
     // calculate along Y
       transpose<<<dimGrid, dimBlock>>>(d_qwdata, d_q1data, x_size, y_size);
       transpose<<<dimGrid, dimBlock>>>(d_uwdata, d_u1data, x_size, y_size);
       transpose<<<dimGrid, dimBlock>>>(d_vwdata, d_v1data, x_size, y_size);
     Invariants_Y<<<dimGrid_, dimBlock>>>(... , y_size, x_size);
       SWater_<<<dimGrid_, dimBlock>>>(..., y_size, x_size);
       RInvariants_Y<<<dimGrid_, dimBlock>>>(..., y_size, x_size);
     transpose<<<dimGrid_, dimBlock>>>(d_qwdata, d_q1data, y_size, x_size);
       transpose<<<dimGrid_, dimBlock>>>(d_uwdata, d_u1data, y_size, x_size);
       transpose<<<dimGrid_, dimBlock>>>(d_vwdata, d_v1data, y_size, x_size);
    // getting data from "sensors" and save result
       ...
}
```

Profiling result

Method	gld_ incoherent	gld_ coherent	gst_ incoherent	gst_ coherent
SWater_	4.92214e+06	105505	5.24688e+06	48144
Inv2	2.62344e+06	10935	5.24688e+06	43740
SWater_r	4.72e+06	91004	5.24688e+06	47772
RInv	2.62344e+06	10935	1.74896e+06	14580
Inv3	2.60116e+06	10935	5.20233e+06	43740

Moving to GPU

```
for(int i=0; i<cfg.steps; i++){
    // calculate along X
    Invariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    RInvariants_X<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    // calculate along Y
    Invariants_Y<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    SWater_r<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    RInvariants_Y<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // getting data from "sensors" and save result
    …
}
```

Moving to GPU

```
Inv1<<<dimGrid, dimBlock>>>(... , x_size, y_size);
for(int i=0; i<cfg.steps; i++){
    // calculate along X

    SWater_<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    Inv2<<<dimGrid, dimBlock>>>(... , x_size, y_size);
    // calculate along Y
    SWater_r<<<dimGrid_, dimBlock>>>(..., x_size, y_size);
    Inv3<<<dimGrid, dimBlock>>>(..., x_size, y_size);
    // getting data from "sensors" and save result
    …  RInv<<<dimGrid, dimBlock>>>(... , x_size, y_size);
}
```

Optimzation

- Remove transpose operation;
- Using textures;
- Alignment for 2D arrays.

Method	gld_ incoherent	gld_ coherent	gst_ incoherent	gst_ coherent
SWater_	0	459794	0	767260
Inv2	0	174897	0	699588
SWater_r	0	488749	0	761316
RInv	0	174900	0	233200
Inv3	0	174897	0	699588

Final results

* Original program– 3 sec per iteration
* Tesla C1060 – 0,02 sec per iteration

* Comparison is not correct
  * Original program was not optimized and executed sequentially
  * Algorithm for program on GPU was modified (calculation in invariants)
  * It worth to compare GPU vs. SandyBridge socket