

# Реализация решения системы уравнений мелкой воды для моделирования цунами

**М.М. Лаврентьев (мл.),  
А.А. Романенко**

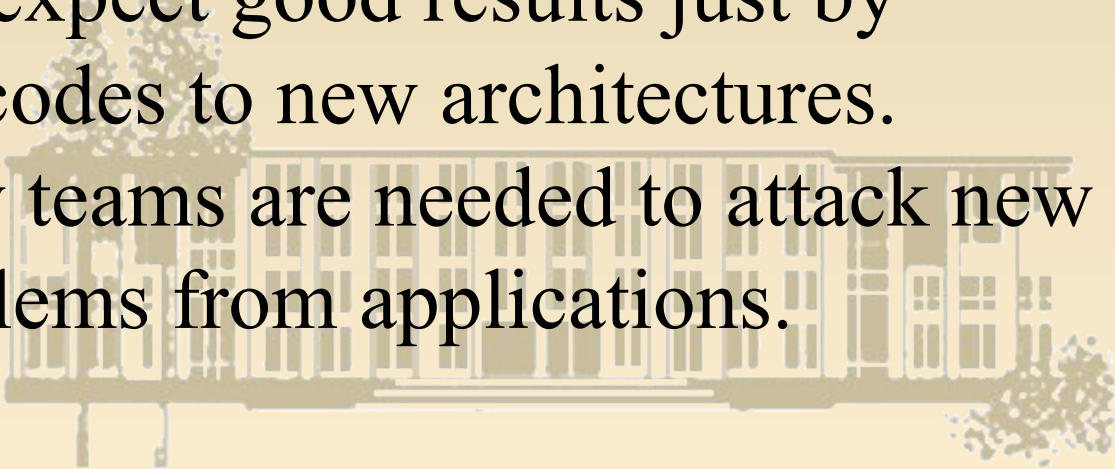
*Новосибирский государственный университет  
Институт математики им. С.Л. Соболева СО РАН*



# Mission

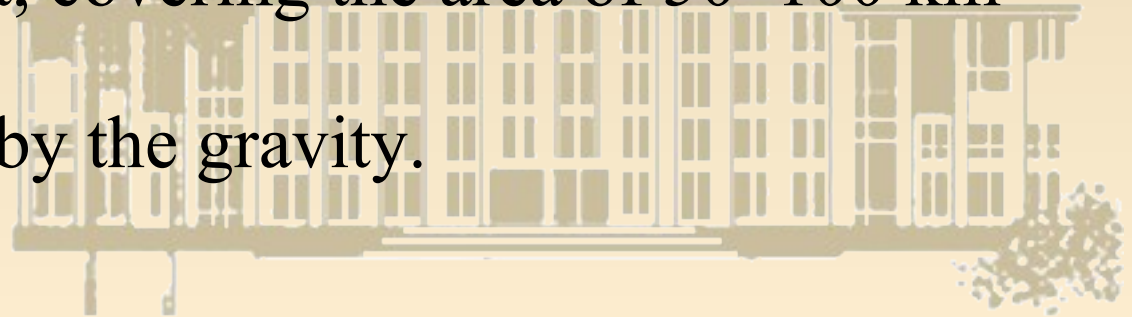
- To demonstrate valuable (up to the orders) performance gain, achieved by using of modern hardware facilities and software tools.

## Message

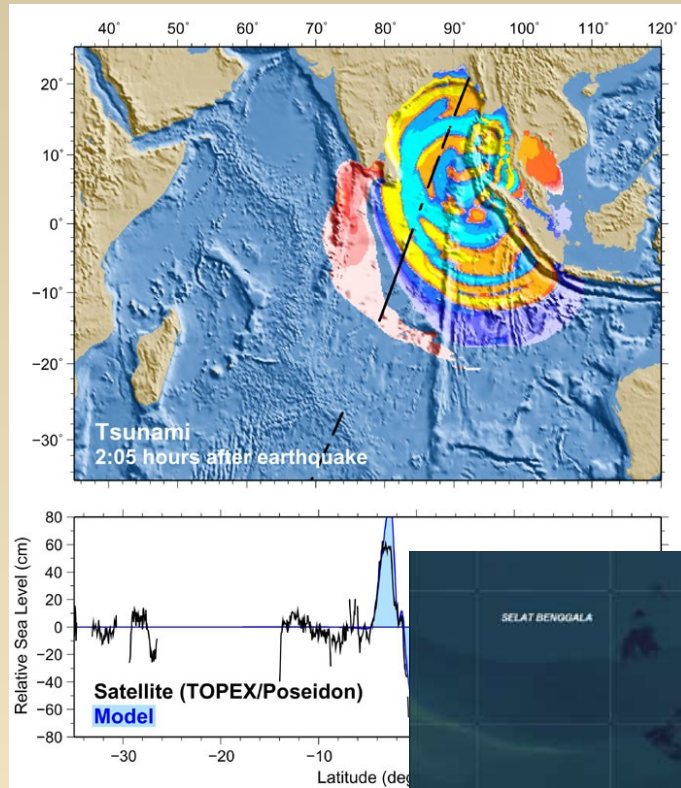
- New HW&SW facilities and tools are now available.
  - One can hardly expect good results just by porting existing codes to new architectures.
  - Interdisciplinary teams are needed to attack new challenging problems from applications.
- 

# Some basic information

- A tsunami is an oceanic gravity wave generated by submarine earthquake or other geological processes such as volcanic eruptions or landslides. Most tsunamis are caused by shallow large earthquakes and hence are distributed along the subduction zones.
- Tsunami is a Japanese word meaning harbor wave.
- Displacement of the sea bed causes (as water is practically incompressible) the initial disturbance at the sea surface. This could be, e.g., elevation of up to 1 m height, covering the area of 50\*100 km or more.
- Wave is driven by the gravity.



# Событие 26 декабря 2004





December 29, 2004



January 3, 2003

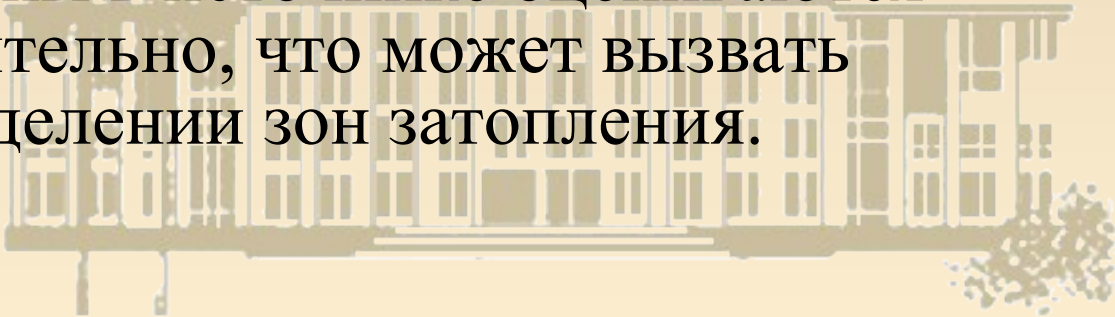






# Проблемы

- Прогноз цунамиопасности необходимо сделать возможно ранее (время для эвакуации и защиты).
- Вместе с тем, ложные тревоги, когда стоимость эвакуации значительно превосходит ущерб от цунами, **недопустимы.**
- Требуемые расчеты затруднительно провести в режиме реального времени и, тем самым, вовремя предупредить население об опасности.
- Параметры волны в источнике оцениваются лишь приблизительно, что может вызвать ошибки в определении зон затопления.





28 февраля 2010, Япония ждет прихода цунами от Чилийского землетрясения 27.02.2010, 5-го по силе за всю историю инструментальных наблюдений

| 区間<br>Section              | 方向<br>Direction | 状況<br>Status                | 原因<br>Cause   |
|----------------------------|-----------------|-----------------------------|---------------|
| 返子~久里浜<br>Tsuzuki~Kurihara | 上下線<br>Inbound  | 運転見合わせ<br>Operation stopped | 波の影響<br>Waves |

5 / 5

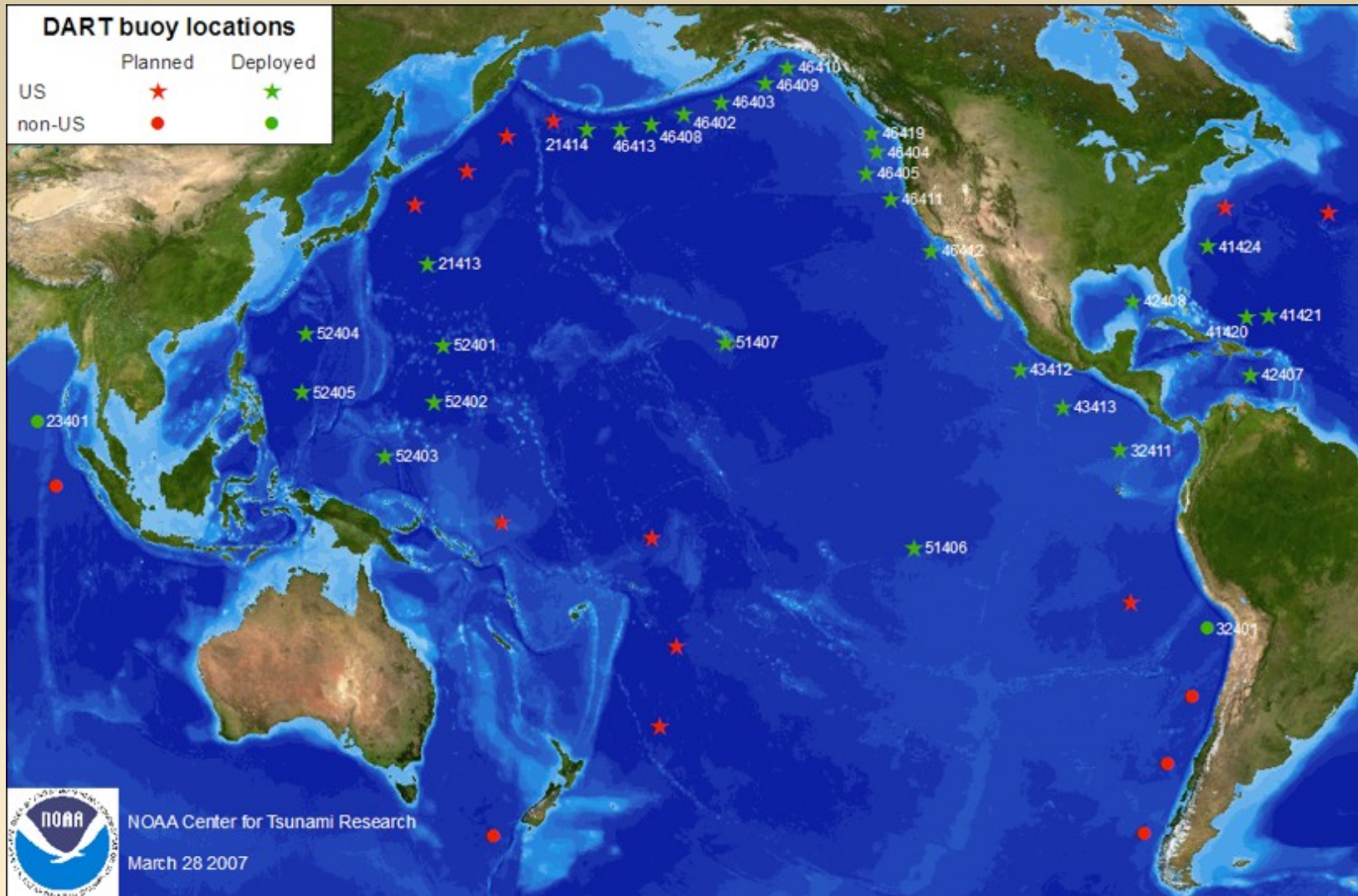


# Пути решения

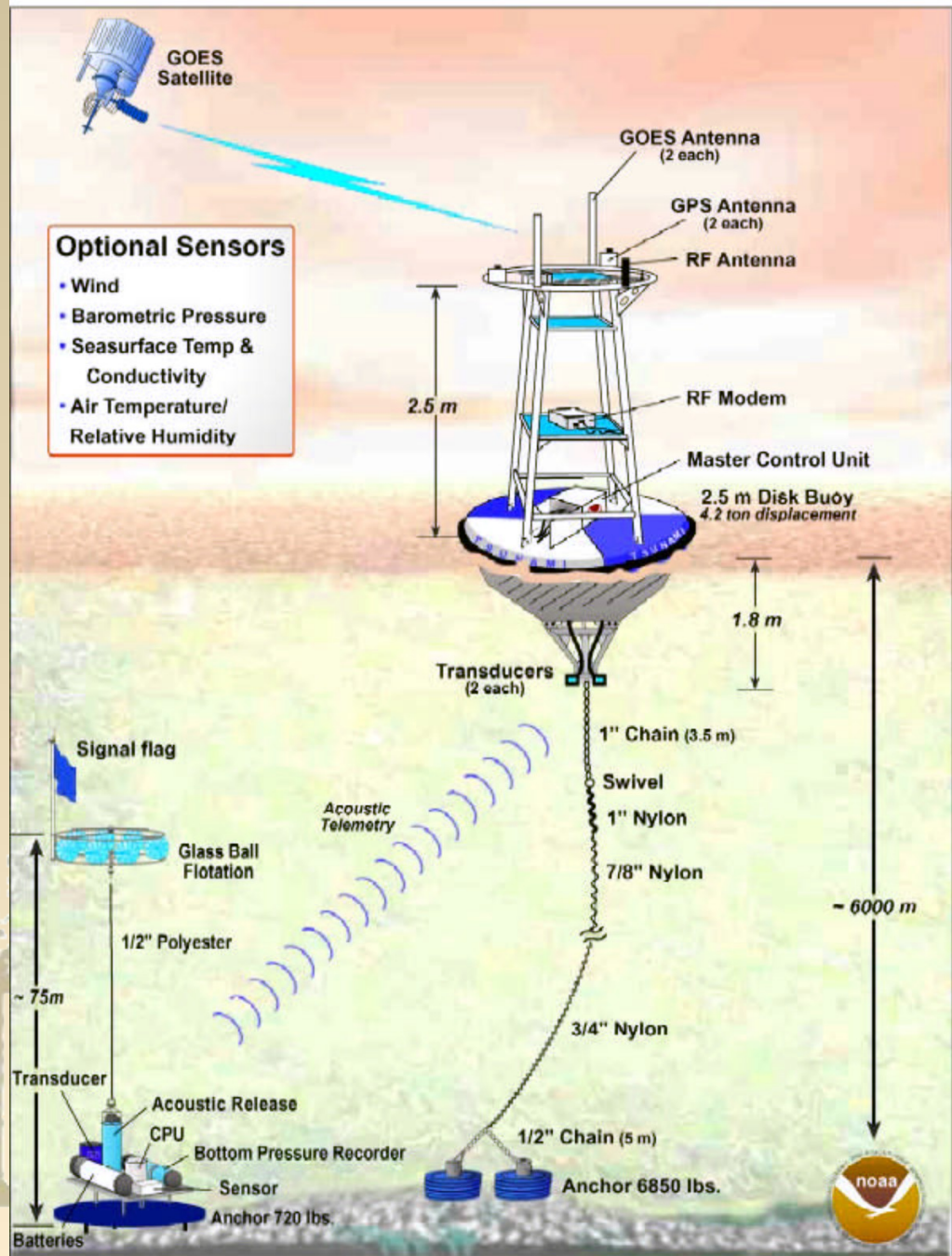
- Новые средства наблюдения – глубоководные гидрофизические станции серии DART (Deep-ocean Assessment and Reporting Tsunami), США. В режиме реального времени по спутниковым каналам передается запись профиля волны.
- GPS датчики при наличии островов близко к источнику.
- Наблюдения со спутников



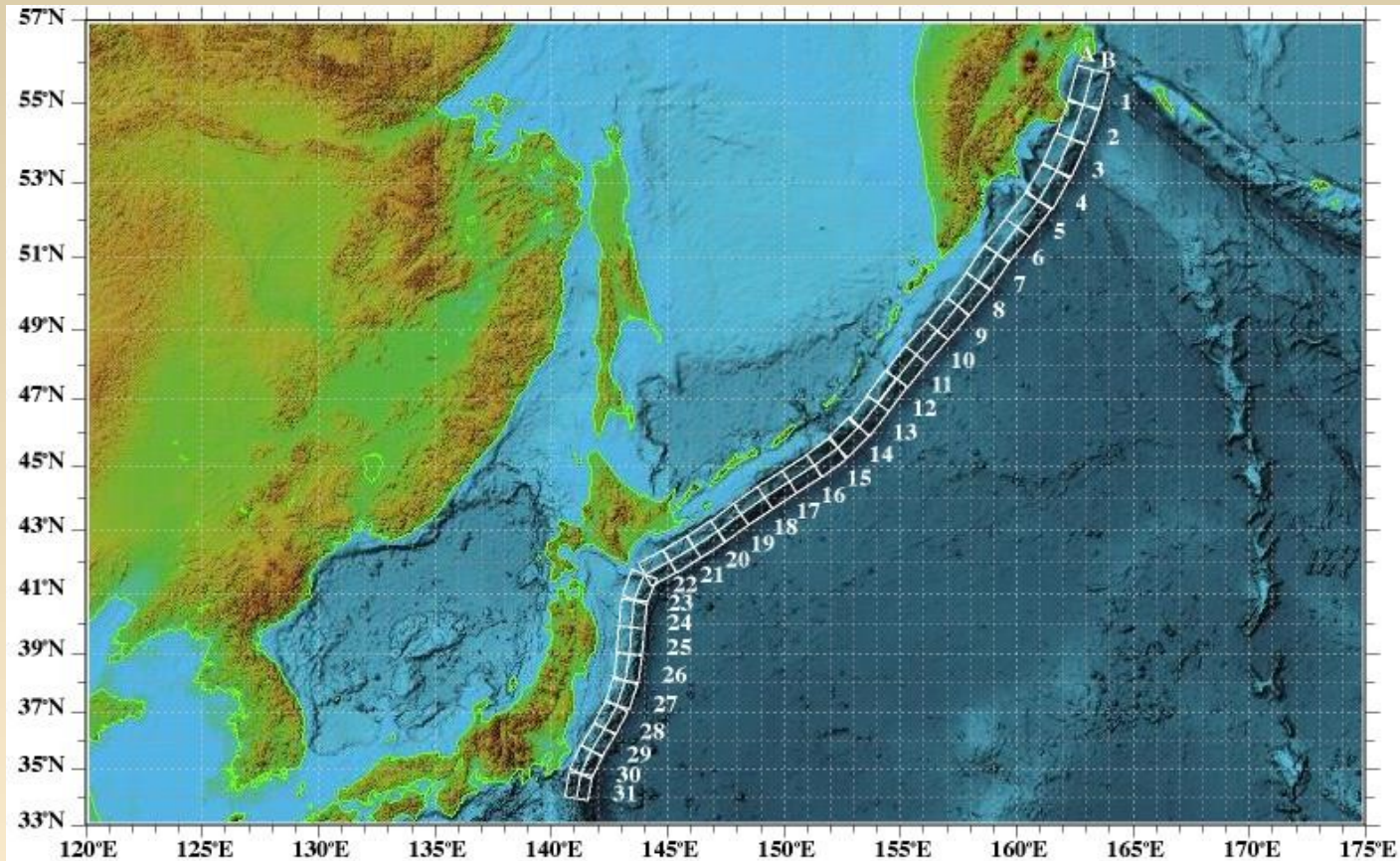
# Tsunami measuring stations, 2007



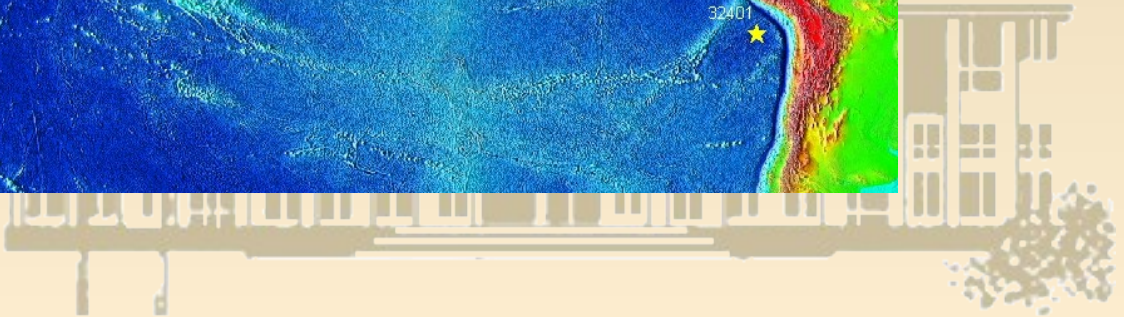
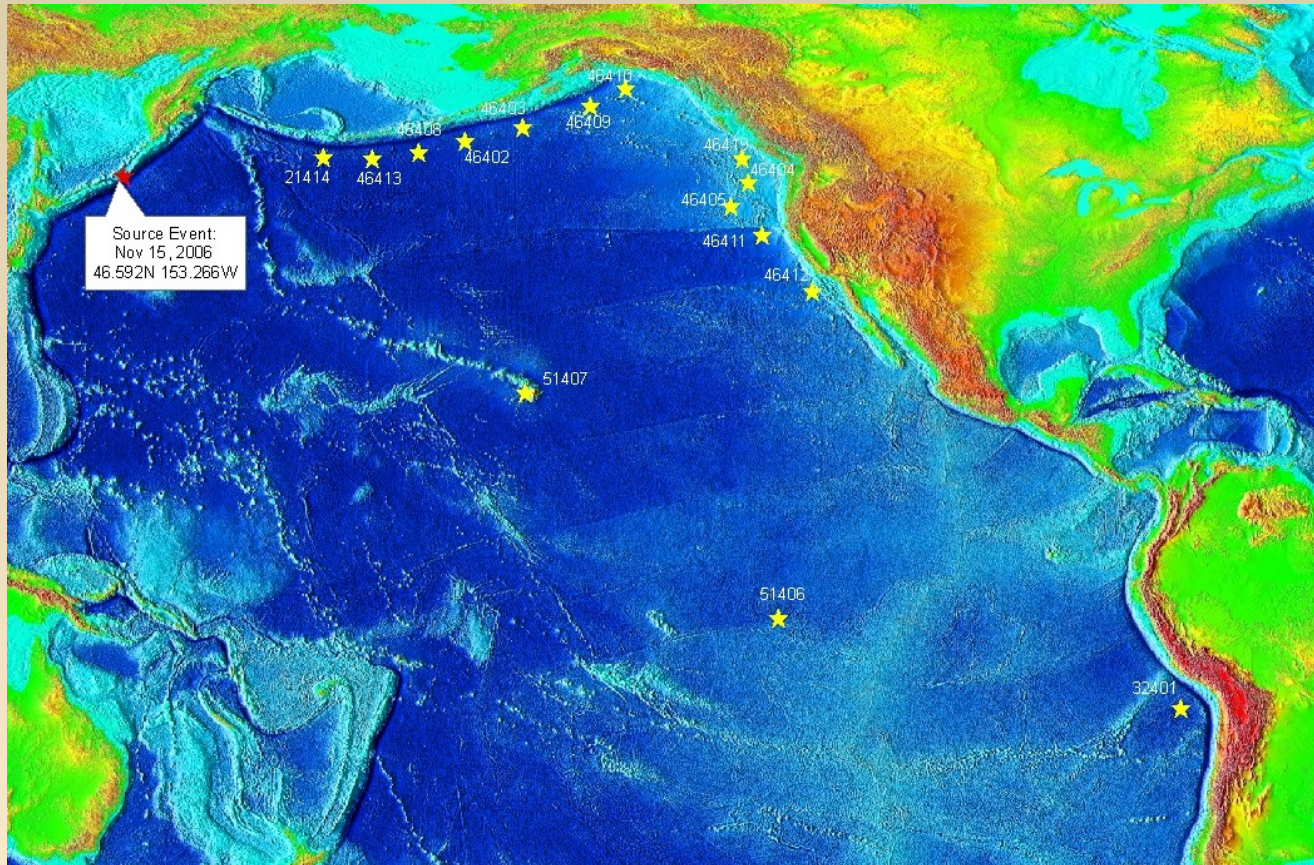
DART buoys  
operation scheme  
– Bottom Pressure  
Recorder and CPU  
are indicated in  
the below left  
corner.



# System of Unit Sources



# Earthquake source and DART stations



# Гидродинамика цунами

Из линейного приближения системы уравнений мелкой воды в одномерном случае получаем волновое уравнение для амплитуды волны:

$$\frac{\partial^2 h}{\partial t^2} = g \frac{\partial}{\partial x} \left( d \frac{\partial h}{\partial x} \right)$$

В случае постоянной глубины  $d$ , скорость волны определяется соотношением:

$$c = \sqrt{gd}$$

При глубине **4 км** получаем скорость **200 м/сек**, то есть **720 км/час** (крейсерская скорость пассажирского самолета)!

# SW tools for Tsunami Simulation

- MOST (Method of Splitting Tsunami) [1,2] allows real time tsunami inundation forecasting by incorporating real-time data from tsunameters. The model MOST is used most often in the United States for developing inundation maps [3]. The new web based community version of MOST is released with the name comMIT.
- TUNAMI N2 was originally authored by Imamura in 1993 for the Tsunami Inundation Modeling Exchange (TIME) program. It is a registered copyright of Professors Imamura, Yalciner and Synolakis and has been applied to several tsunami events [4,5].



# References

1. Titov, V.V., 1989, Numerical modeling of tsunami propagation by using variable grid. *Proceedings of the IUGG/IOC International Tsunami Symposium*, 46-51. Computing center Siberian Division USSR Academy of Sciences, Novosibirsk, USSR.
2. Titov V.V. and Synolakis, C.E., 1998 Numerical modeling of tidal wave runup, *Journal of Waterway, Port, Coastal and Ocean Engineering*, 124(4), 157-171.
3. Borrero, J.C., Cho, S. Moore, J.E, Richardson, H.W., Synolakis, C.E., 2005 Could it happen here, *Civil Engineering*, 75(4), 55-67.
4. Shuto, N., C. Goto, F. Imamura (1990), Numerical simulation as a means of warning for near field tsunamis, *Coastal Engineering in Japan*, 33(2), 173-193.
5. Yalciner A. C. Alpar B., Altinok Y., Ozbay I., Imamura F., (2002), "Tsunamis in the Sea of Marmara: Historical Documents for the Past, Models for Future" *Marine Geology*, 2002, 190, pp:445-463

# **MOST (Method Of Splitting Tsunami)**

- MOST consists of numerical simulation codes capable of simulating three stages of tsunami evolution:
  1. **Generation** (earthquake),
  2. **Propagation** (transoceanic),
  3. **Run up** (inundation of dry land).

**Here we address only the problem of wave propagation at deep water area.**



# **MOST – some explanations**

- Wave propagation is described by the system of hyperbolic partial differential equations
- Assumptions:
  - shallow water
  - non-linear approximation
- Equivalent transform of the governing hyperbolic system into canonical form
- In this form it is possible to split calculation of wave propagation by dimensions



# MOST – math model

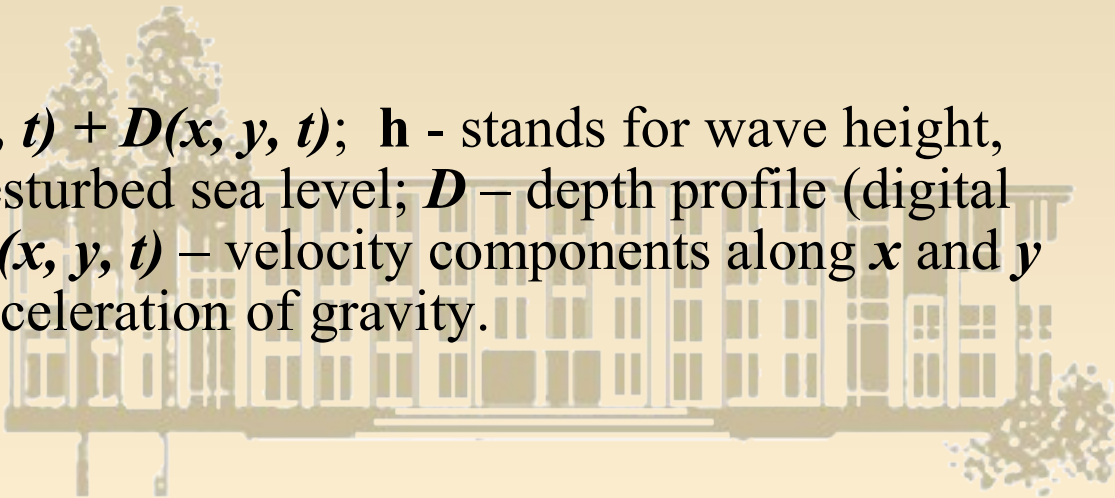
- Nonlinear hyperbolic “shallow water” system is used in the form

$$q_t + (uq)_x + (vq)_y = 0,$$

$$u_t + uu_x + vu_y + gq_x = gD_x,$$

$$v_t + uv_x + vv_y + gq_y = gD_y,$$

where  $q(x, y, t) = h(x, y, t) + D(x, y, t)$ ;  $h$  - stands for wave height, calculated from the undisturbed sea level;  $D$  – depth profile (digital bathymetry),  $u(x, y, t)$ ,  $v(x, y, t)$  – velocity components along  $x$  and  $y$  axis, respectively;  $g$  – acceleration of gravity.



# MOST – some explanations

$$\frac{\partial z}{\partial t} + A \frac{\partial z}{\partial x} + B \frac{\partial z}{\partial y} = F$$

The above system could be presented as

$$z = \begin{pmatrix} u \\ v \\ q \end{pmatrix}, \quad A = \begin{pmatrix} u & 0 & g \\ 0 & u & 0 \\ q & 0 & u \end{pmatrix}, \quad B = \begin{pmatrix} v & 0 & 0 \\ 0 & u & g \\ 0 & q & u \end{pmatrix}, \quad F = \begin{pmatrix} gD_x \\ gD_y \\ 0 \end{pmatrix}.$$

Splitting method for numerical treatment is based on two auxiliary systems, each depending on **one** spatial variable:

$$\frac{\partial \varphi}{\partial t} + A \frac{\partial \varphi}{\partial x} = F_1, \quad 0 \leq x \leq X; \quad \frac{\partial \psi}{\partial t} + B \frac{\partial \psi}{\partial x} = F_2, \quad 0 \leq y \leq Y$$

$$F_1 = \begin{pmatrix} gD_x \\ 0 \\ 0 \end{pmatrix}, \quad F_2 = \begin{pmatrix} 0 \\ gD_y \\ 0 \end{pmatrix}.$$

# SoftWarea Algorithm – One Iteration

- Reading input data (shape of ocean floor, shape of water surface)
- Eliminating “one-pixel” islands
- Calculations (for each time step)
  - Prepare invariants along **X** axis for each row
  - **Do** calculations (use invariants only)
  - Converting new values of invariants back
  - Prepare invariants along **Y** axis for each row
  - **Do** calculations (use invariants only)
  - Converting new values of invariants back

# Grid Size and Sequential Execution Time

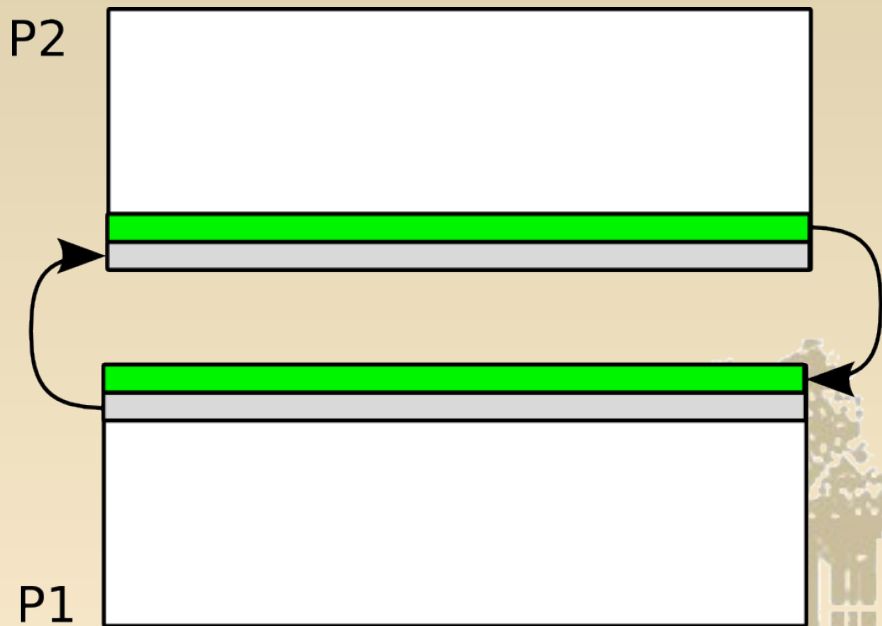
- Calculation domain is 2581x2879 points (4' grid). This size is quite enough to cover Pacific ocean. All initial data are required could be placed in 120MB.
- The number of time steps is compared to 8600 to simulate tsunami wave propagation during 24 hours period.
- One time step requires approximately  **$110 \times 2581 \times 2879 \sim 10^9$**  elementary math operations (+, -, \*, /).
- The one time step takes about 3 seconds (for the original sequential program execution) at **P4 2,8GHz hardware.**
- It takes  $3 \text{ sec} * 8600 = \sim 7 \text{ hours}$  for total tsunami propagation modeling.

# ”Direct” Parallel Implementation for MPI (distributed memory)

- Prepare invariants along **X** axis for each row
- Do calculations (use invariants only)
- Converting new invariants back
- **MPI\_AllToAll communication (3 arrays)**
- Prepare invariants along **Y** axis for each row
- Do calculations (use invariants only)
- Converting new invariants back
- **MPI\_AllToAll communication (3 arrays)**



# Alternative parallelization strategy for MPI



- Do calculations along X axis (P1, P2)
- Do calculations along Y axis (P1)
- Pass the edge (grey bar) to the next process (P1)
- Do calculations along Y axis (P2)
- Pass the edge (green bar) to the previous process (P2)

# Parallelization for OpenMP (shared memory)

- **#pragma omp parallel for**

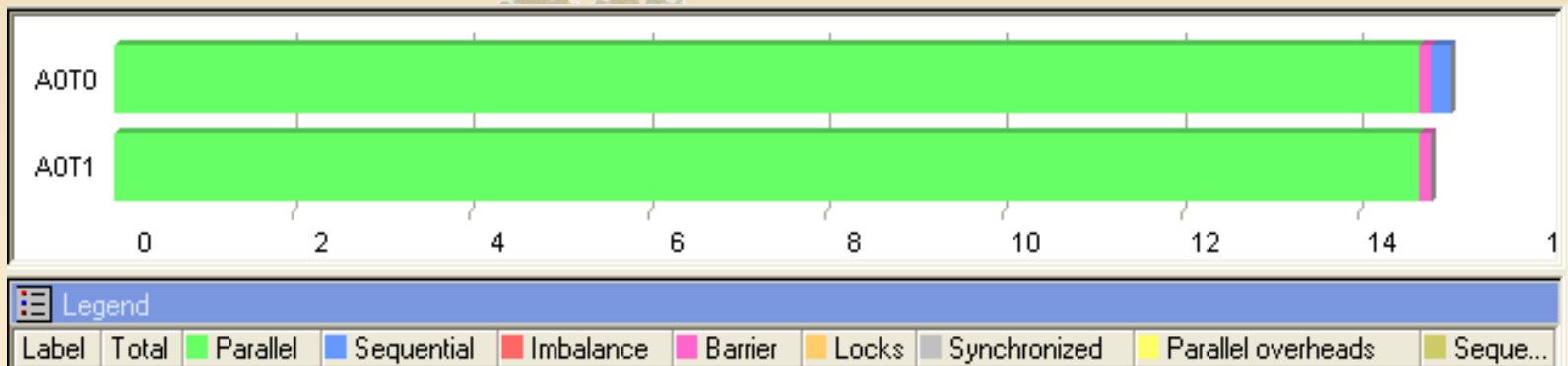
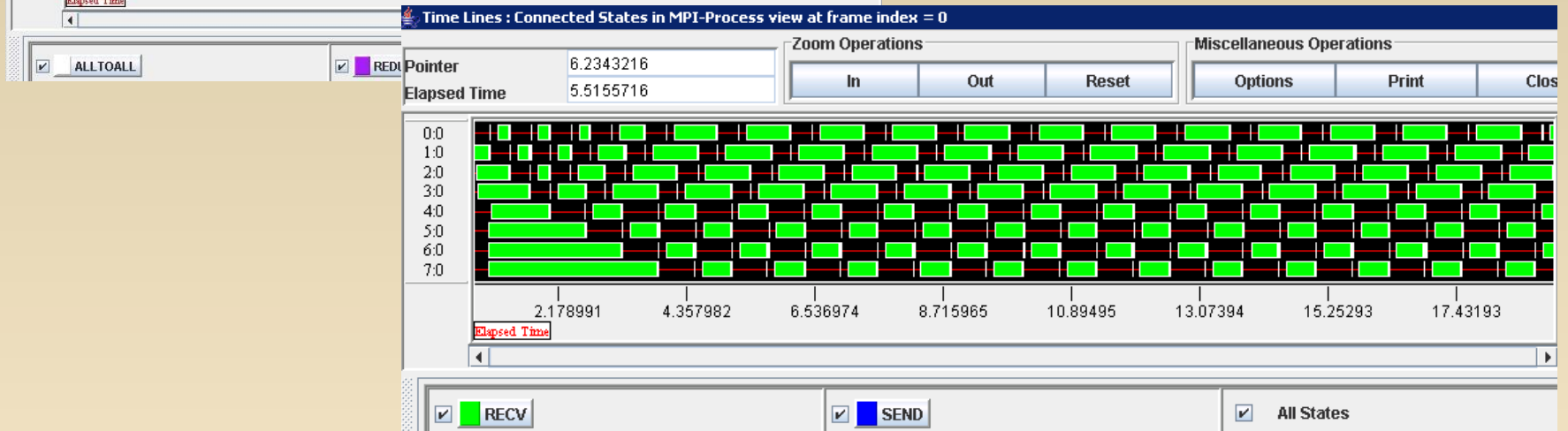
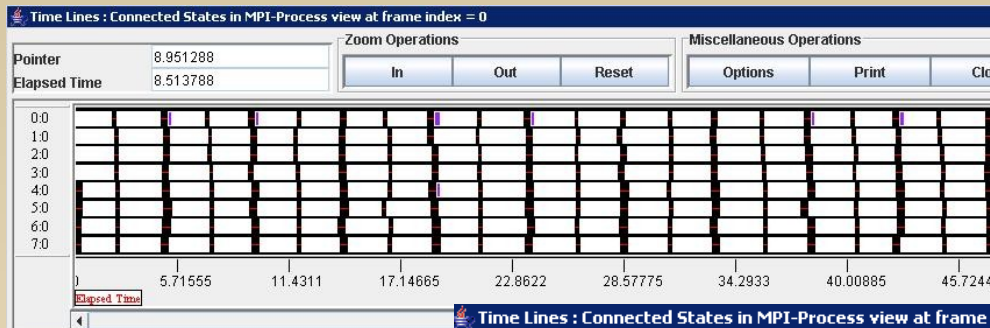
- Prepare invariants along **X** axis for each row
- Do calculations (use invariants only)
- Converting new invariants back

- **#pragma omp parallel for**

- Prepare invariants along **Y** axis for each row
- Do calculations (use invariants only)
- Converting new invariants back



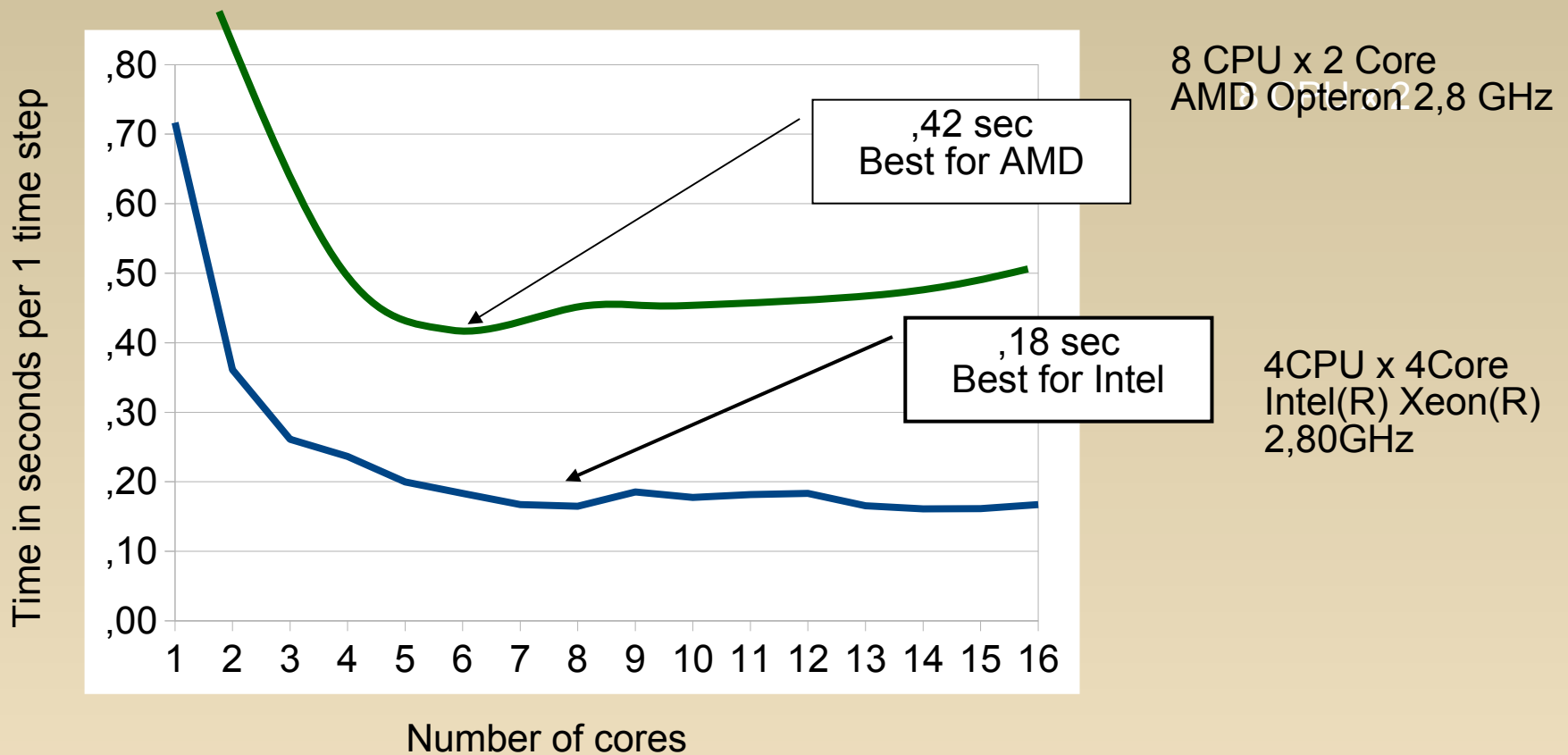
# CPU vs Memory time



# Performance results vs HW's

- Sun Fire X4600 M2 16 CPUs, Dual-Core AMD Opteron (2593.152Mhz), RAM 64GB
- Sun Blade X8420 Server Module, 8 CPUs, Dual-Core AMD Opteron (2800.264Mhz), RAM 32GB
- Dual Intel Xeon HT (3200 Mhz), RAM 4GB
- 8 Node Linux cluster, Dual Intel Xeon HT (2800 Mhz), RAM 4GB, Gigabit Ethernet
- 6 Node Linux cluster, Dual AMD Opteron (1400 Mhz), RAM 2GB, Fast Ethernet





## Intel(R) Fortran Compiler Professional for applications running on Intel(R) 64, Version 11.0

```
ifort -V -ipo -O3 -xsse4.2 par_tsunami.f90 -openmp -fno-alias -no-prec-div -opt-prefetch -auto -static
```

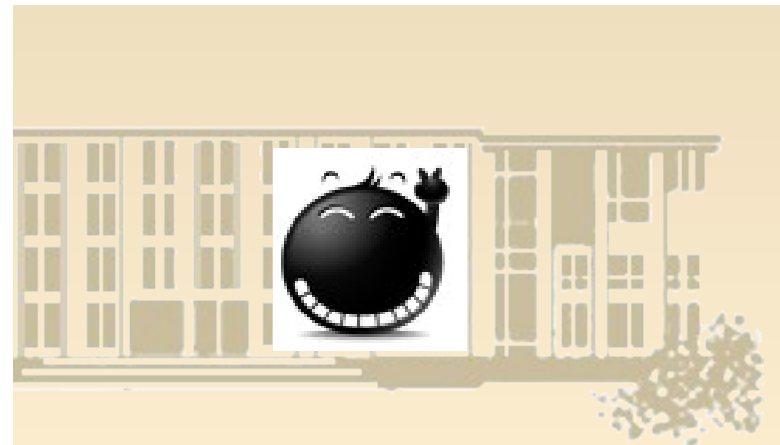
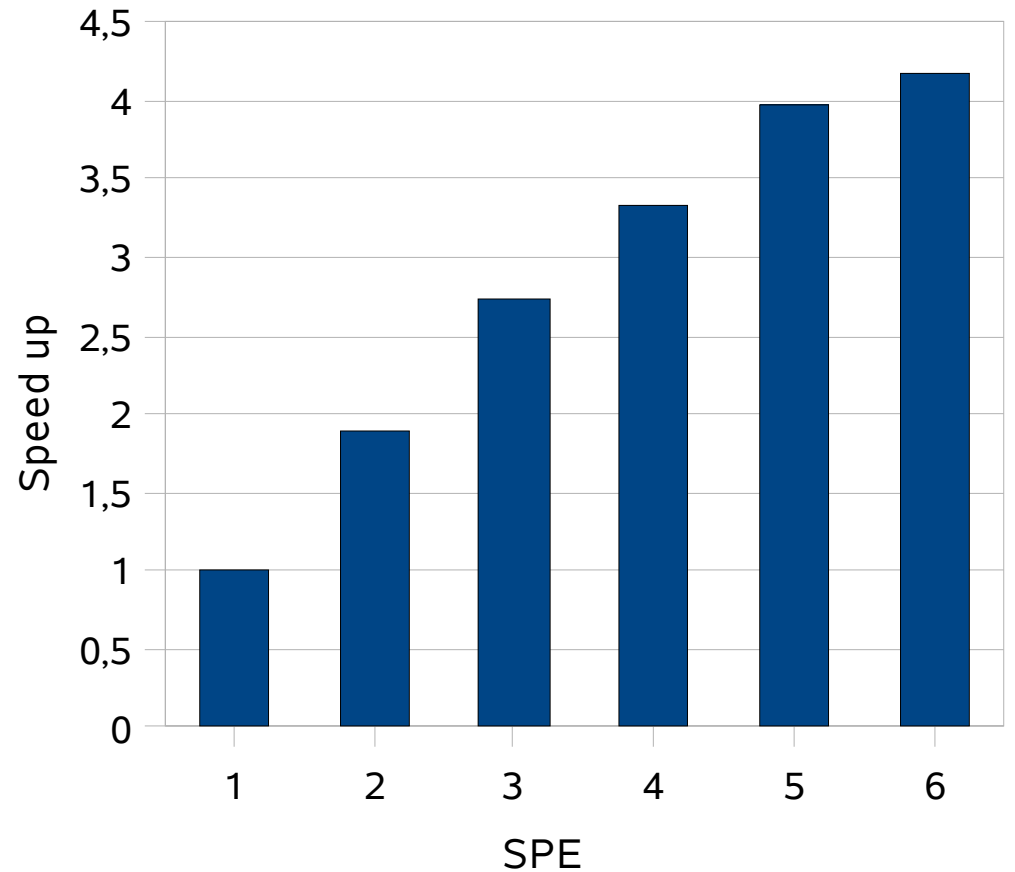
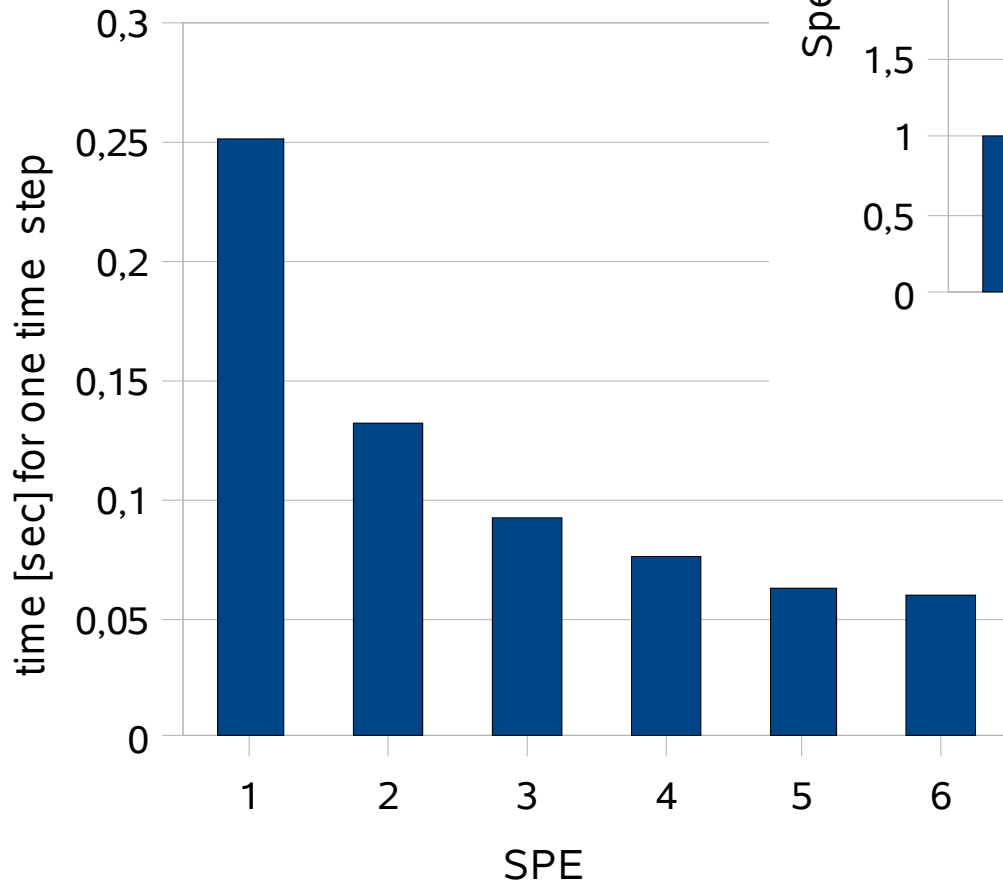
**f95: Sun Fortran 95 8.2 2005/10/13**

```
f95 -O3 -xopenmp=parallel -fast -native -xvector=simd par_tsunami.f90
```

# Performance Optimization

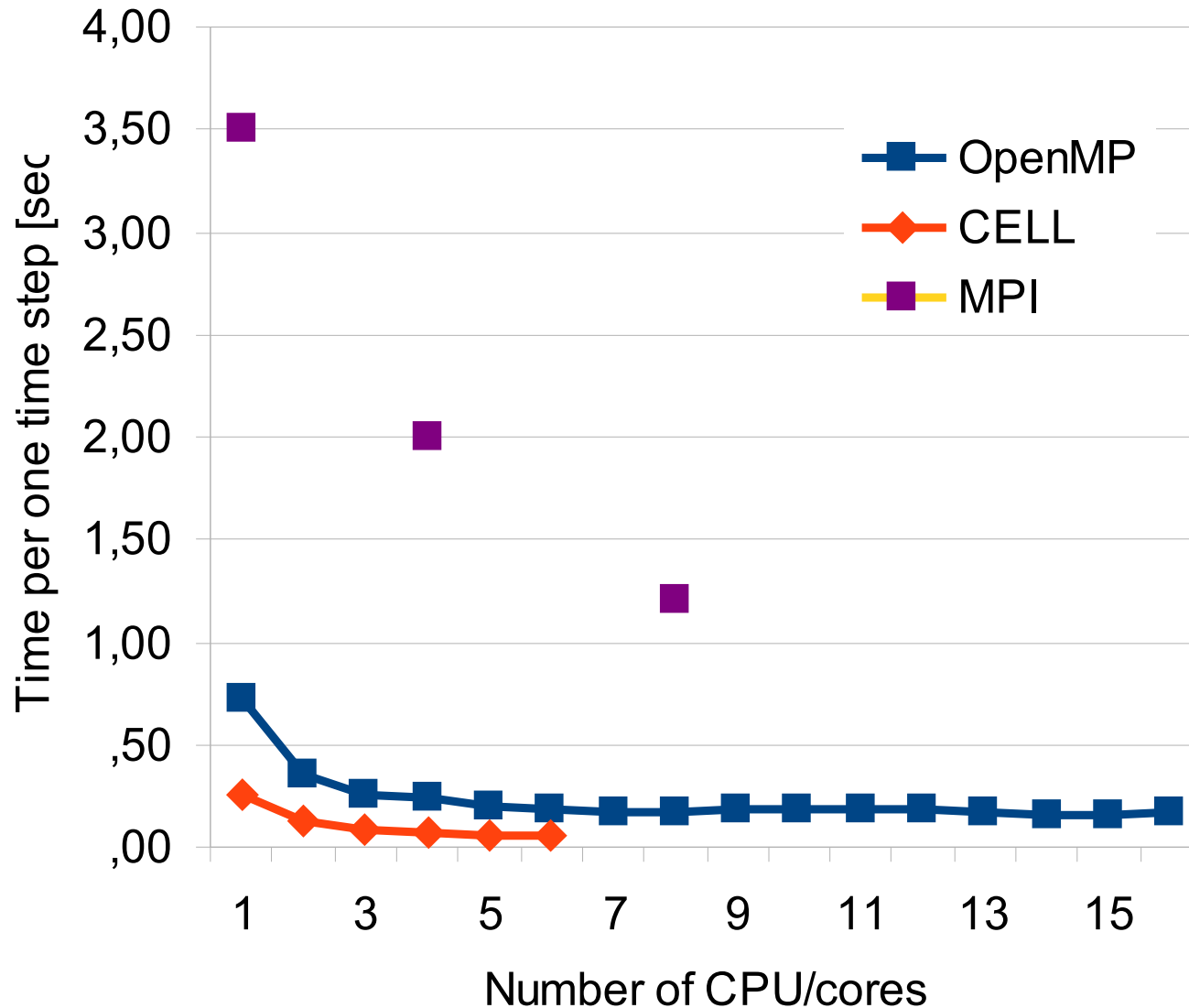
- В процессе данной работы был выполнен перенос кода программы с языка Fortran на язык C с оптимизацией под вычислительную платформу Intel. В итоге при использовании системы над общей памятью SMP 4 x Intel Xeon CPU X7350, 2.93GHz с компилятором icc v11.1 среднее время выполнения одной вычислительной итерации алгоритма сократилось в 16 раз, и итоговая производительность на сетке размером 2581 x 2879 составила 5.84 итераций в секунду. Измерения производительности при использовании процессора Intel Core i7 показали производительность 8.93 итераций в секунду.

# IBM CELL BE performance

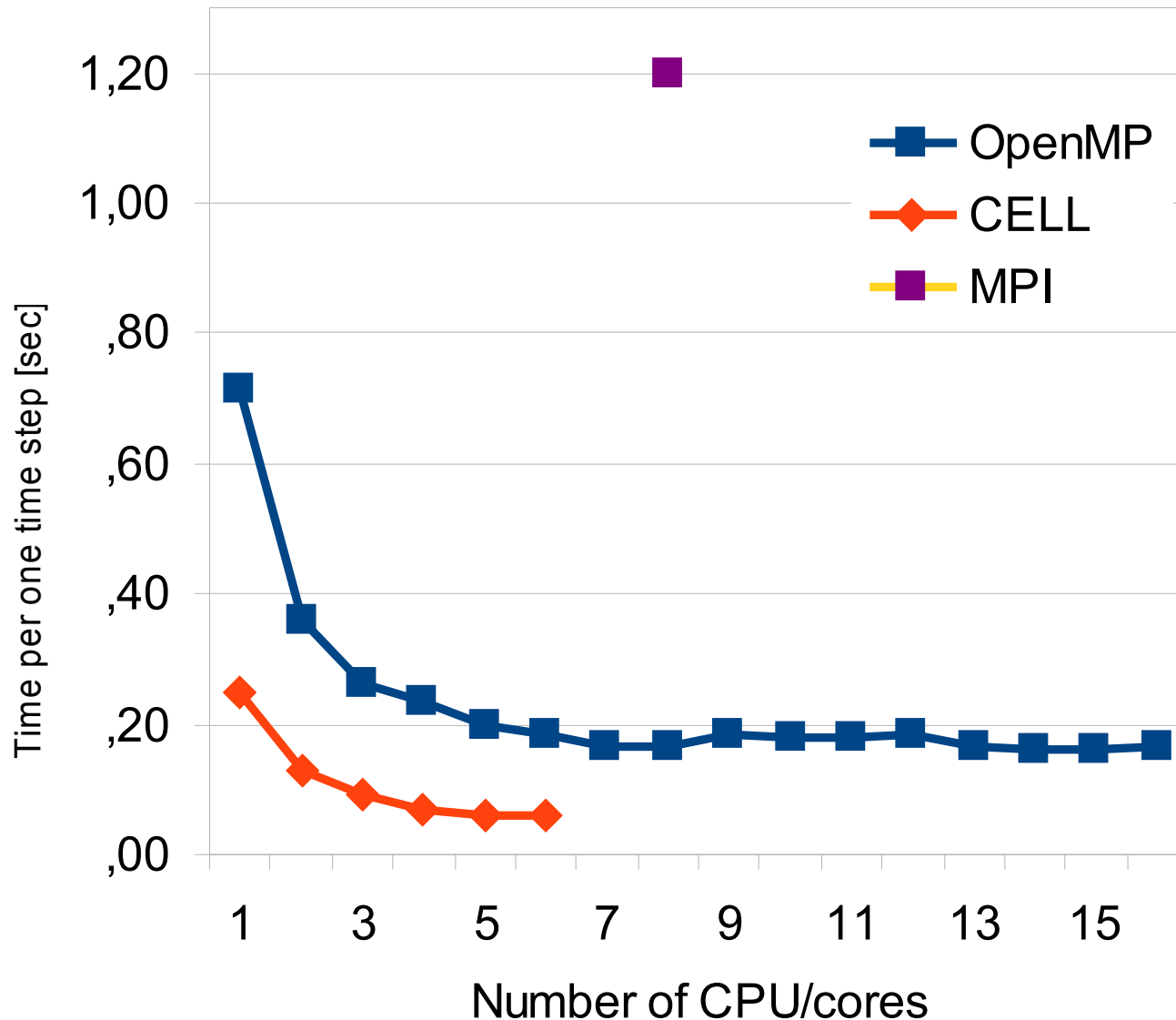




# Comparing Scalability



# Performance and Scalability



# Graphics Processing Units



- Can be used as co-processor
- Can process in parallel different data when using the same algorithm
- Up to 1000 threads simultaneously

λ TPC - Texture Processor Cluster

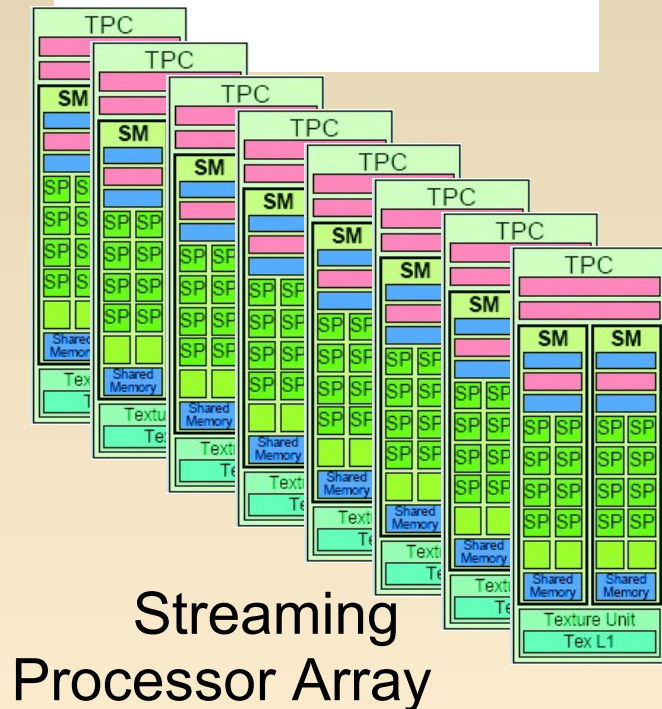
λ SM — Streaming Multiprocessor

λ Multi-threaded processor core

λ Fundamental processing unit for CUDA thread block

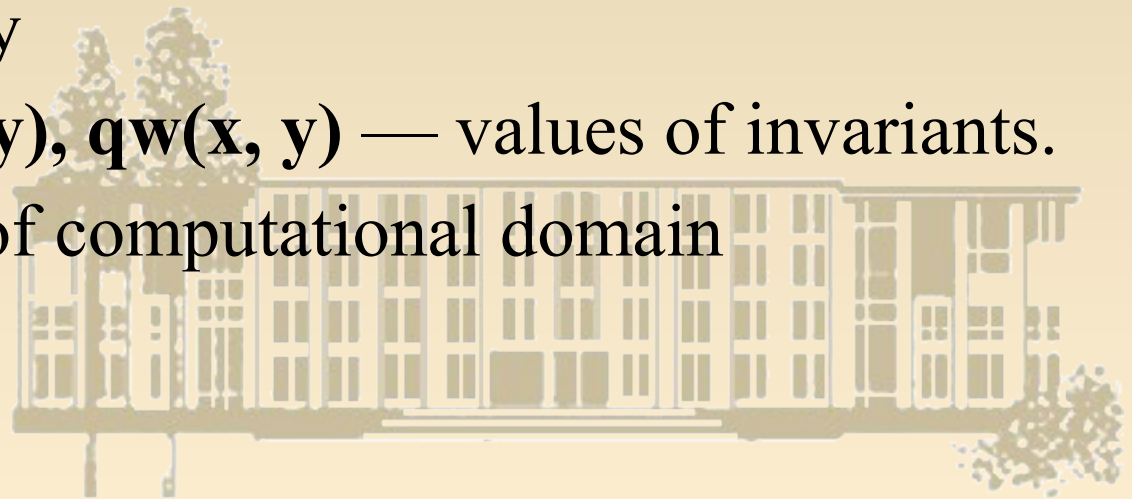
λ SP — Streaming Processor

λ Scalar ALU for a single CUDA thread



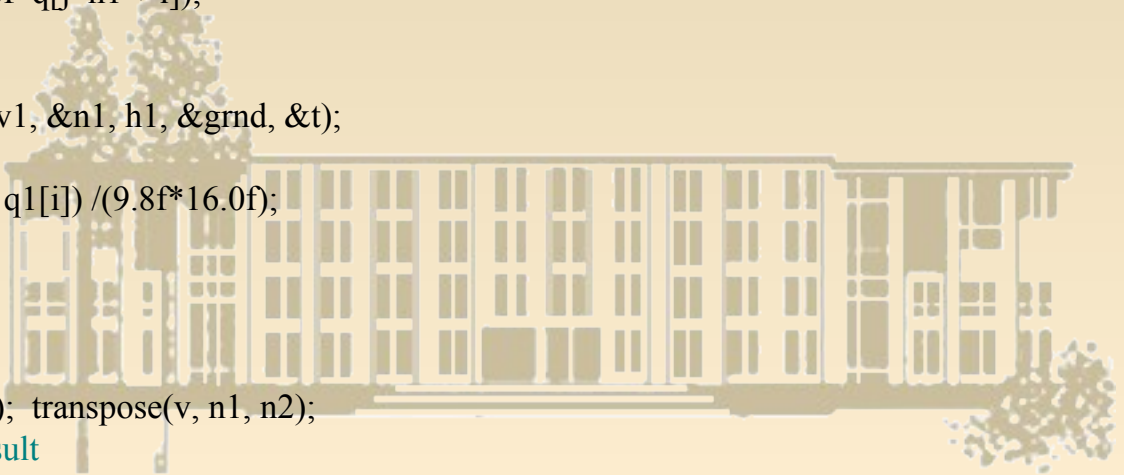
# Data in Use

- $D(x, y)$  - depth profile (digital bathymetry)
- $h(x, y)$ - stands for wave height, calculated from the undisturbed sea level
- $q(x, y) = h(x, y) + D(x, y)$
- $u(x, y), v(x, y)$  – velocity components along x and y axis, respectively
- $uw(x, y), vw(x, y), qw(x, y)$  — values of invariants.
- $n1, n2$  — sizes of computational domain



# Main loop of sequential programm

```
for(n=0; n<mmax; n++){
  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
}
```



# The main loop, adopted for GPU(CUDA)

- 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\_qldata, x\_size, y\_size);  
  transpose<<<dimGrid, dimBlock>>>(d\_uwdata, d\_uldata, x\_size, y\_size);  
  transpose<<<dimGrid, dimBlock>>>(d\_vwdata, d\_vldata, 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\_qldata, y\_size, x\_size);  
  transpose<<<dimGrid\_, dimBlock>>>(d\_uwdata, d\_uldata, y\_size, x\_size);  
  transpose<<<dimGrid\_, dimBlock>>>(d\_vwdata, d\_vldata, y\_size, x\_size);
  - *// getting data from "sensors" and save result*  
  …
- }

# Performance Analyzers

- Gprof — GNU profiler for Unix
- VTune — Intel tool for Intel CPU
- CodeAnalyzer — AMD tool
- CELL BE simulator — IBM tool
  
- There exists special tools (profiler) for NVidia GPUs

Performance analysis (profiling) is the investigation of a program's behavior using information gathered as the program executes (e.g. I/O memory operations, number of system calls, etc.)

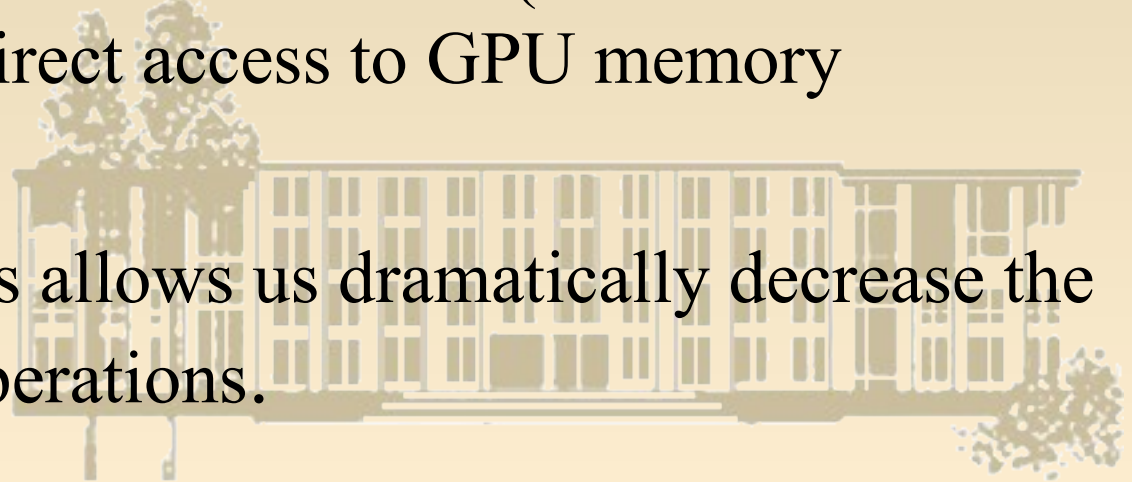
# Number of operations before optimization

| 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        |



# Optimization techniques

- Move **sqrtdf()** function out of the main loop, as it takes valuable time to calculate and do it not accurate
- Rewrite **swater()** function which allows us to eliminate **transpose()** function
- Align ends of arrays' rows
- Rewrite some code to use textures (texture could be cached) instead direct access to GPU memory
- The last two items allows us dramatically decrease the number of I/O operations.



# Number of operations after optimization

| 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       |

eliminate **millions** incoherent load and store operations



# Integrated Performance Results

|                 | Time per one time step |                    |
|-----------------|------------------------|--------------------|
|                 | Before optimization    | After optimization |
| Initial program | <b>3000 ms</b>         | <b>3000 ms</b>     |
| AMD             | <b>1800 ms</b>         | <b>420 ms</b>      |
| Intel           | <b>300 ms</b>          | <b>180 ms</b>      |
| IBM CELL BE     | <b>5000 ms</b>         | <b>60 ms</b>       |
| Tesla C1060     | <b>530 ms</b>          | <b>20 ms</b>       |

# Contacts

Mikhail Lavrentiev, Jr.

Faculty of Information Technologies

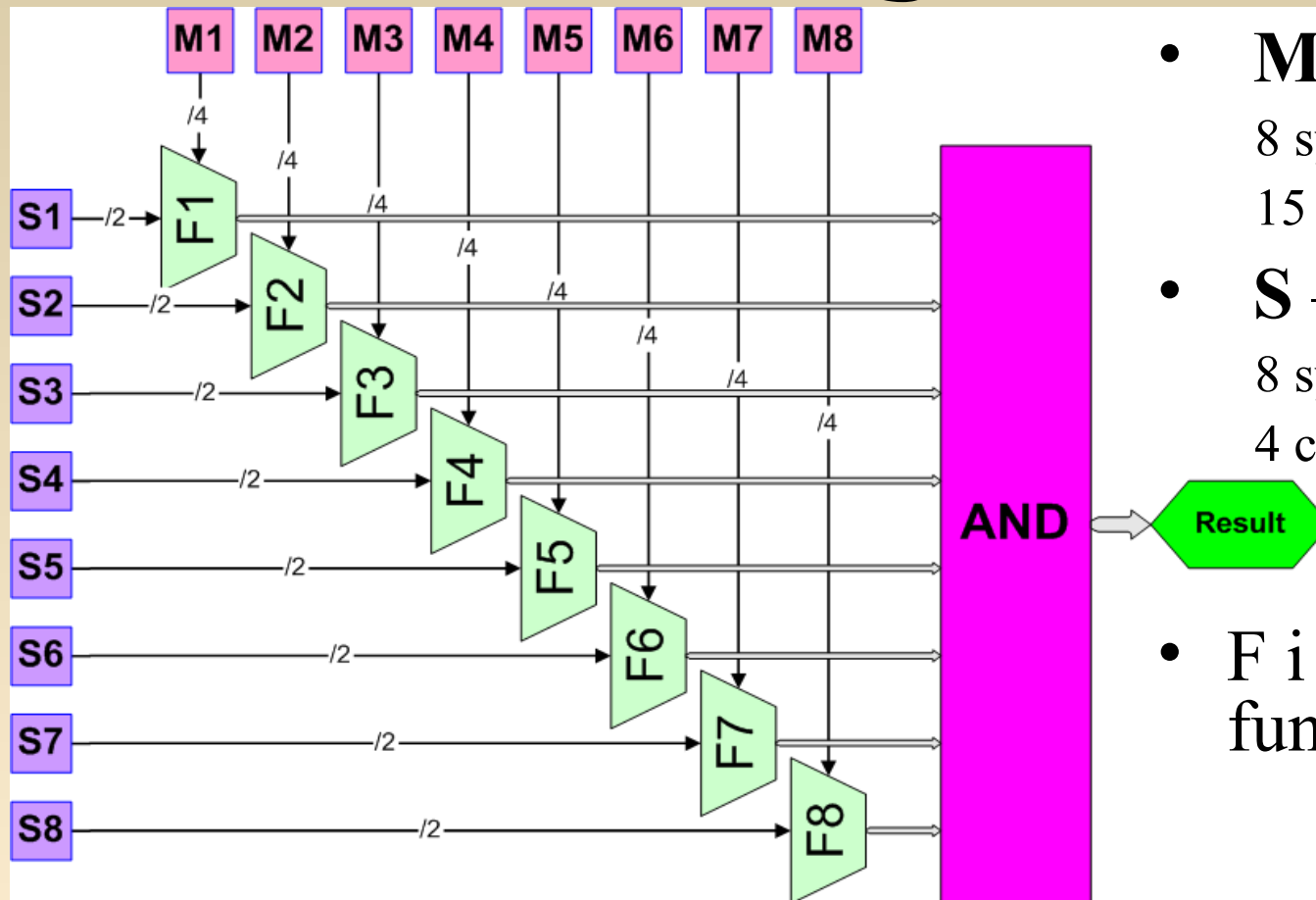
Novosibirsk State University

[mmlavr@nsu.ru](mailto:mmlavr@nsu.ru)

[mmlavrentiev@gmail.com](mailto:mmlavrentiev@gmail.com)

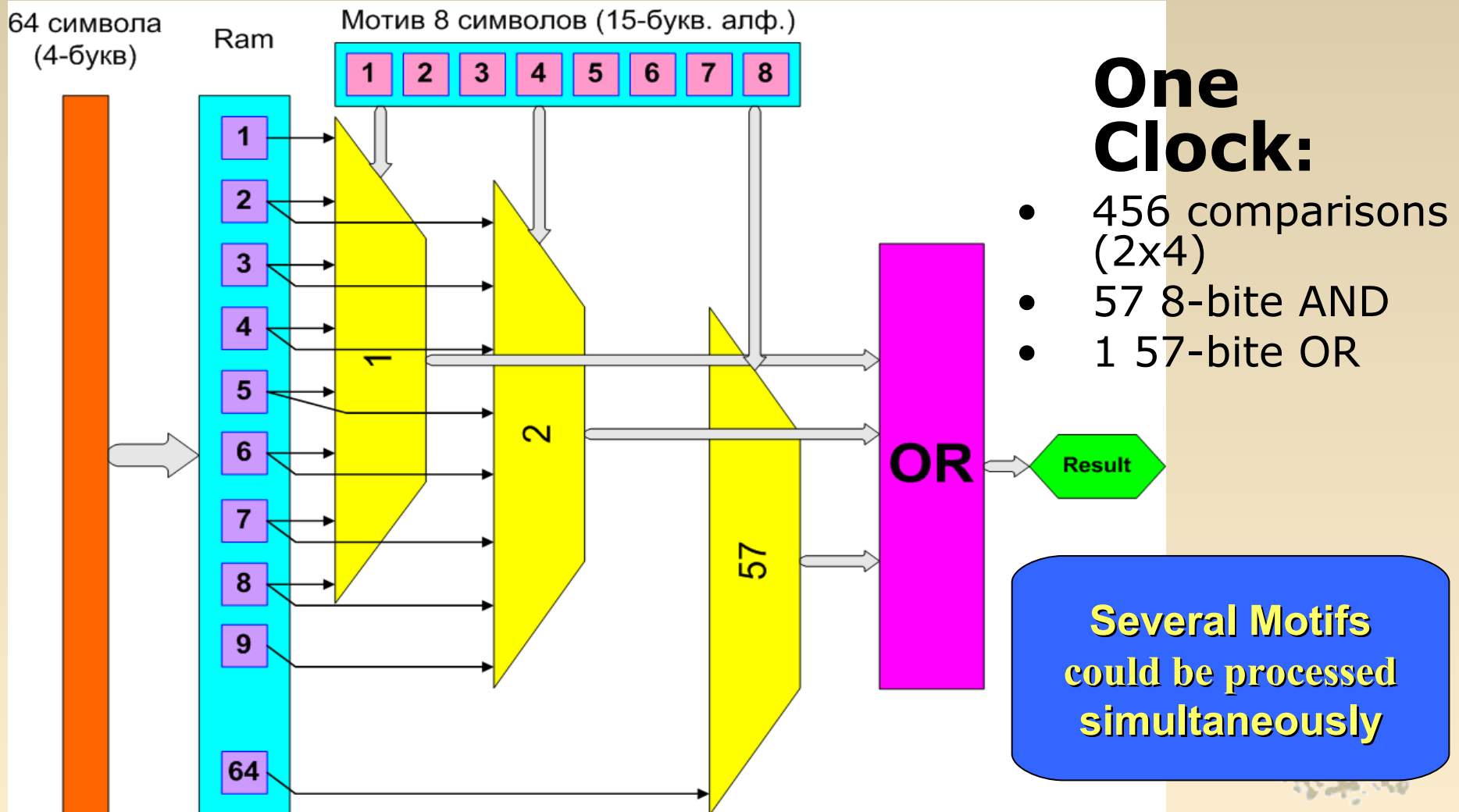


# FPGA Implementation of Motif Search Algorithm

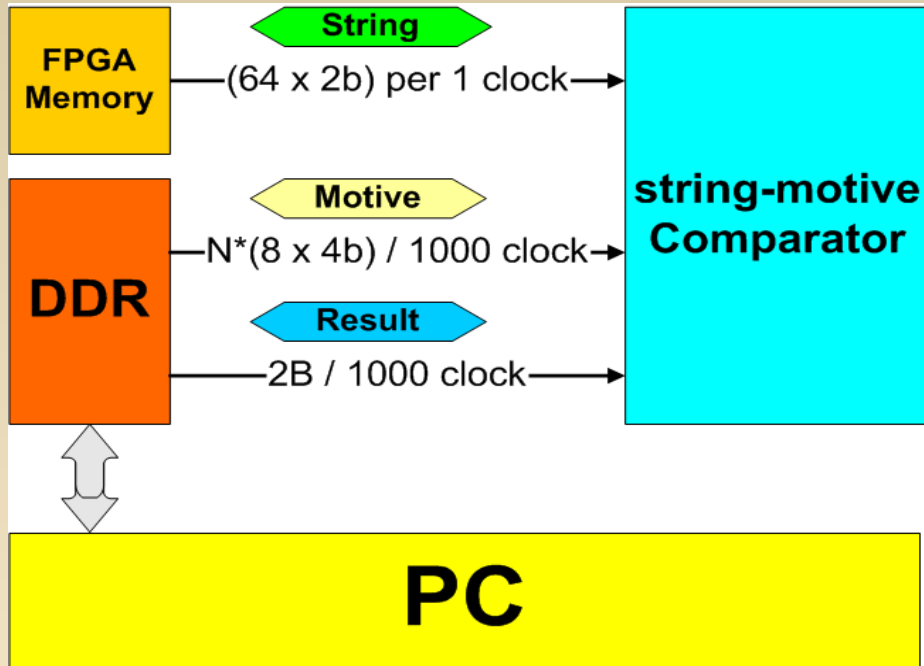


- **M** – motif  
8 symbols,  
15 char. alphabet
- **S** – input data  
8 symbols,  
4 char. alphabet
- $F_i$  – comparison function

# Motif Search in Line



# Input Data Flows



**Суммарный поток данных:  
(16002 + 4N) Байт / 1000 тактов**

*Одновременно 1 мотивов (N = 1)  
Тактовая частота 250 МГц*

$16,006 * 0,25 * 10^9$  Байт/с

**~ 3,73 ГБ/с**

**Использование FPGA позволяет не передавать строки,  
а хранить их все во внутренней памяти  
1000 строк x 64 символа x 2 бита = 16 КБ  
(объем RamB 117 – 1458 КБ)**

# Motif Search Performance at Virtex5 FPGA

|                   | <b>Logic Capacity (Slices)</b> | <b>Processing Speed</b><br>(motives per 1 clock for 64 line) | <b>Processing Time of <math>2,6 \cdot 10^9</math> motives</b><br>(64x1000) | <b>Input Data Flow (General / FPGA)</b><br><b>MB/sec.</b> | <b>Number of comparisons per 1 clock</b> |
|-------------------|--------------------------------|--|--|---|--|
| <b>XC5VLX50T</b>  | 7,200                          | 8  | 22 МИН   | 3 830 / 8   | ~ 3 650                                  |
| <b>XC5VLX110T</b> | 17,280                         | 36   | 5 МИН  | 3 850 / 35  | ~ 16 500                                 |
| <b>XC5VLX330T</b> | 51,840                         | 170  | 1 МИН  | 4 000 / 163   | ~ 77 500                                 |

**Comparison Parallelization**

**Input Data Storage  
Inner FPGA memory**