Experiences in Porting Scientific Applications to GPUs Using OpenACC

Saber Feki[1] and Ahmed Al-Jarro[2]

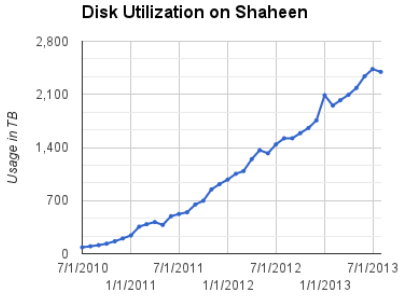
[1]KAUST Supercomputing Laboratory,
King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia

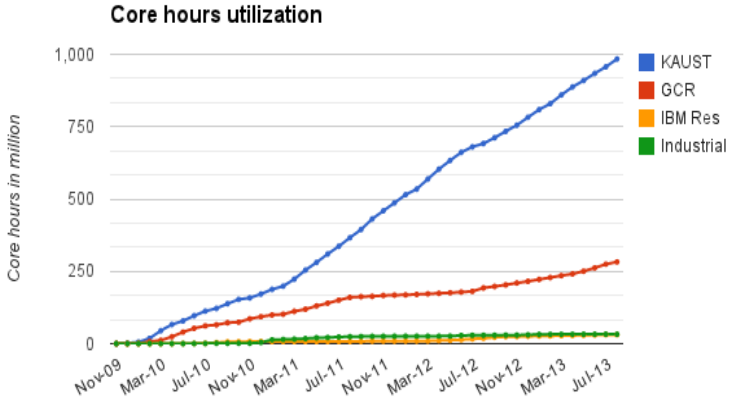
[2]Technical Computing Research,
Fujitsu Laboratories of Europe (FLE), London, United Kingdom

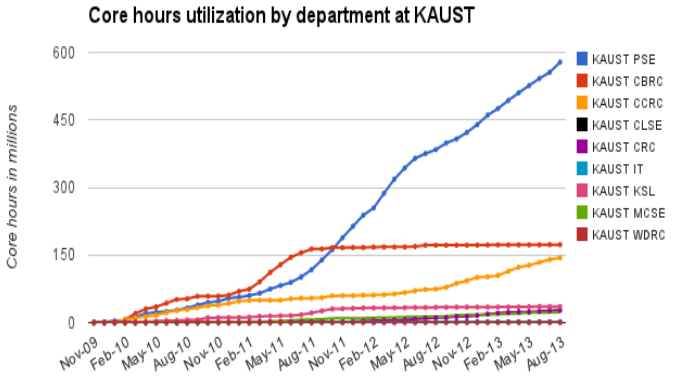
GTC 2015 – San Jose- CA – March 19th 2015

FUJITSU

KAUST Supercomputing Laboratory







KAUST Supercomputing Laboratory

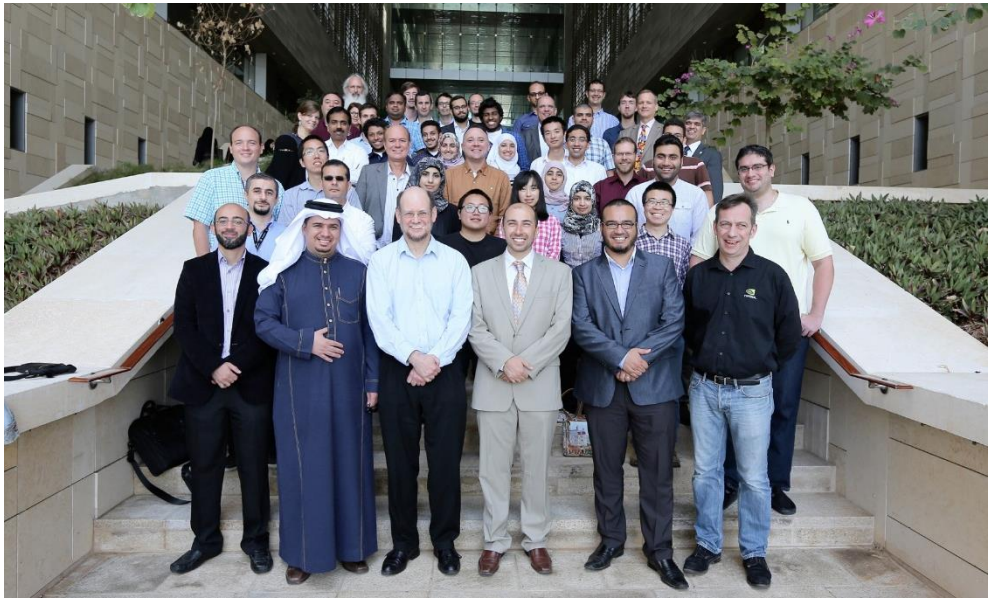
Code	Science Area	Description/Comments
WRF, WRF-Chem	Climate Modeling	Capability + Capacity
WRF, MITgcm, WAVEWATCH-III, ADCIRC	Ocean Modeling	High-speed interconnect and fast CPUs are very important for these applications.
In-house code	CFD/MHD	Capability, implicit multi grid Poisson solver
NGA, Hypre	DNS/Combustion	Capability
In-house	Computational Biology	
a) **SORD** b) **SeisSol** c) **FD-KBO** d) **SPECFEM_3D_GLOBE**	Computational Earthquake Seismology	a) A generalized 3D-finite-difference support-operator code for rupture dynamics b) a arbitrary high-order discontinous Glaerkin code for seismic wave propagation and rupture dynamics c) a highly optimized 3D-FD code for seismic wave propagation d) a 3D spectral-element code for global seismic wave propagation
In-house explicit code	CEM	Capability
Mizan, ERA , ACME - in house codes	Big Data/ Analysis of Large Graphs	Mizan (graph processing), ERA (string indexing), ACME (string mining) - in house codes. Fast communication network very important. A lot of interdependencies.
In-house and commercial FE codes + open source MD codes	Multiscale CSD	Capability + Capacity
VASP, LAMMPS, Gaussian, Wein2k, QE	Computational Chemistry	Our codes do not need high-speed interconnect. We need lots of fast nodes

KAUST Supercomputing Laboratory

The basics		Notes & detail
What?	Manufacturer: Cray	Model: XC40
Where, when?	Delivery early March 2015 to the KAUST campus in Thuwal, Saudi Arabia	
Speed	7.2 petaflops peak theoretical performance	Will achieve peak sustained LINPACK of over 5 petaflops
Power	Up to 2.8 MW	Water cooled
Size and weight	Nearly 100 metric tons	36 compute cabinets, plus disk, blowers, management, etc.
Main components	Processor type: Intel Haswell	2 CPU sockets per node, 16 processor cores per CPU. 2.3 Ghz base frequency, turbo to 3.6 Ghz
	6,192 nodes	198,144 processor cores
	128 GB of memory per node	Over 790 TB total memory
Network	Cray Aries interconnect with Dragonfly topology	MPI latency of .35 us (min) to 2.5 (max)
Storage	DataWarp burst buffer	Solid State Devices (SSD) fast data cache. Over 1 TB/s bandwidth, delivery September 2015
	Sonexion Lustre appliance	17.6 petabytes usable, 23 petabytes raw. Over 500 GB/s bandwidth
	Tiered Adaptive Storage (TAS)	Hierarchical storage with 200 TB disk cache and 20 PB of tape storage, using a Spectra Logic tape library and IBM tapes & drives.
Also	Urika-GD appliance	For graph analytics and big data.

KAUST Supercomputing Laboratory

- **Upgrade plan with accelerators due to power constraints**
- Preparing our users for the upgrade:



Intel Parallel Computing Centers

NVIDIA CUDA RESEARCH CENTER

Second Workshop on Accelerating Scientific Applications using GPUs

- **First KAUST Hackathon – February 18-19, 2015**

Outline

- **Experience in porting and auto-tuning Seismic Imaging kernels**
- Experience in porting a FSI code to GPUs using OpenACC
- Experience with a CEM code ported to multiple GPUs using MPI + OpenACC

Agenda

- Motivation
- Seismic imaging application
- Auto-tuning methodology
- Experimental results
- Conclusion and Future work

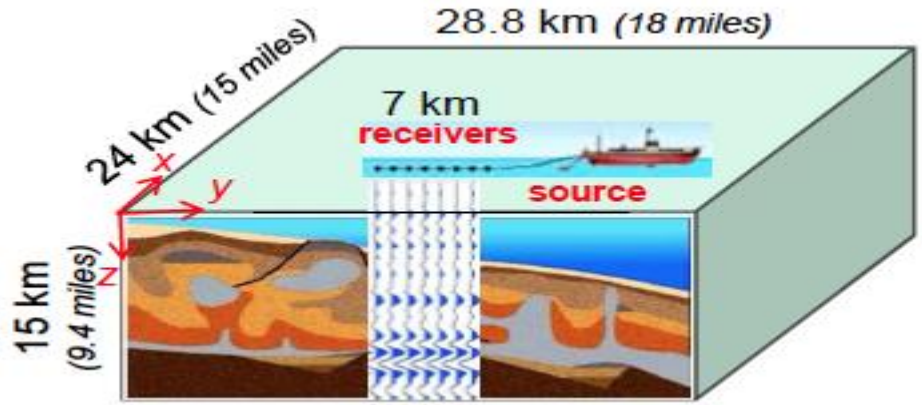
Motivation

- Increased **complexity** and **diversity** of HPC systems, programming languages and applications
- Typically, codes lifetime is longer than the system's
- Getting the best performance out of such systems is becoming a challenging task
- A tedious, resource and time consuming procedure that **requires expertise** in many disciplines

Automatic Performance Tuning

Application: Seismic Imaging

- The boat has 10 cables With 556 receivers per cable, total number of receivers for 10 cables: 5,560
- Total number of shots: 82,557
- Total number of records (“traces”) = shots x receivers = 82,557x5,560 = 459,016,920
- Each record length is 14.336 seconds with the 4.0ms sampling rate. Yield 3,584 samples per trace.
- Total data size = 459,016,920 x 3584 x 4 byte = ~ 6.6TB .
- Useful spectra are from 3.5Hz to about 45-50Hz, with depth up to 15km

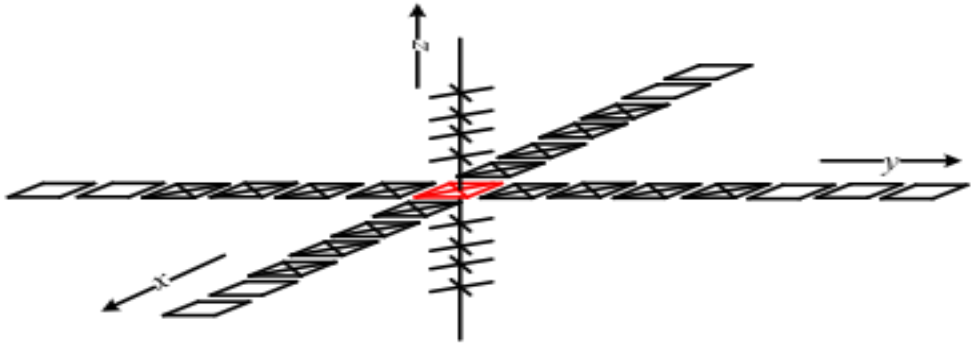


Two information here : data from source and data recorded from receivers

Computational Kernel

- Solve the acoustic wave equation (Isotropic case)

$$\frac{1}{c^2}\frac{\partial^2 \mathbf{u}}{\partial t^2} = \frac{\partial^2 \mathbf{u}}{\partial x^2} + \frac{\partial^2 \mathbf{u}}{\partial y^2} + \frac{\partial^2 \mathbf{u}}{\partial z^2}$$



- Finite difference scheme, 2nd order in time, 8th order in space

Seismic Imaging Kernel

```
for (i=0; i<nt; i+=2) { // time loop
  for(x=4; x< p.ip-4; x++) {
    for(y=4; y< p.jp-4; y++) {
      for(z=4; z< p.kp-4; z++) {
      lap=coef[0]*V(x,y,z)
          +coef[1]*(V(x+1,y  ,z  )+V(x-1,y  ,z  ))
          +coef[1]*(V(x  ,y+1,z  )+V(x  ,y-1,z  ))
          +coef[1]*(V(x  ,y  ,z+1)+V(x  ,y  ,z-1))
          +coef[2]*(V(x+2,y  ,z  )+V(x-2,y  ,z  ))
          +coef[2]*(V(x  ,y+2,z  )+V(x  ,y-2,z  ))
          +coef[2]*(V(x  ,y  ,z+2)+V(x  ,y  ,z-2))
          +coef[3]*(V(x+3,y  ,z  )+V(x-3,y  ,z  ))
          +coef[3]*(V(x  ,y+3,z  )+V(x  ,y-3,z  ))
          +coef[3]*(V(x  ,y  ,z+3)+V(x  ,y  ,z-3))
          +coef[4]*(V(x+4,y  ,z  )+V(x-4,y  ,z  ))
          +coef[4]*(V(x  ,y+4,z  )+V(x  ,y-4,z  ))
          +coef[4]*(V(x  ,y  ,z+4)+V(x  ,y  ,z-4));

        U(x,y,z) = 2.*V(x,y,z) - U(x,y,z) + ROC2(x,y,z)*lap;
        if( (x==ixs) && (y==iys) && (z==izs) ){
          U(ixs,iys,izs) =  U(ixs,iys,izs) + source[i];
        }
      }
    }
  }
```

OpenACC Implementation

```
#pragma acc data copyin(U2[0:p.domain_size],U3[0:p.domain_size], source[0:nt], coef[0:five]) copy(U1[0:p.domain_size])
  for (i=0; i<nt; i+=2) { // time loop
    #pragma acc parallel loop num_gangs(4) vector_length(32)
    for(x=4; x< p.ip-4; x++) {
      #pragma acc loop
      for(y=4; y< p.jp-4; y++) {
        #pragma acc loop
        for(z=4; z< p.kp-4; z++) {
        lap=coef[0]*V(x,y,z)
            +coef[1]*(V(x+1,y  ,z  )+V(x-1,y  ,z  ))
            +coef[1]*(V(x  ,y+1,z  )+V(x  ,y-1,z  ))
            +coef[1]*(V(x  ,y  ,z+1)+V(x  ,y  ,z-1))
            +coef[2]*(V(x+2,y  ,z  )+V(x-2,y  ,z  ))
            +coef[2]*(V(x  ,y+2,z  )+V(x  ,y-2,z  ))
            +coef[2]*(V(x  ,y  ,z+2)+V(x  ,y  ,z-2))
            +coef[3]*(V(x+3,y  ,z  )+V(x-3,y  ,z  ))
            +coef[3]*(V(x  ,y+3,z  )+V(x  ,y-3,z  ))
            +coef[3]*(V(x  ,y  ,z+3)+V(x  ,y  ,z-3))
            +coef[4]*(V(x+4,y  ,z  )+V(x-4,y  ,z  ))
            +coef[4]*(V(x  ,y+4,z  )+V(x  ,y-4,z  ))
            +coef[4]*(V(x  ,y  ,z+4)+V(x  ,y  ,z-4));

          U(x,y,z) = 2.*V(x,y,z) - U(x,y,z) + ROC2(x,y,z)*lap;
          if( (x==ixs) && (y==iys) && (z==izs) ){
            U(ixs,iys,izs) =  U(ixs,iys,izs) + source[i];
          }
        }
      }
    }
```

Directive syntax

- Fortran

!$acc directive [clause [,] clause] …]

… often paired with a matching end directive

!$acc end directive

- C

#pragma acc directive [clause [,] clause] …]

Often followed by a structured code block

Gang and Vector Clauses

- OpenACC execution model has up to three level of parallelism: **gang, worker** and **vector**
- For GPUs, the mapping is implementation-dependent. Some possibilities:
  - gang==block, worker==warp, and vector==threads of a warp
  - omit “worker” and just have **gang==block, vector==threads** of a block

- Typically set by the compiler using some heuristics.
- Optionally, developer can specify the mapping of loops to gang and vector for further tuning.
  - Auto-tuning opportunity

Automatic Performance Tuning Model

Input code annotated with OpenACC

```
#pragma acc kernels
#pragma acc loop independent
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

Accelerator Specification

Automatic code generator

```
#pragma acc kernels
#pragma acc loop independent gang(a),vector(b)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent vector(d)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

```
#pragma acc kernels
#pragma acc loop independent
for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(a),vector(b)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent gang(c),vector(d)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

```
#pragma acc kernels
#pragma acc loop independent gang(a)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(b),vector(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent vector(d)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

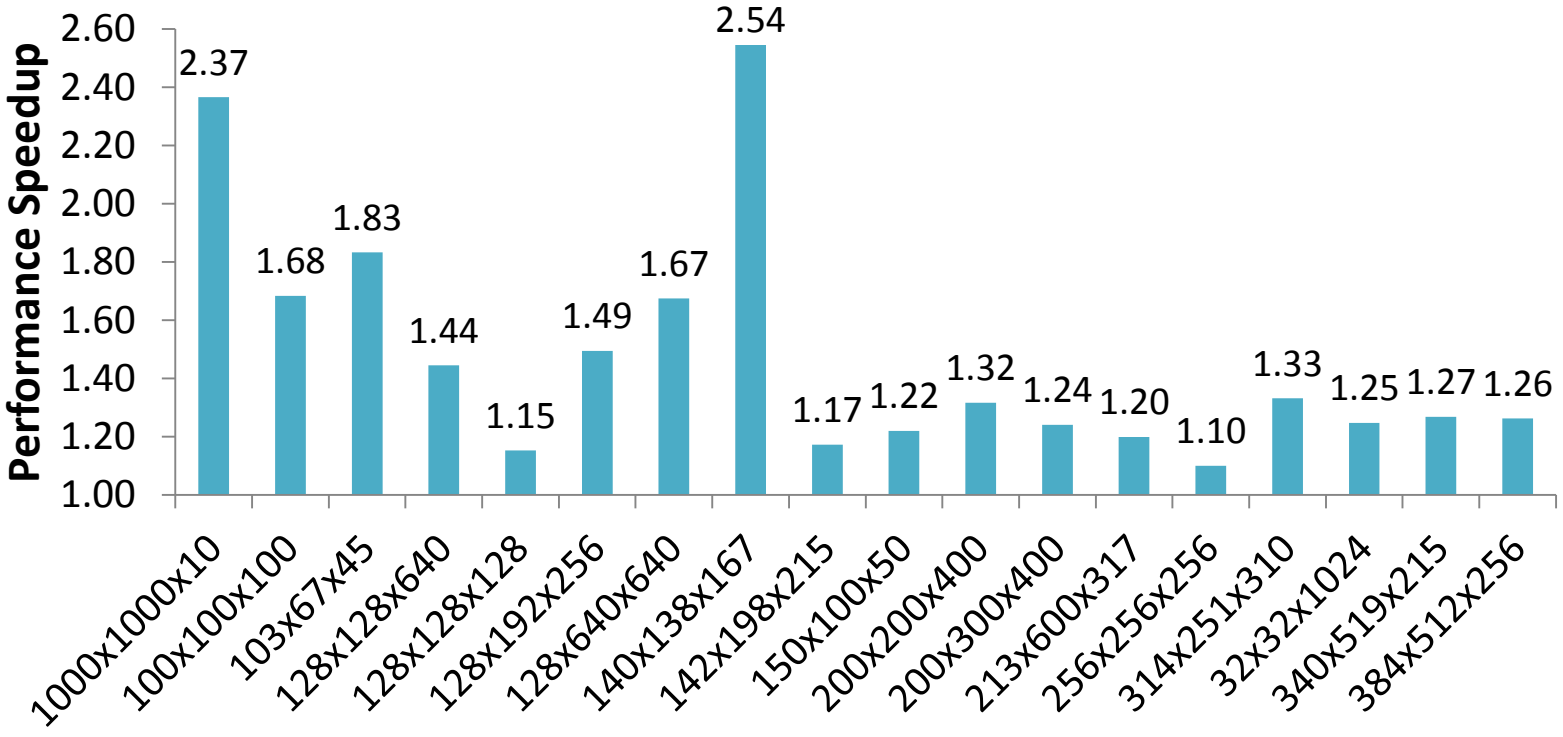
```
#pragma acc kernels
#pragma acc loop independent gang(a),vector(b)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent vector(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent gang(d),vector(e)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

Runtime evaluation and selection ⟺ **Database**

Experimental Results (I)

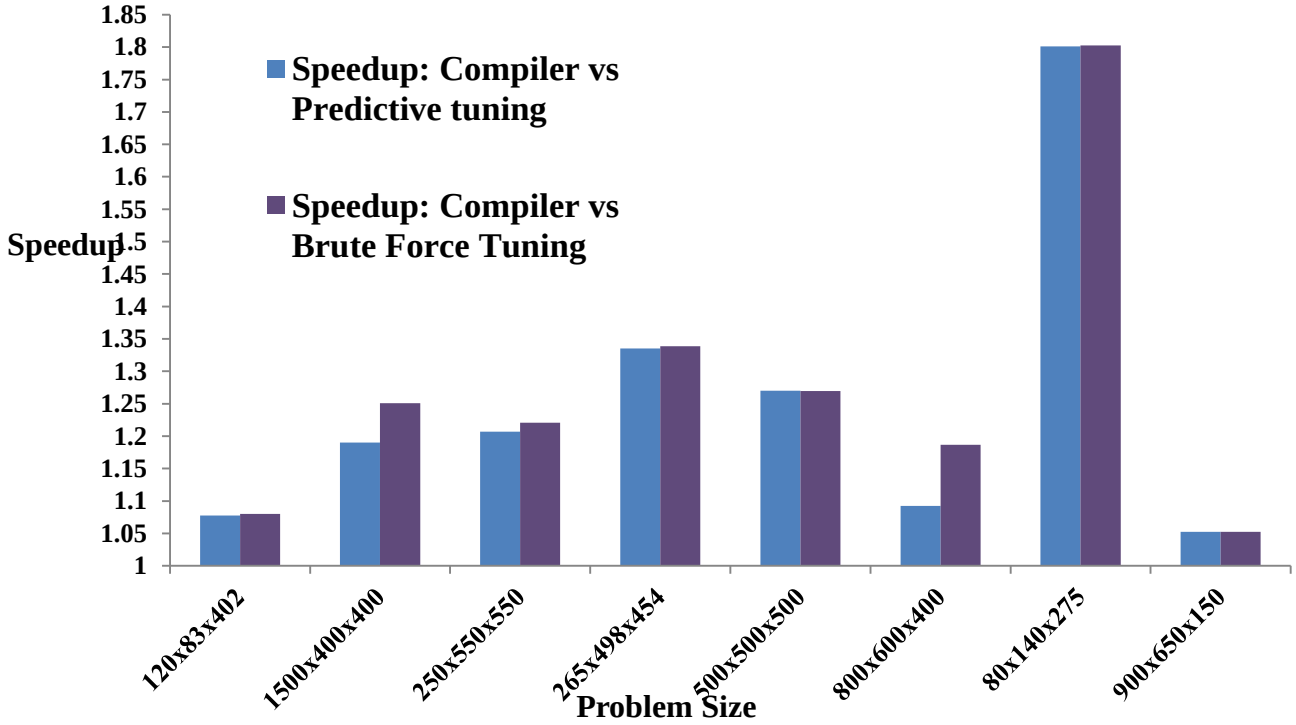
- Seismic kernel solving the acoustic wave equation
- Finite difference scheme, 2^{nd} order in time, 8^{th} order in space
- Performance reported on NVIDIA K20c GPUs
- NVIDIA recommends the vector size to be multiple of warp size (32) on thus we explored the values [32,64,96,…,1024]
- Gang values tested on increments of 2 starting from 2 up till 1024
- Applied brute force on 18 problem sizes to populate a history database.
- Applied the historic learning approach to 8 different problem sizes and compared the performance with the optimal we can get while using a brute force search

Experimental Results (II)



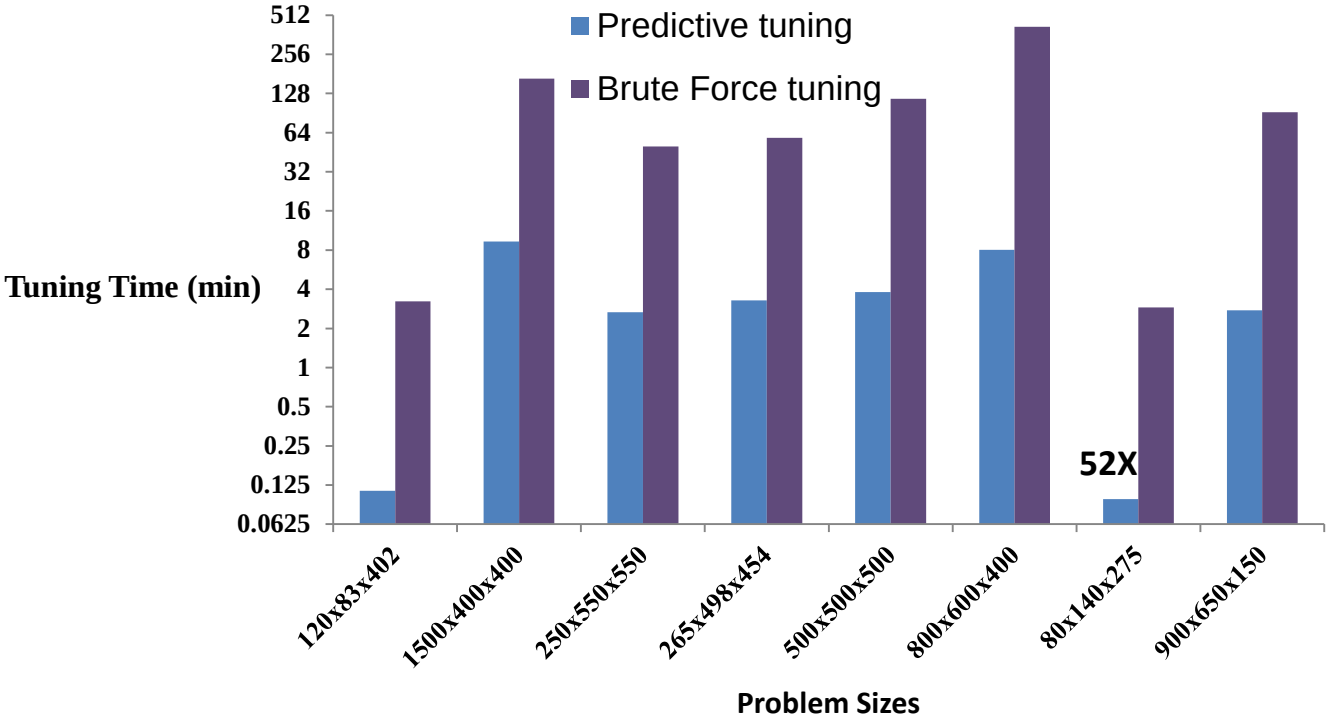
Brute force tuning of 18 problem sizes to generate the history data

Experimental Results (III)



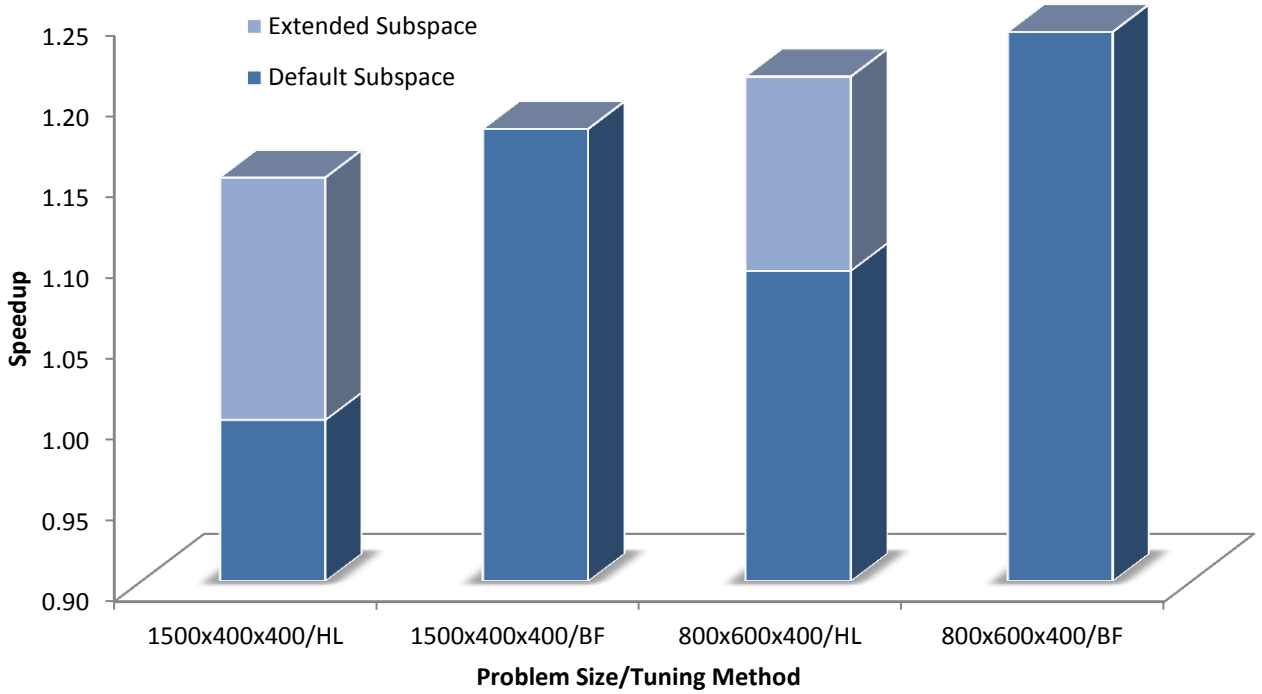
Performance speedup using either a brute force search or predictive tuning against the compiler tuned code

Experimental Results (IV)



Tuning time using either a brute force search or predictive tuning methods

Experimental Results (V)



Performance speedup improvement by adapting the size of the search subspace

Adaptation to the Underlying Hardware

Accelerator Type	Gang	Vector	Speedup
NVIDIA Fermi	256	144	1.29X
NVIDIA K20c	884	96	1.43X
NVIDIA K40	784	128	1.41X
PGI compiler	92	256	-

Conclusion

- OpenACC: a promising parallel programming model for productive programming on accelerators.

- Automatic performance tuning model based on parameters prediction from historic tuning data achieve significantly better performance.

- Auto-tuning is well known technique towards **performance portability**

Future Work

- Experiments and validation of the auto-tuning model:
  - on different architectures (Phi, AMD GPUs)
  - with different scientific applications

- Exploring other search algorithms: Genetic Algorithms, Nedler Mead, Random Search …

- Targeting tunable parameters in OpenACC 2.0 features such as the tile clause.

Outline

- Experience in porting and auto-tuning Seismic Imaging kernels
- **Experience in porting a FSI code to accelerators using OpenACC**
- Experience with a CEM code ported GPUs using different OpenACC compilers

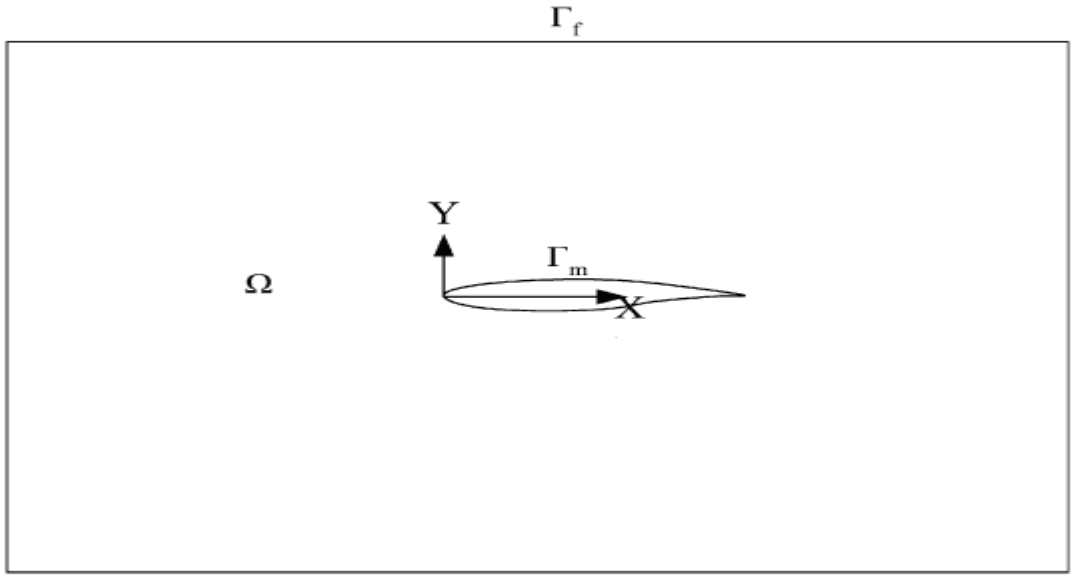
Mesh Motion Scheme

Given: *g*, the prescribed displacements at the moving boundary, find the mesh displacement field $u : \Omega \rightarrow \Re^{n_{sd}}$ such that

$$\nabla \cdot \tau_m \nabla \boldsymbol{u} = 0 \qquad \text{in } \Omega$$

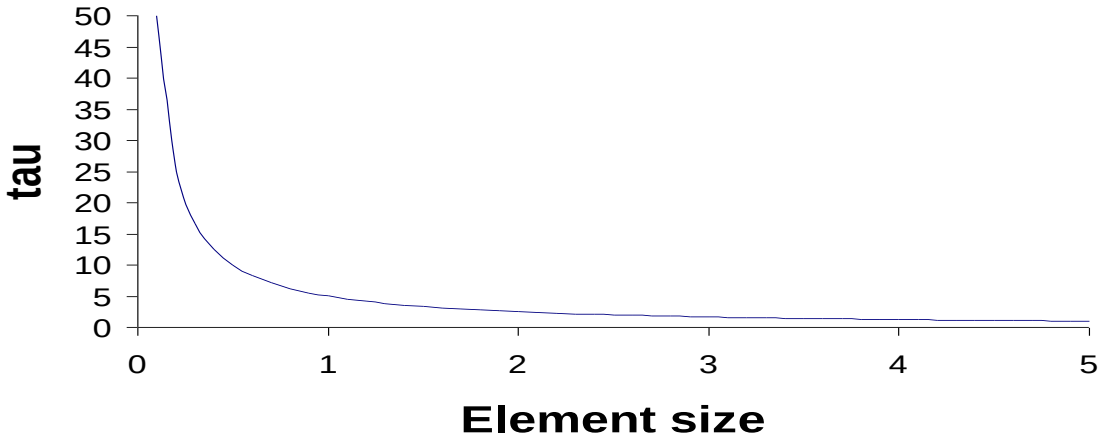
$$\boldsymbol{u} = \boldsymbol{g} \qquad \text{on } \Gamma_m$$

$$\boldsymbol{u} = \boldsymbol{0} \qquad \text{on } \Gamma_f$$



Definition of Mesh Control Parameter

$$\tau_m = \frac{\Delta_{\max} - \Delta_{\min}}{\Delta^e}$$



The standard weak form:

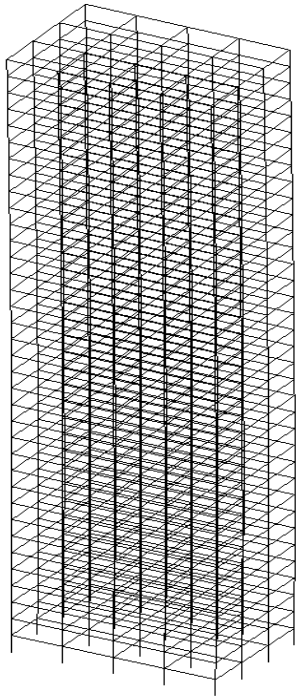
$$(\nabla \boldsymbol{w}, \tau_{\mathrm{m}} \nabla \boldsymbol{u}) = 0$$

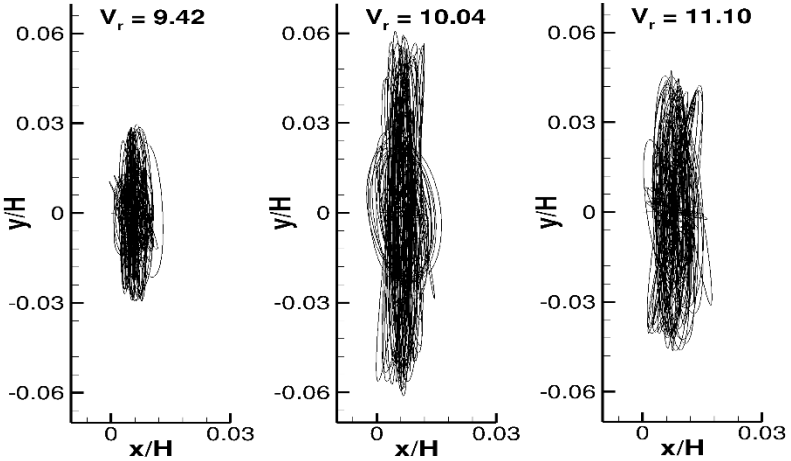
$$Kd = F$$

$$\boldsymbol{K} = \mathop{\mathbf{A}}_{\boldsymbol{e}=1}^{\boldsymbol{n}} \boldsymbol{k}^{\boldsymbol{e}} \qquad \boldsymbol{F} = \mathop{\mathbf{A}}_{\boldsymbol{e}=1}^{\boldsymbol{n}} \boldsymbol{f}^{\boldsymbol{e}}$$

$$k^e = \int_{\Omega^e} \left(\nabla N^T \tau_m \nabla N\right) d\Omega^e \qquad f^e = \int_{\Omega^e} N^T f(\mathrm{x}) d\Omega^e$$

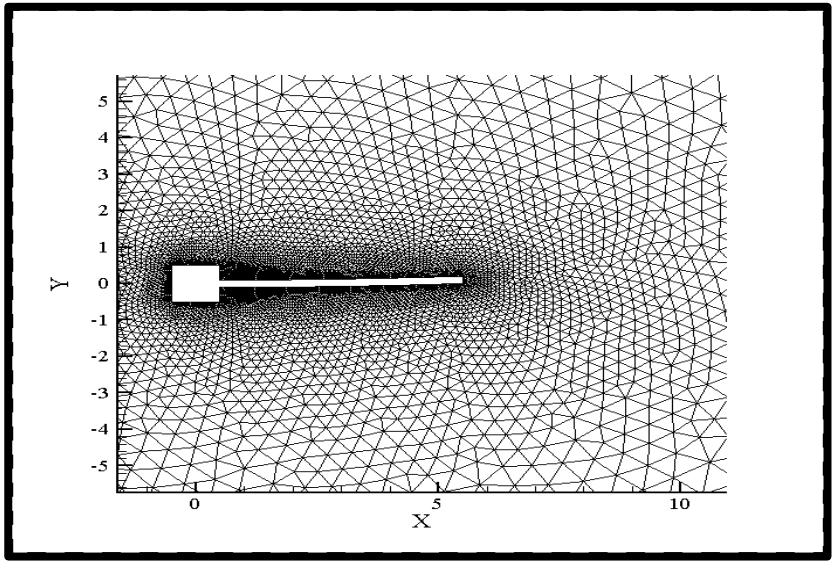
Wind Induced Vibration of CAARC Building





Yue Zhang, **Wagdi G. Habashi**, Rooh A. Khurram, Wind-induced vibration of high-rise buildings using unsteady CFD and modal analysis, submitted to Journal of Wind Engineering and Industrial Applications, 2014

Deformation of Mesh



References:

M. Fossati, R.A.Khurram, W. G. Habashi, An arbitrary Lagrangian-Eulerian mesh movement scheme for long-term in-flight ice accretion, International Journal of Numerical Methods in Fluids, Volume 68, Issue 8, Pages: 958–976, 2012

Arif Masud, Manish Bhanabhagvanwala, Rooh A. Khurram, An adaptive mesh rezoning scheme for moving boundary flows and fluid–structure interaction, Computers & Fluids 36 (2007) 77–91

A. Masud and T.J.R. Hughes, A space-time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems. Computer Methods in Applied Mechanics and Engineering , vol. 146, 91-126,1997

Mesh Motion Code

- **Read mesh file**
  - Mesh coordinates
  - Element types
  - Element connectivity
- **Read boundary conditions**
  - Fixed nodes (displacement = 0)
  - Moving nodes (specified displacement/FSI)
- **Generate element stiffness matrices**
  - Calculation of element matrices
  - Element based stiffness
  - Imposing displacement BCs
- **Linear solve**
  - CG algorithm
- **Write mesh file, if needed**
- **Repeat for next time step**

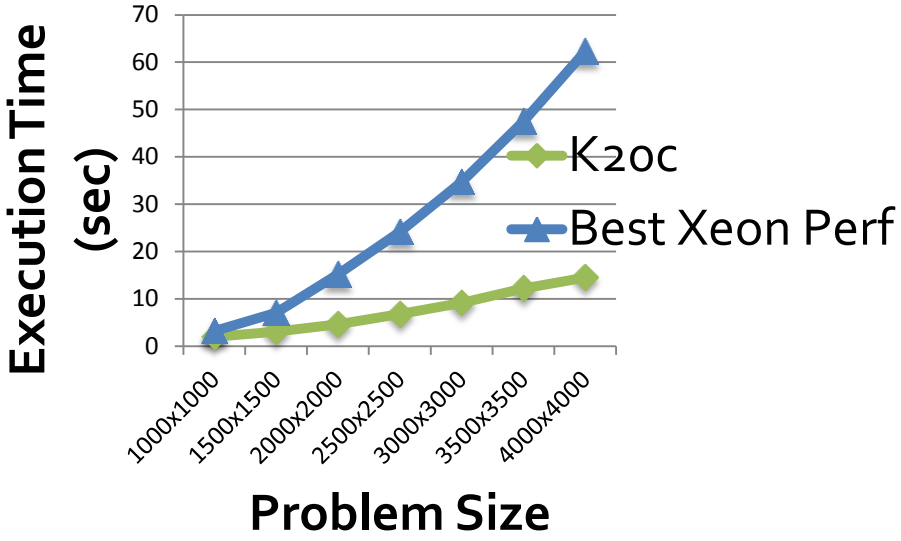
Code profile

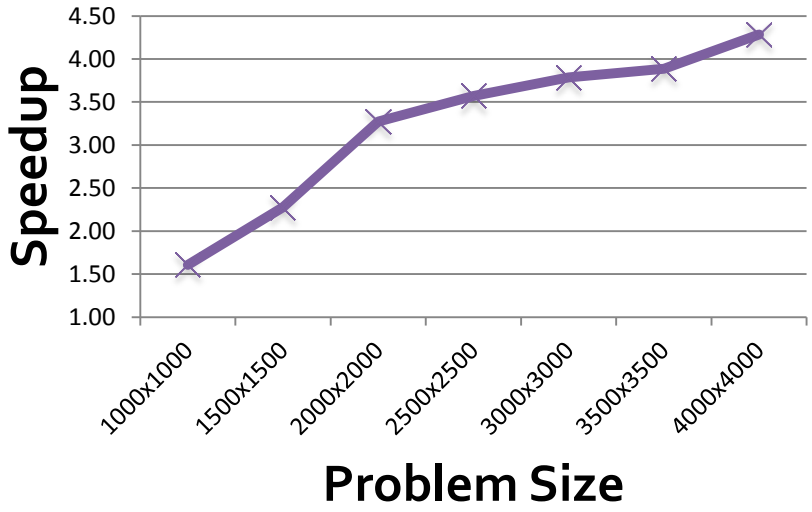
time	seconds	seconds	calls	s/call	s/call	name
94.75	36.39	36.39	2	18.20	18.20	**solvemmove_**
3.65	37.79	1.40	1	1.40	1.40	**stiffmmove_**
0.70	38.06	0.27	1	0.27	0.27	bca_
0.29	38.17	0.11	1	0.11	0.11	tecplot_
0.21	38.25	0.08	1	0.08	0.08	get_input_
0.21	38.33	0.08	1	0.08	38.41	getstarted_
0.10	38.37	0.04	1	0.04	0.04	updatemmove_
0.05	38.39	0.02	1	0.02	0.02	checkareaale_
0.05	38.41	0.02	1	0.02	0.02	readbcmove_
0.00	38.41	0.00	1	0.00	0.00	MAIN__
0.00	38.41	0.00	1	0.00	38.12	alemmove_
0.00	38.41	0.00	1	0.00	0.00	initial_
0.00	38.41	0.00	1	0.00	0.00	mem_alloc_

Test Bed Specifications

	Intel Xeon Sandy Bridge	NVIDIA Kepler GPU K20c
Sockets	2	1
Cores	16	2496
Threads	16	2496
Frequency	2.6 GHz	706 MHz
RAM	64 GB DDR3	6 GB GDDR5
Peak Performance	332.8 Gflop/s	**1170** Gflop/s
Power	115 W	225 W

Solver Performance





- The larger the problem size, the better speedup obtained
- The bottleneck in 3D case will be the limited memory on the accelerator (which tends to improve in the next generations; e.g. 12 GB in K40)

* Speedup compared to the best OpenMP Sandy Bridge performance

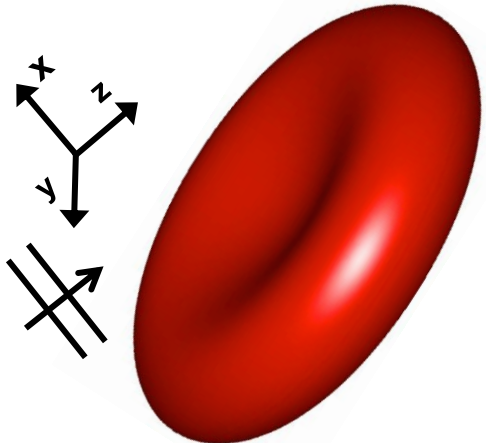
Concluding Remarks

- It took 2-3 weeks for 3 researchers to port, modify, optimize, and test the code
- Speedup is a strong function of problem size
- Speedup of 2-4 is achieved with modest effort
- Accelerators provide energy efficient solutions

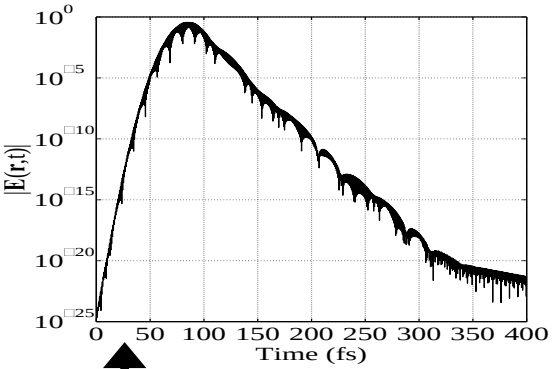
Outline

- Experience in porting and auto-tuning Seismic Imaging kernels
- Experience in porting a FSI code to GPUs using OpenACC
- **Experience with a CEM code ported to multiple GPUs using MPI + OpenACC**

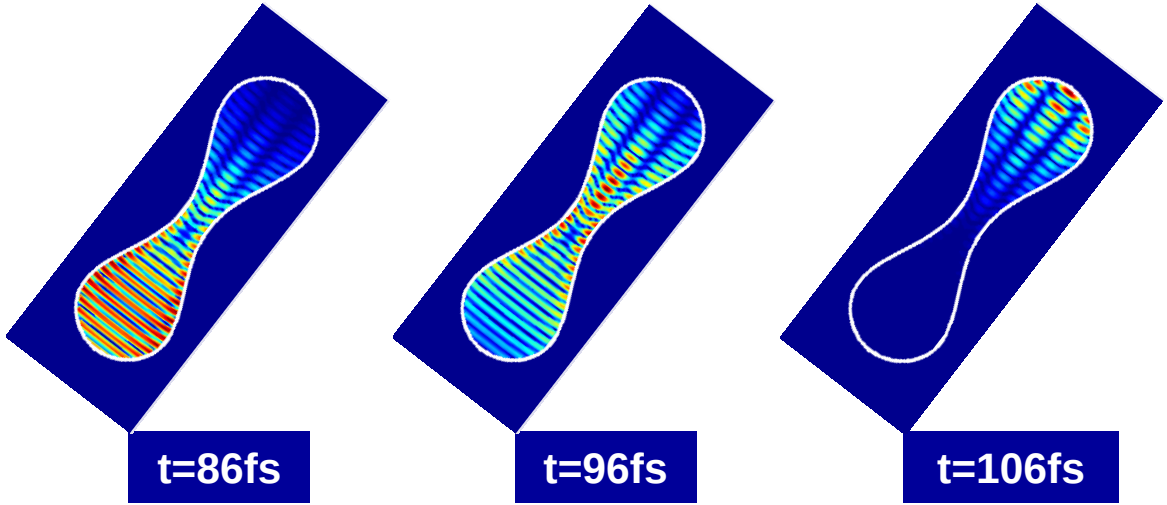
Light Interaction with Red Blood Cells



- **Used in biomedical applications: devices utilizing lasers for disease diagnosis**
- **Can provide essential information for blood related diseases**



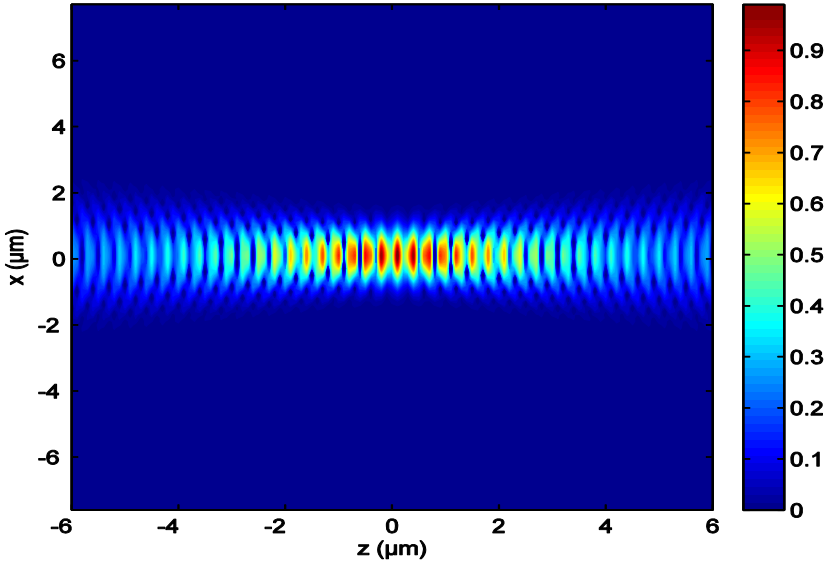
The amplitude of the transient electric field induced at the centre of the RBC

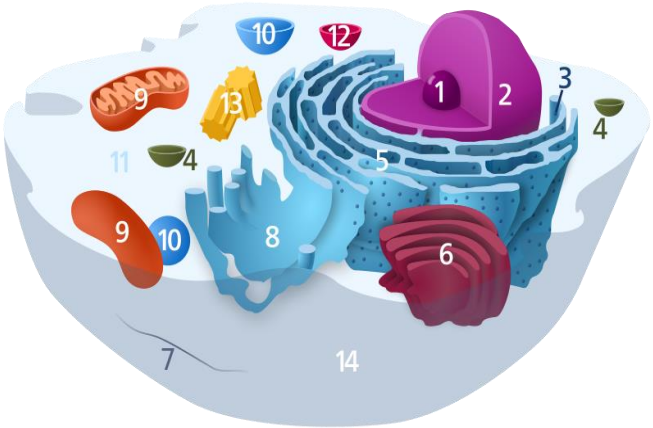


Normalised intensity of the electric field distribution at three moments in time

Light Interaction with Tissue Cells (Organelle)

- **Used in microscopy and 3D imaging**
  - **Two photon florescence microscopy for 3D imaging analysis**



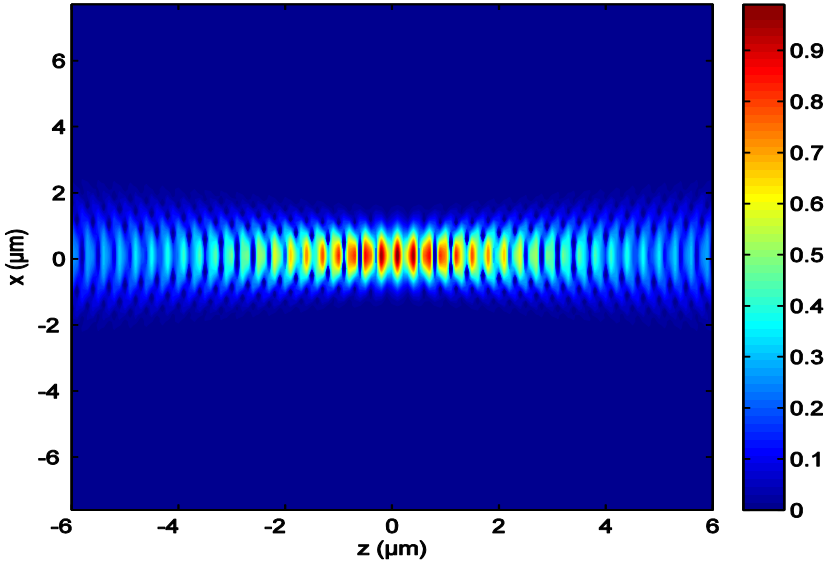


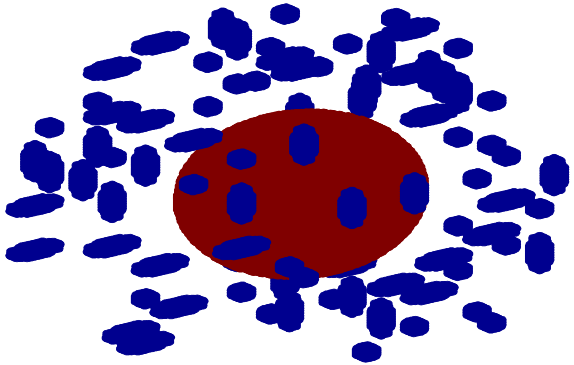
2) Nucleus, 9) Mitochondria, 14) membrane

Incident electric field profile (laser source) on Organelle with heterogeneous nucleus that is placed in cytoplasm background

Light Interaction with Tissue Cells (Organelle)

- **Used in microscopy and 3D imaging**
  - **Two photon florescence microscopy for 3D imaging analysis**



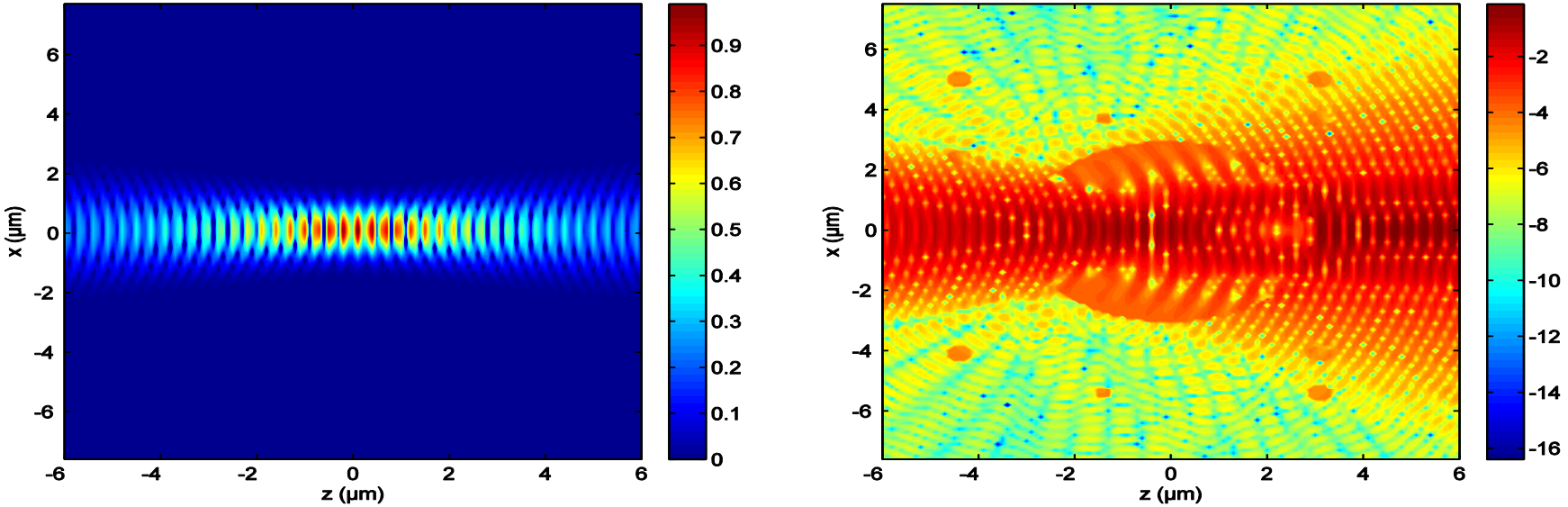


Nucleus, **Mitochondria**

Incident electric field profile (laser source) on Organelle with heterogeneous nucleus that is placed in cytoplasm background

Light Interaction with Tissue Cells (Organelle)

- **Used in microscopy and 3D imaging**
  - **Two photon florescence microscopy for 3D imaging analysis**



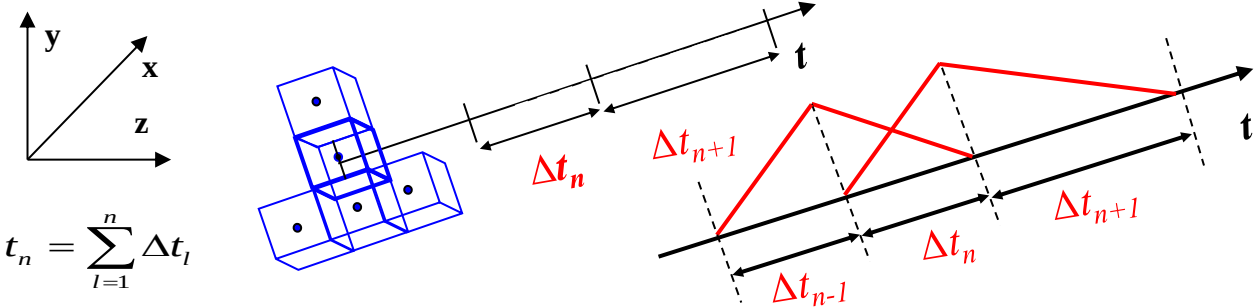
Incident electric field profile (laser source) on Organelle with heterogeneous nucleus that is placed in cytoplasm background

MOT-TDVIE Solver

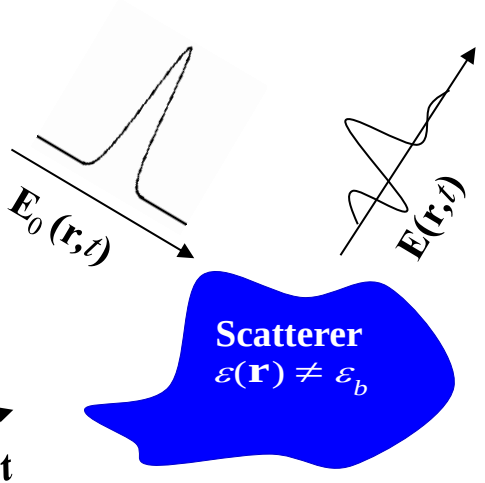
For an arbitrarily shaped 3D dielectric object residing in a homogeneous background, the total electric field satisfies

$$\mathbf{E}(\mathbf{r},t)=\mathbf{E}_0(\mathbf{r},t)+\left[\nabla\nabla\cdot-\frac{\partial_t^2}{c_b^2}\right]\int_V d\mathbf{r}'\,\frac{\left[\varepsilon(\mathbf{r}')-\varepsilon_b\right]\mathbf{E}(\mathbf{r}',t')}{4\pi\varepsilon_b\left|\mathbf{r}-\mathbf{r}'\right|}$$

$$\mathbf{E}(\mathbf{r}_i,t_n)=\mathbf{E}_0\left(\mathbf{r}_i,t_n\right)+\sum_{i=1}^{N_e}\sum_{j=1}^{N_t}\mathbf{E}(\mathbf{r}_i,t_j)\,p_i(\mathbf{r})T_j(t)$$



$$t_n=\sum_{l=1}^{n}\Delta t_l$$



$$\varepsilon(\mathbf{r})=\varepsilon_b$$

A. Al-Jarro, *et al.,* "Explicit solution of the time domain volume integral equation using a predictor-corrector scheme", IEEE Trans. on Antennas and Propag., vol. 60, no. 11, 5203-5214, 2012.

MOT-TDVIE Solver

An explicit predictor-corrector solver is employed

Step 1 : **Predictor** – f is evaluated from up to previous time step

$$\mathbf{E}^{P}(\mathbf{r}_i,t_n)=\bar{\mathbf{M}}^{-1}\cdot\{Q[2\mathbf{E}(\mathbf{r}_i,t_{n-1})-\mathbf{E}(\mathbf{r}_i,t_{n-2})]+\mathbf{E}_0(\mathbf{r}_i,t_n)+\mathbf{f}(\mathbf{r}_i,t_n)\}$$

Step 2 : **Corrector** – f is evaluated from an averaged contribution from previous and current time steps for increased stability

$$\mathbf{E}^{C}(\mathbf{r}_i,t_n)=\bar{\mathbf{M}}^{-1}\cdot\{Q[2\mathbf{E}(\mathbf{r}_i,t_{n-1})-\mathbf{E}(\mathbf{r}_i,t_{n-2})]+\mathbf{E}_0(\mathbf{r}_i,t_n)+0.5\left[\mathbf{f}(\mathbf{r}_i,t_n)+\mathbf{f}(\mathbf{r}_i,t_{n-1})\right]\}$$

$$\mathbf{F}(\mathbf{r}_i,t_n)=\sum_{i\neq j}^{N_e}\Delta d^3\,p_i(\mathbf{r})\frac{1}{R_{ij}}\mathbf{E}(\mathbf{r}_j,t_n-R_{ij}/c_b)$$

$$\mathbf{f}(\mathbf{r}_i,t_n)=\left(\tilde{\nabla}\tilde{\nabla}\cdot-\frac{\tilde{\partial}_t^2}{c_b^2}\right)\mathbf{F}(\mathbf{r}_i,t_n)$$

Computational Cost:

$$O\left(N_e^{\,2}N_t\right)$$

Parallelization and/or Hardware Acceleration

$$\bar{\mathbf{M}}=[1+Q]\bar{\mathbf{I}}-\bar{\mathbf{D}}(\mathbf{r}_i)$$
$$Q=S(\mathbf{r}_i)/c_b^2\Delta t^2$$

$$\bar{\mathbf{D}}(\mathbf{r}_i)=p_i(\mathbf{r})\int_{V_i}\nabla'\nabla'\frac{dv'}{|\mathbf{r}_i-\mathbf{r}'|},\qquad S(\mathbf{r}_i)=p_i(\mathbf{r})\int_{V_i}\frac{dv'}{|\mathbf{r}_i-\mathbf{r}'|}$$

A. Al-Jarro, *et al.,* "Explicit solution of the time domain volume integral equation using a predictor-corrector scheme", IEEE Trans. on Antennas and Propag., vol. 60, no. 11, 5203-5214, 2012.

GPGPU Acceleration: OpenACC

CAPS

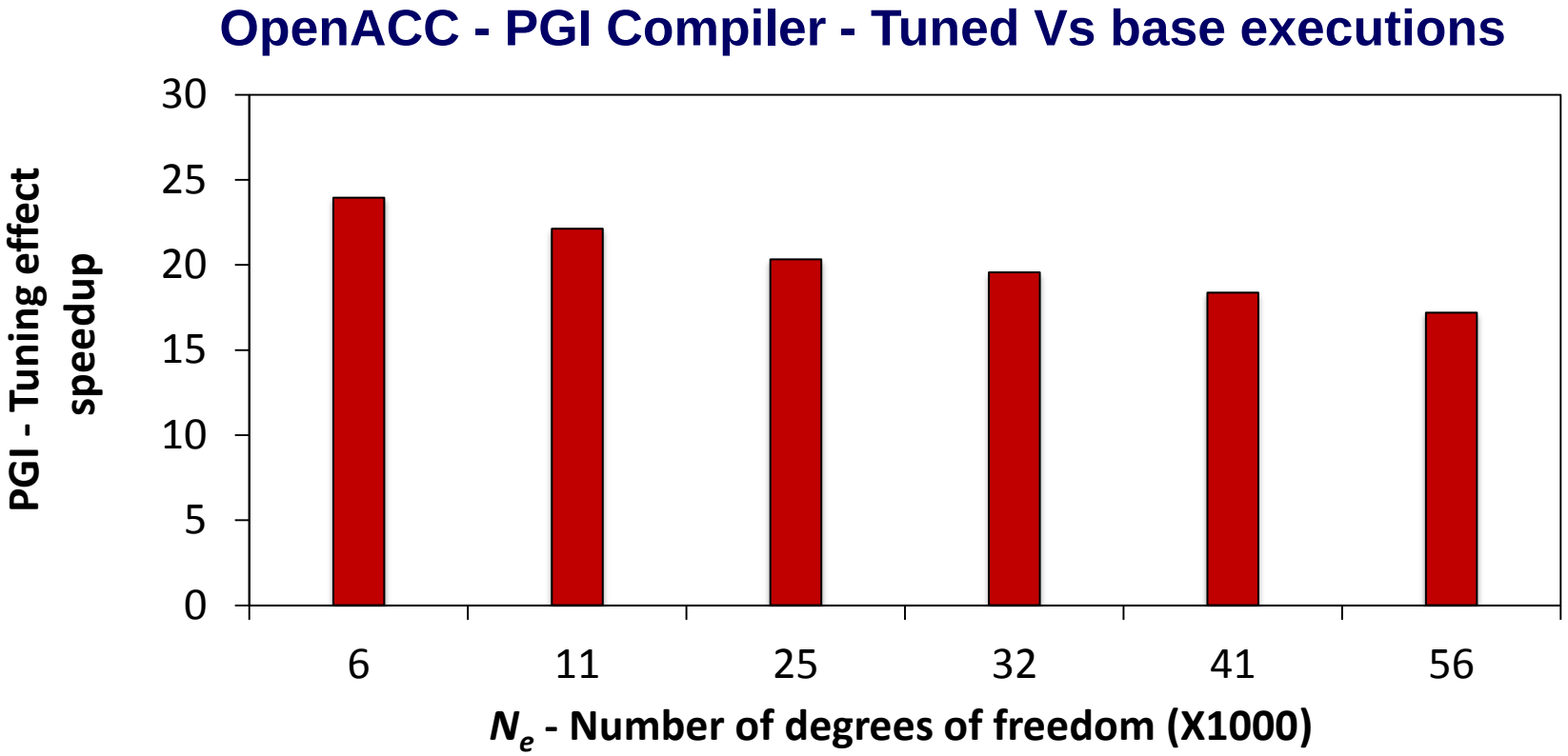
```
#pragma acc kernels
{
for ( l = 0 ;  l < nt; ++l) {  // time loop
// spatio-temporal convolution computation
#pragma acc loop independent collapse (3)
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < n; ++j){
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
      }
    }
  }

// spatial finite difference operations
#pragma acc loop independent collapse (3)
  for (int i = 0; i < n; ++i){
    for (int j = 0; j < n; ++j){
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
      }
    }
  }
 } // end time loop
}
```

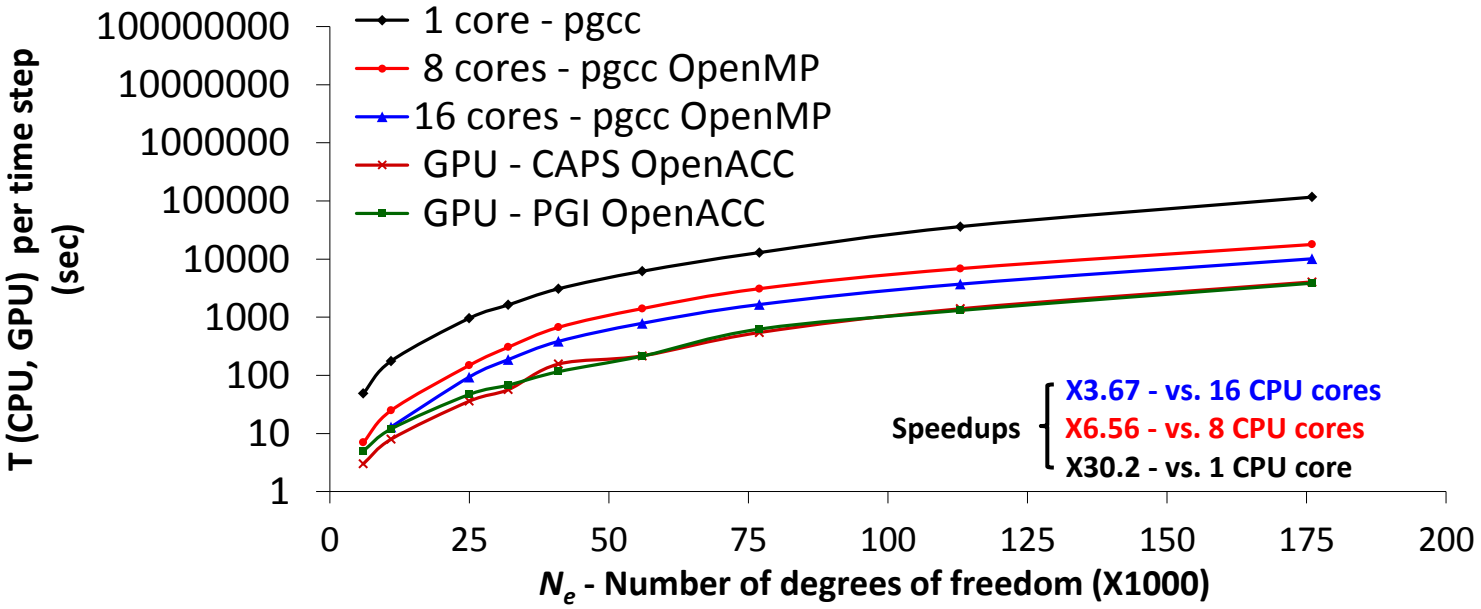
PGI

```
#pragma acc data
for ( l = 0 ;  l < nt; ++l) {  // time loop
// spatio-temporal convolution computation
#pragma acc kernels
#pragma acc loop independent gang
  for (int i = 0; i < n; ++i){
#pragma acc loop independent gang,vector
    for (int j = 0; j < n; ++j){
#pragma acc loop independent gang,vector
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
   } } }
// spatial finite difference operations
#pragma acc kernels
#pragma acc loop independent gang
  for (int i = 0; i < n; ++i){
#pragma acc loop independent gang,vector
    for (int j = 0; j < n; ++j){
#pragma acc loop independent gang,vector
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
   } } }
 } // end time loop
```

Tuning Efficiency



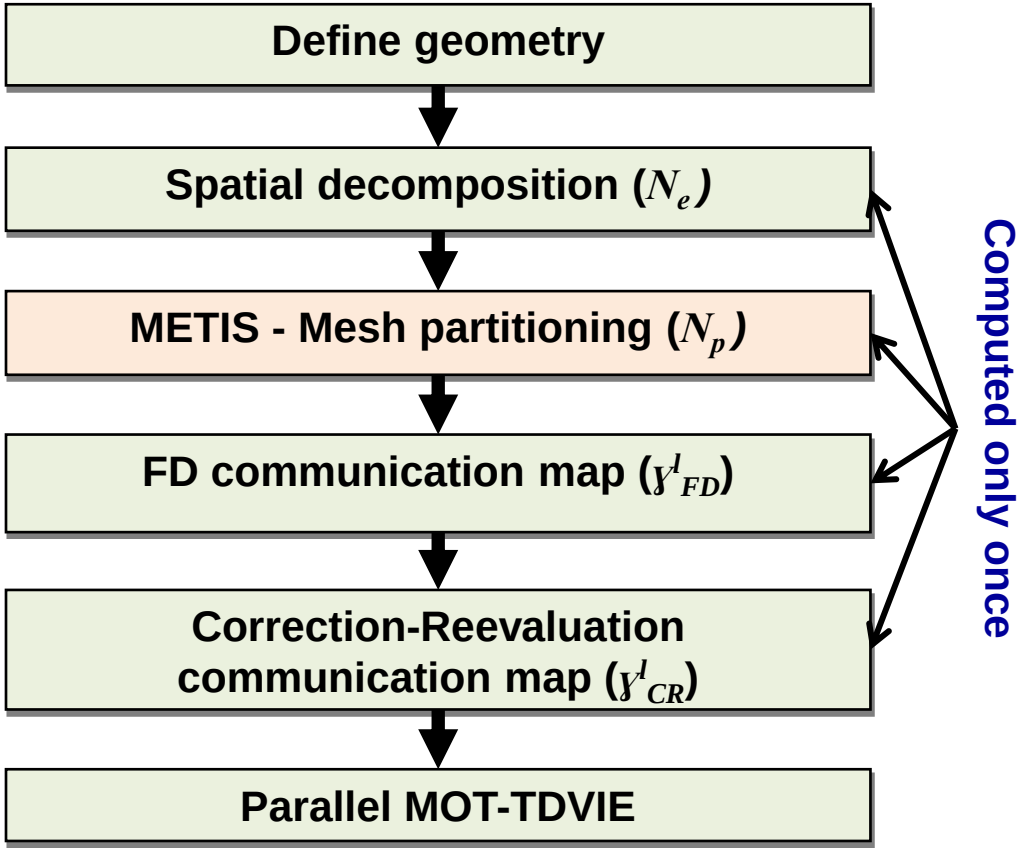
GPGPU Acceleration: OpenACC

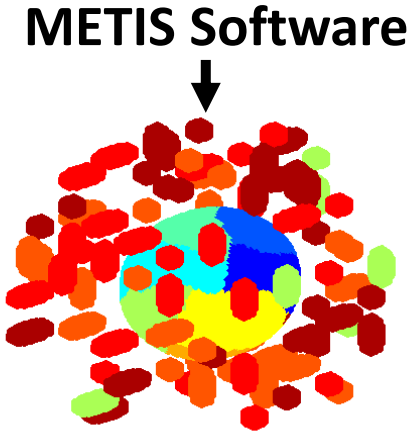


SandyBridge:	1 core	2.0 GHz	16 Gflops	32 GB Memory
SandyBridge:	8 core	2.0 GHz	128 Gflop	32 GB Memory
SandyBridge:	16 core	2.0 GHz	256 Gflops	64 GB Memory
GPU Kepler K20 :	2688 cores	732 Mhz	1.17 Tflops	6.1 GB Memory

S. Feki, A. Al-Jarro, A. Clo, and H. Bağcı, "Porting an explicit time domain volume integral equation solver on GPUs with OpenACC," IEEE Antennas Propag. Mag., Vol. 56, Issue 2, pp. 265-277, April, 2014.

GPGPU Acceleration: MPI-OpenACC





$$N^l_{ep} \Big/ N^k_{ep} \approx 1,\ (l,k) = 1,\ldots, N_p$$

$$\gamma^l_{FD} \Big/ \gamma^k_{FD} \approx 1,\ (l,k) = 1,\ldots, N_p$$

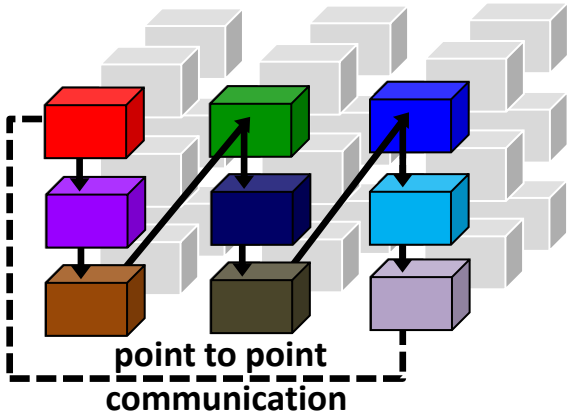
$$\gamma^l_{CR} \Big/ \gamma^k_{CR} \approx 1,\ (l,k) = 1,\ldots, N_p$$

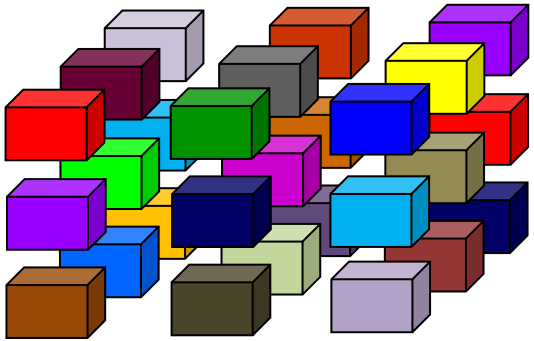
MOT-TDVIE Solver: Parallelization

Sources are equally distributed amongst processes

$$O\left(\left(N_e N_g\right) / N_p\right)$$

can also use hybrid MPI/OpenMP (OpenACC) parallelization scheme





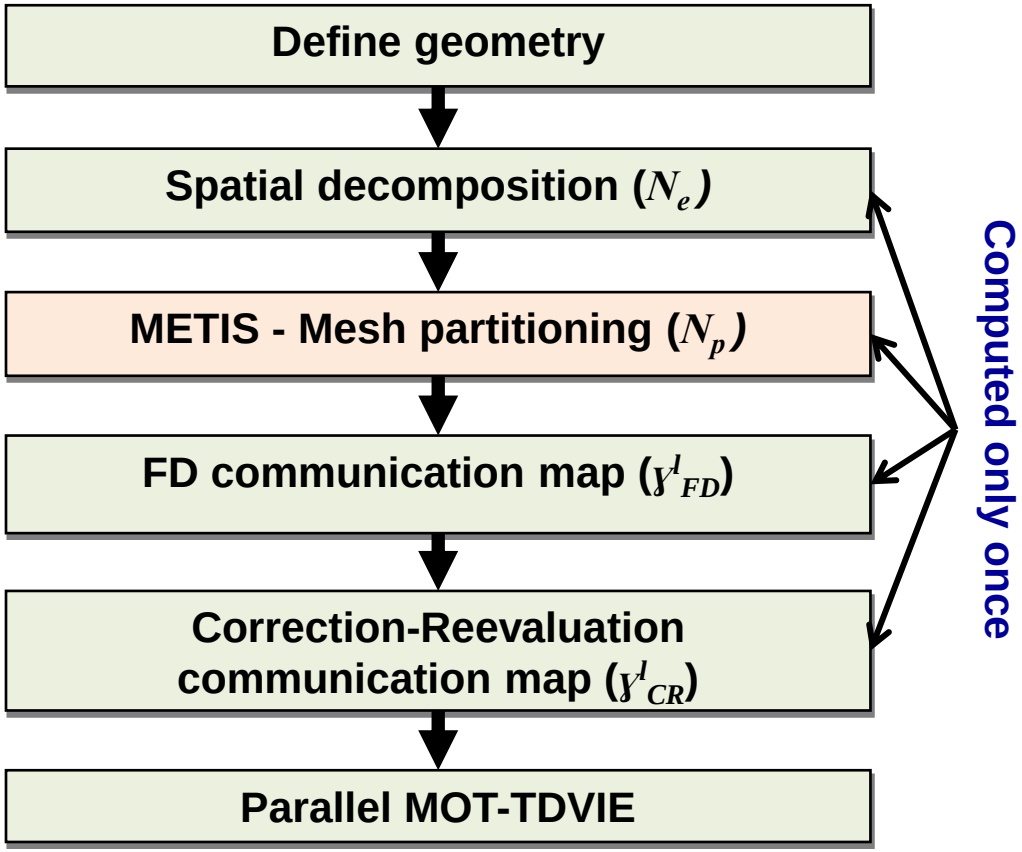
point to point communication strategy : MPI_SendReceive

- Observation (tested) field samples, operations pertinent to the computation of tested fields at current time step and sources are equally distributed among processors
- The full time history N_g is continuously updated on all processors during time marching

---

A. Al-Jarro , *et al.,* "A distributed-memory parallelization of the explicit time-domain volume integral equation solver using a rotating tiles paradigm," 28th International Review of Progress in Applied Computational Electromagnetics, Ohio, USA, April 10-14, 2012.

GPGPU Acceleration: MPI-OpenACC




```
#pragma acc data
for ( l = 0 ;  l < nt; ++l) {  // time loop
// spatio-temporal convolution computation
// MPI library communication routines -
MPI_Sendrecv(); MPI_Barrier();
#pragma acc kernels
#pragma acc loop independent gang
  for (int i = 0; i < n; ++i){
#pragma acc loop independent gang,vector
    for (int j = 0; j < n; ++j){
#pragma acc loop independent gang,vector
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
   } } }
// spatial finite difference operations
// MPI library communication routines -
MPI_Sendrecv(); MPI_Barrier();
#pragma acc kernels
#pragma acc loop independent gang
  for (int i = 0; i < n; ++i){
#pragma acc loop independent gang,vector
    for (int j = 0; j < n; ++j){
#pragma acc loop independent gang,vector
      for (int k = 0; k < n; ++k){
        B[i][j][k] = B[i]][j][k] + ....
   } } }
 } // end time loop
```

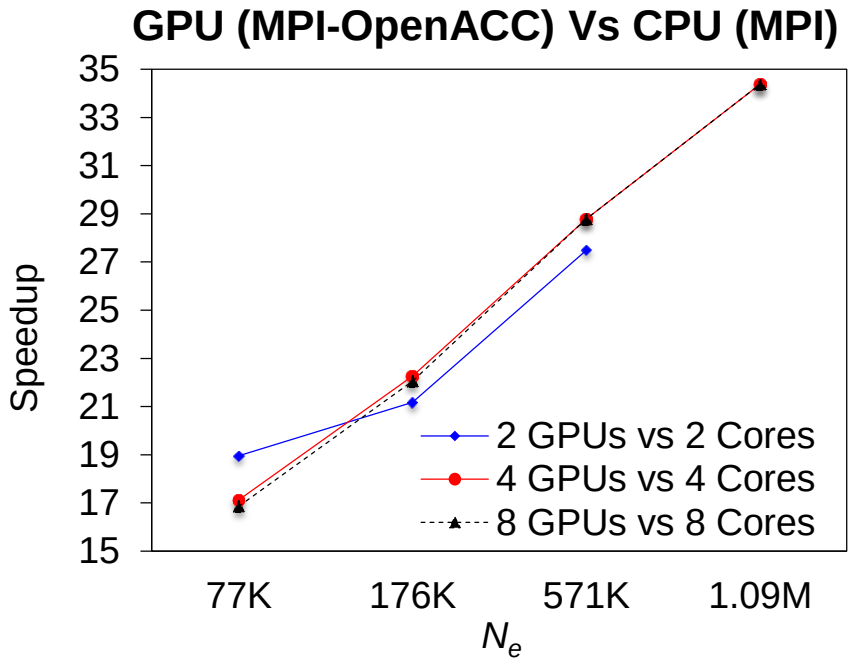
GPGPU Acceleration: MPI-OpenACC

Hardware

- **Dual Socket SandyBridge Intel(R) Xeon(R) CPU E5-2650 0**
  - **8 cores / socket**
  - **2.00GHz**
  - **64 GB**
  - **Infiniband FDR 4X (56 Gbit/s)**

- **4 GPUs NVIDIA Kepler - K20c**
  - **2496 CUDA cores / GPU**
  - **700Mhz**
  - **5GB GDDR5 / GPU**
  - **Connected to CPU through PCIe Gen 3 X16**

Speedup



MPI-OpenACC on K20c GPUs Vs MPI on SandyBridge CPUs

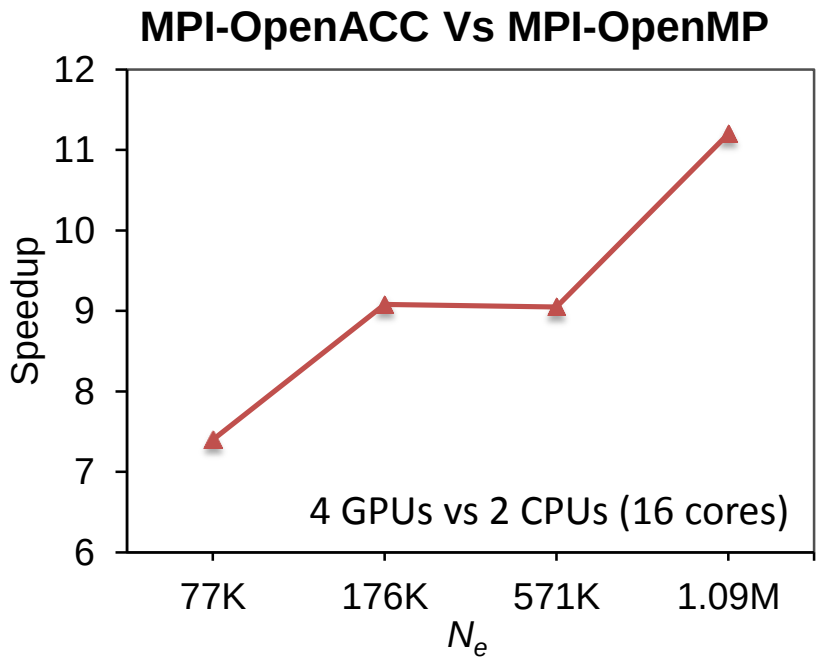
GPGPU Acceleration: MPI-OpenACC

Hardware

- **Dual Socket SandyBridge Intel(R) Xeon(R) CPU E5-2650 0**
  - **8 cores / socket**
  - **2.00GHz**
  - **64 GB**
  - **Infiniband FDR 4X (56 Gbit/s)**

- **4 GPUs NVIDIA Kepler - K20c**
  - **2496 CUDA cores / GPU**
  - **700Mhz**
  - **5GB GDDR5 / GPU**
  - **Connected to CPU through PCIe Gen 3 X16**

Speedup



MPI-OpenACC on 4 K20c GPUs Vs MPI-OpenMP on 2 SandyBridge CPUs

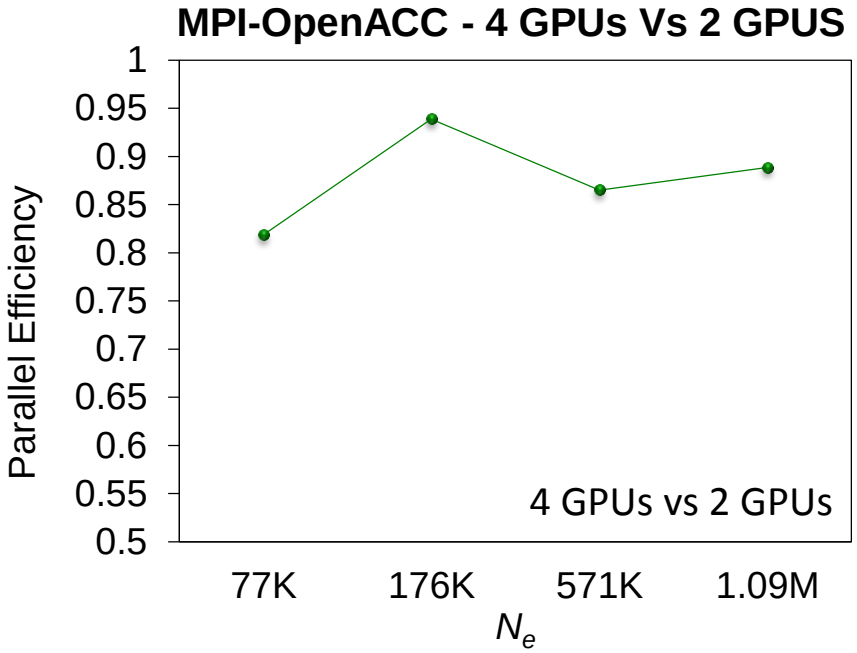
GPGPU Acceleration: GPU Weak-scaling

Hardware

- **Dual Socket SandyBridge Intel(R) Xeon(R) CPU E5-2650 0**
  - **8 cores / socket**
  - **2.00GHz**
  - **64 GB**
  - **Infiniband FDR 4X (56 Gbit/s)**

- **4 GPUs NVIDIA Kepler - K20c**
  - **2496 CUDA cores / GPU**
  - **700Mhz**
  - **5GB GDDR5 / GPU**
  - **Connected to CPU through PCIe Gen 3 X16**

Parallel efficiency - GPU



MPI-OpenACC - 4 K20c GPUs Vs 2 K20c GPUs

CPU & GPU Strong-scaling - EMERALD

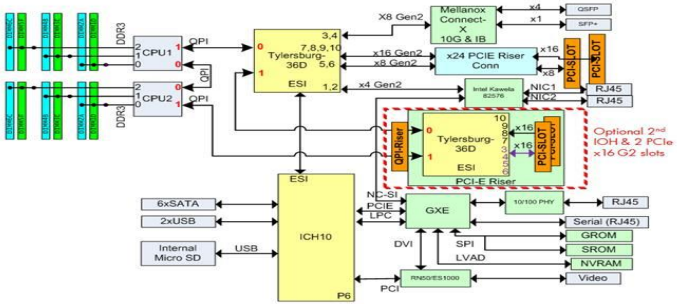
EMERALD Cluster

- 60 x 2 CPUs (nodes) X5650 Xeon
  - 3 M2090 NVIDIA GPUs
  - 48 GB
- 24 x 2 CPUs (nodes) X5650 Xeon
  - 8 M2090 NVIDIA GPUs
  - 96 GB
- Both Systems
  - 12 cores / CPU
  - 2.00GHz
  - 48 GB
  - 10 Gbit Ethernet
  - 3 x 512core M2090 NVIDIA GPUs

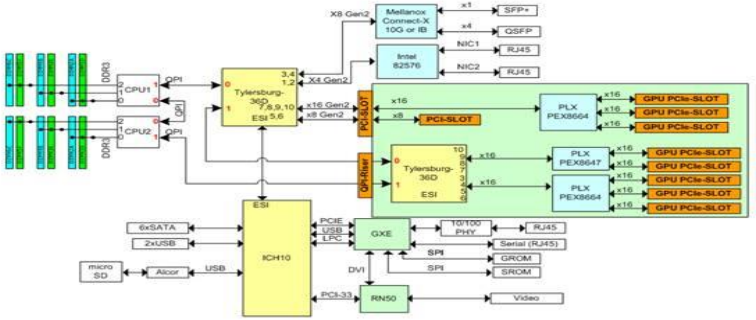
EMERALD GPUs

- 372 NVIDIA M2090 GPUs
  - 512 CUDA cores / GPU
  - 1.3 Ghz
  - 6 GB GDDR5 / GPU
  - Connected to CPU through PCIe Gen2

SL390s Block Diagram 0 to 3 GPUs

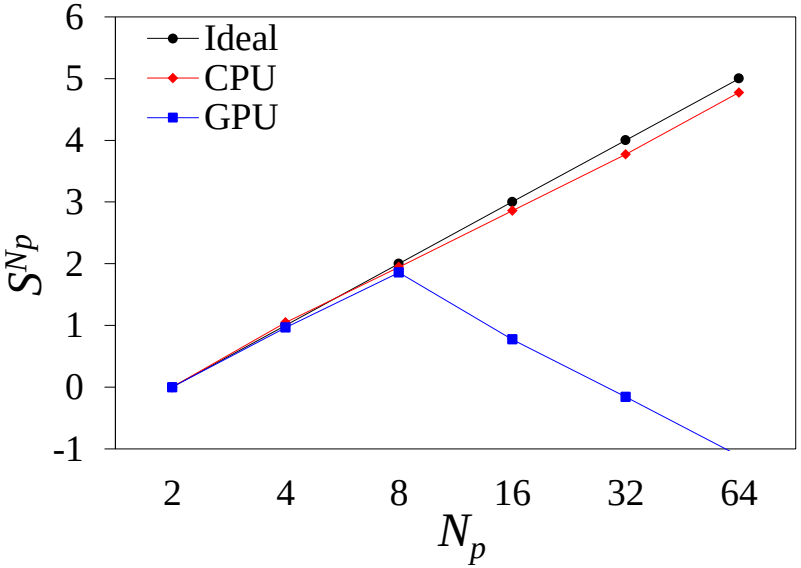


SL390 8 GPU Block Diagram



Strong-scaling: Small Problem Size

$N_e = 77\,653$



$N_g = 108, \quad 3N_g = 324$

Partitions	load	CPU	CPU	GPU	GPU
N_p	N_{ep}	$S_{CPU}^{N_p}$	$T_{CPU}^{N_p}$	$S_{GPU}^{N_p}$	$T_{GPU}^{N_p}$
2	38826	0	14.12	0	13.79
4	19413	1.051	6.825	0.967	7.056
8	9706	1.947	3.661	1.858	3.805
16	4853	2.857	1.949	0.775	8.060
32	2426	3.769	1.035	-0.1539	15.35
64	1213	4.773	0.516	-1.113	29.86
128	606	.	.	.	.
256	303	.	.	.	.

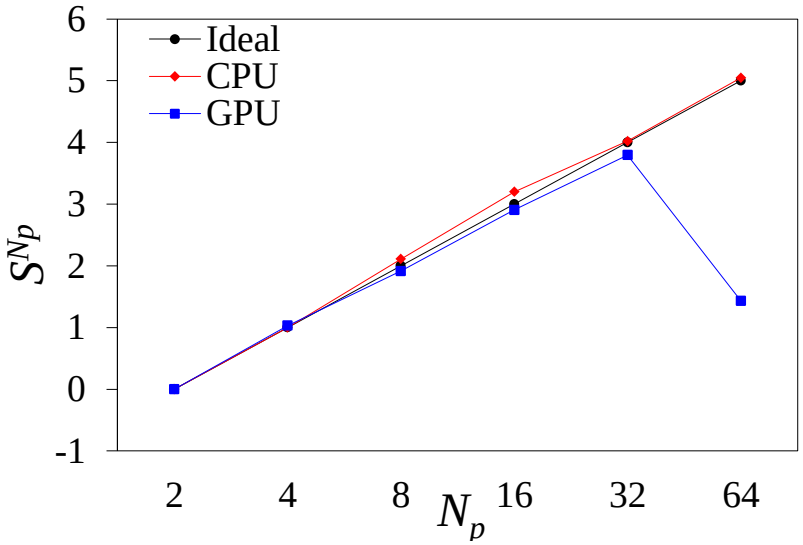
N_p = number of partitions used – one partition per CPU/GPU

T^{N_p} = communication and computation times

$S^{N_p} = \log_2(T^{N_p} / T^{ref})$; T_{ref} is reference total time with the lowest number of partitions

Strong-scaling: Medium Problem Size

$N_e = 176\,773$



$N_g = 143, \quad 3N_g = 429$

Partitions	load	CPU	CPU	GPU	GPU
N_p	N_{ep}	$S_{CPU}^{N_p}$	$T_{CPU}^{N_p}$	$S_{GPU}^{N_p}$	$T_{GPU}^{N_p}$
2	88386	0	91.25	0	85.02
4	44193	1.041	45.63	1.032	41.56
8	22096	2.110	21.13	1.916	22.52
16	11048	3.199	9.933	2.907	11.33
32	5524	4.021	5.619	3.795	6.123
64	2762	5.046	2.761	1.430	31.55
128	1381	.	.	.	.
256	690	.	.	.	.

N_p = number of partitions used – one partition per CPU/GPU

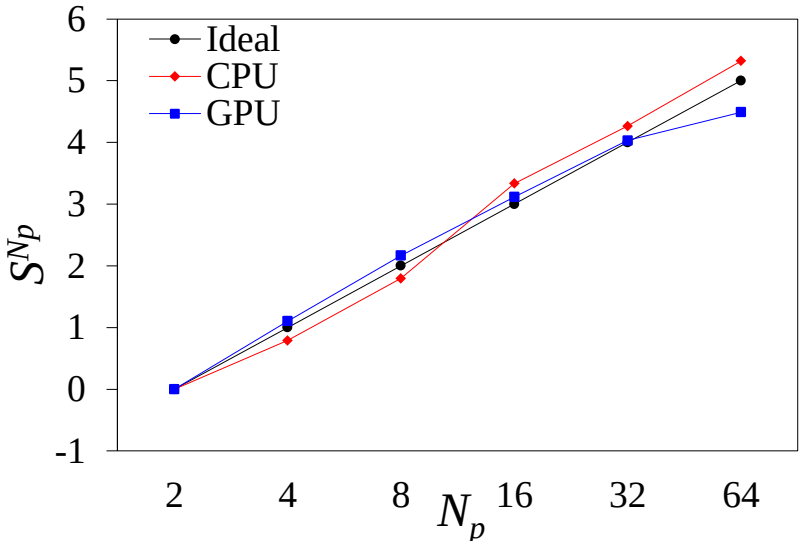
T^{N_p} = communication and computation times

$S^{N_p} = \log_2(T^{N_p} / T^{ref})$; T_{ref} is reference total time with the lowest number of partitions

Strong-scaling: Large Problem Size

$N_e = 571\,595$

$N_g = 211, \quad 3N_g = 633$



Partitions	load	CPU	CPU	GPU	GPU
N_p	N_{ep}	$S_{CPU}^{N_p}$	$T_{CPU}^{N_p}$	$S_{GPU}^{N_p}$	$T_{GPU}^{N_p}$
2	285797	0	1556.54	0	1324.6
4	142898	0.790	899.86	1.105	615.7
8	71449	1.796	448.00	2.167	294.7
16	35724	3.334	154.27	3.115	152.82
32	17862	4.265	80.916	4.030	81.03
64	8931	5.319	38.991	4.489	58.95
128	4465	.	.	.	.
256	2232	.	.	.	.

N_p = number of partitions used – one partition per CPU/GPU

T^{N_p} = communication and computation times

$S^{N_p} = \log_2(T^{N_p} / T^{ref})$; T_{ref} is reference total time with the lowest number of partitions

Experiences in Porting Scientific Applications to GPUs Using OpenACC

Thank You!

Saber Feki and Ahmed Al-Jarro

KAUST
King Abdullah
University of Science and Technology

GTC 2015 - San Jose CA - March 2015

FUJITSU