Fourth Workshop on Accelerator Programming Using Directives (WACCPD),
Nov. 13, 2017

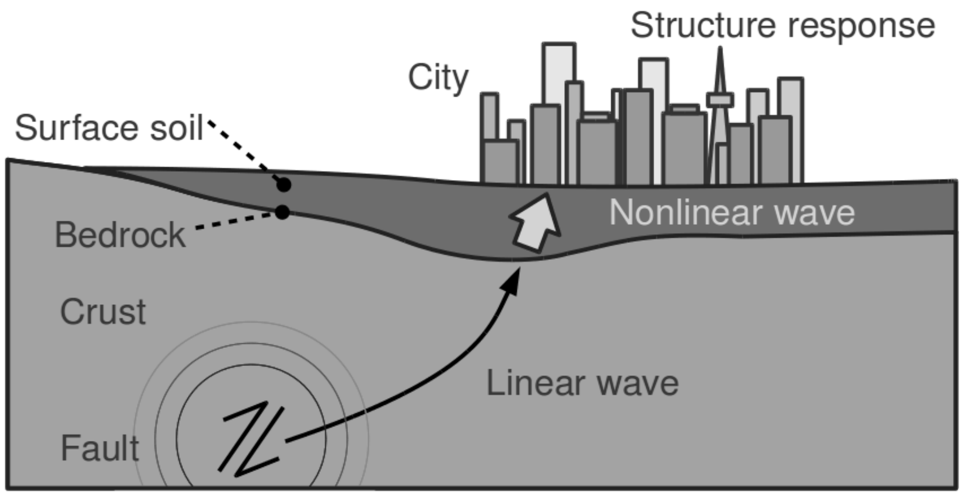
# Implicit Low-Order Unstructured Finite-Element Multiple Simulation Enhanced by Dense Computation using OpenACC

Takuma Yamaguchi, Kohei Fujita, Tsuyoshi Ichimura,
Muneo Hori, Lalith Maddegedara, Kengo Nakajima

THE UNIVERSITY OF TOKYO

# Introduction

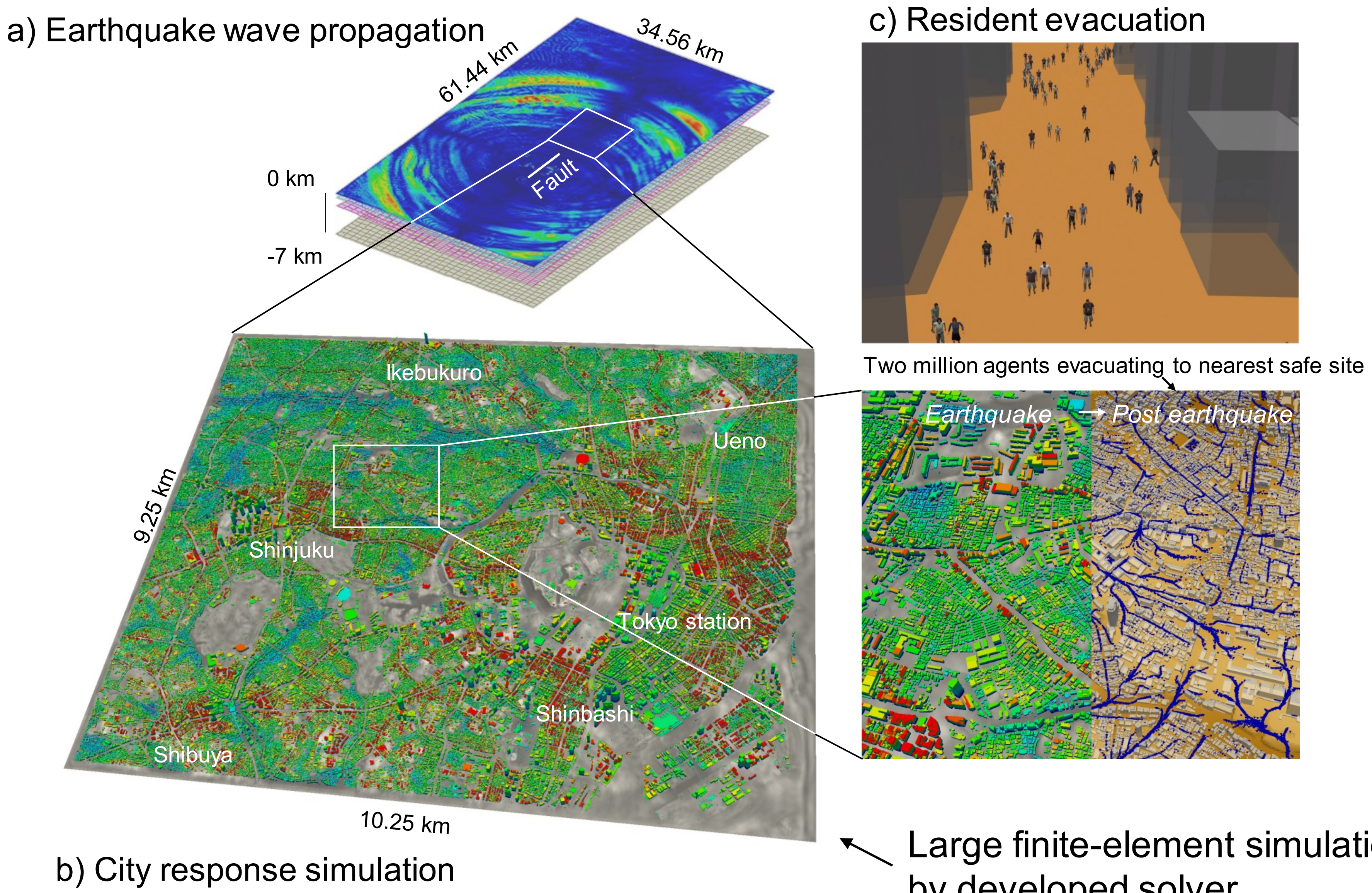
- Contribution of HPC to earthquake mitigation highly anticipated from society
- We are developing comprehensive earthquake simulation that simulate all phases of earthquake disaster by use of full K computer system
  - Simulate all phases of earthquake by speeding up core solver
  - **SC14 Gordon Bell Prize Finalist, SC15 Gordon Bell Prize Finalist & SC16 Best Poster & SC17 Best Poster Finalist**
- Ported this solver to GPU environment using OpenACC in WACCPD 2016 (**Best Paper**)
- Today's topic is enhancement of this GPU solver, and report performance on Pascal and Volta GPUs



Earthquake disaster process


K computer: 8 core CPU x 82944 node system with peak performance of 10.6 PFLOPS

# Comprehensive earthquake simulation



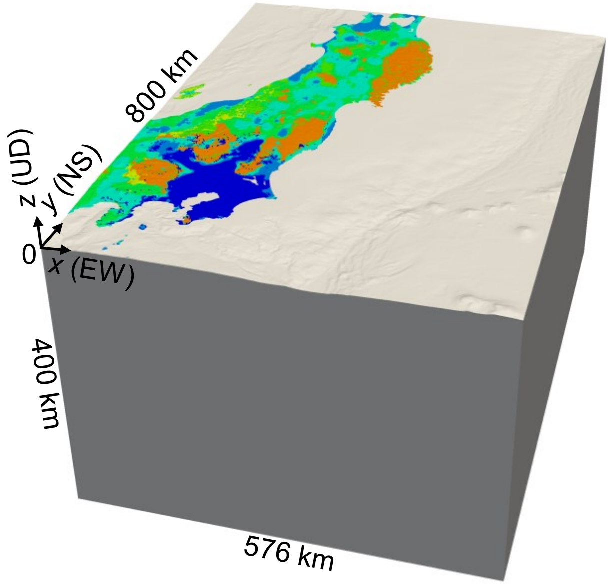

Large finite-element simulation enabled by developed solver

City simulation


Visualized by CYBERNET SYSTEMS CO., LTD

# Target problem: Earth's crust deformation problem

- Compute elastic response to given fault slip
  - Many case analysis required for inverse analyses and Monte Carlo simulations
- Compute using finite-element method: solve large matrix equation many times
  - Involves many random data access & communication
- Difficulty of problem
  - Attaining load balance & peak-performance & convergency of iterative solver & short time-to-solution at same time
  - Smart use of compute precision space, constraints in solver search space according to physical solution space required




Earth's crust deformation problem

$$Ku = f$$

Outer force vector

Unknown vector with up to 1 trillion degrees of freedom

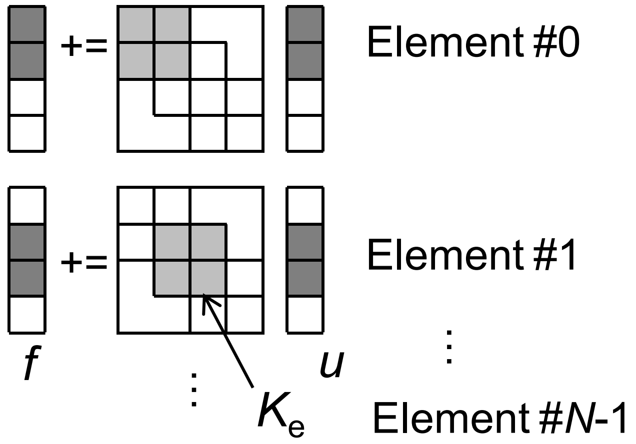
Sparse, symmetric positive definite matrix

# Designing scalable & fast finite-element solver

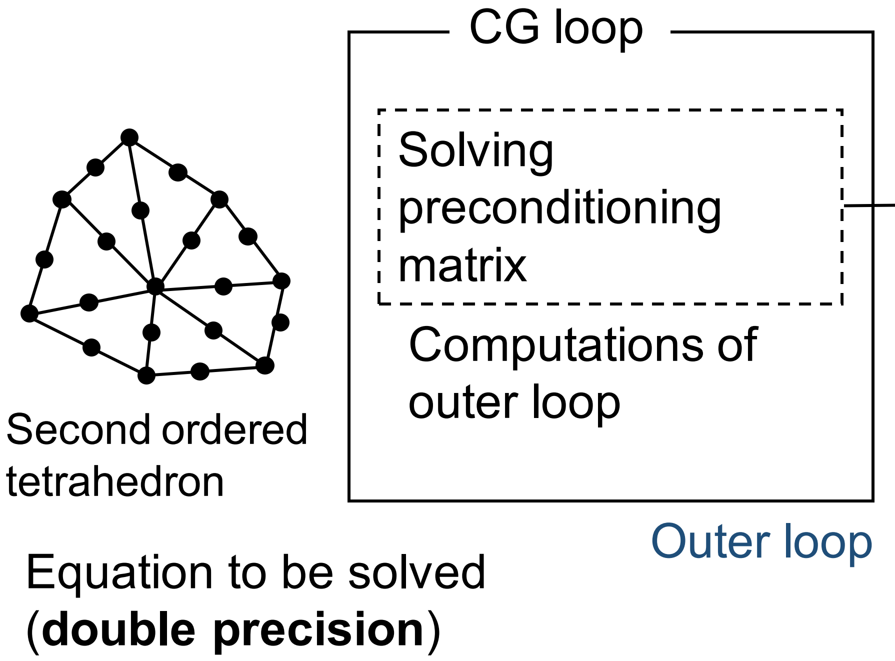
- Design algorithm that can obtain equal granularity at O(million) cores
  - Matrix-free matrix-vector product (Element-by-Element method) is promising: Good load balance when elements per core is equal
    - Also high-peak performance as it is on-cache computation
- Combine Element-by-Element method with multi-grid, mixed precision arithmetic, and adaptive conjugate gradient method
  - Scalability & peak-performance good (core computation kernels are Element-by-Element), convergency good, time-to-solution good

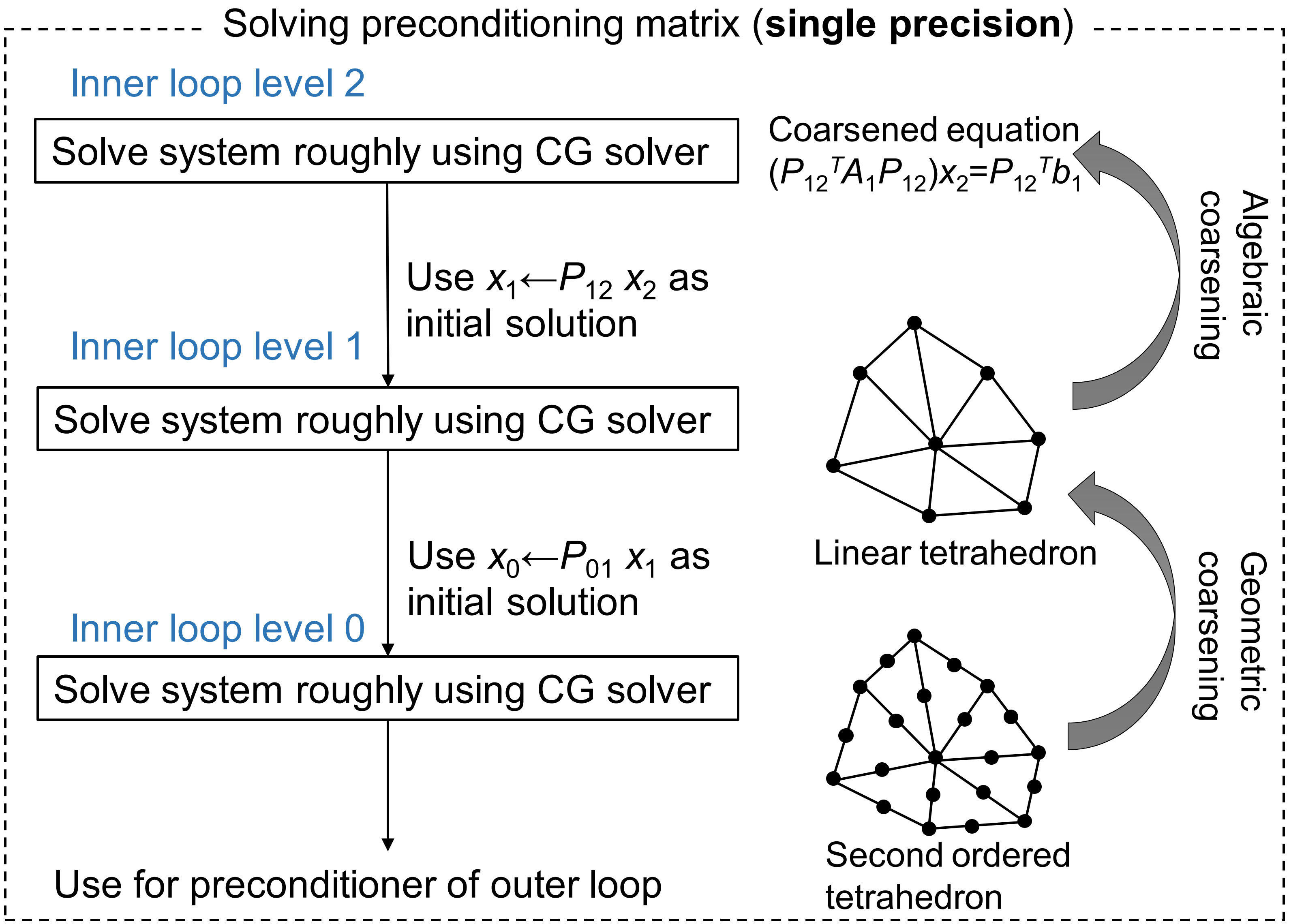
Element-by-Element method

$f = \Sigma_e P_e K_e P_e^T u$
[$K_e$ is generated on-the-fly]



# Solver algorithm

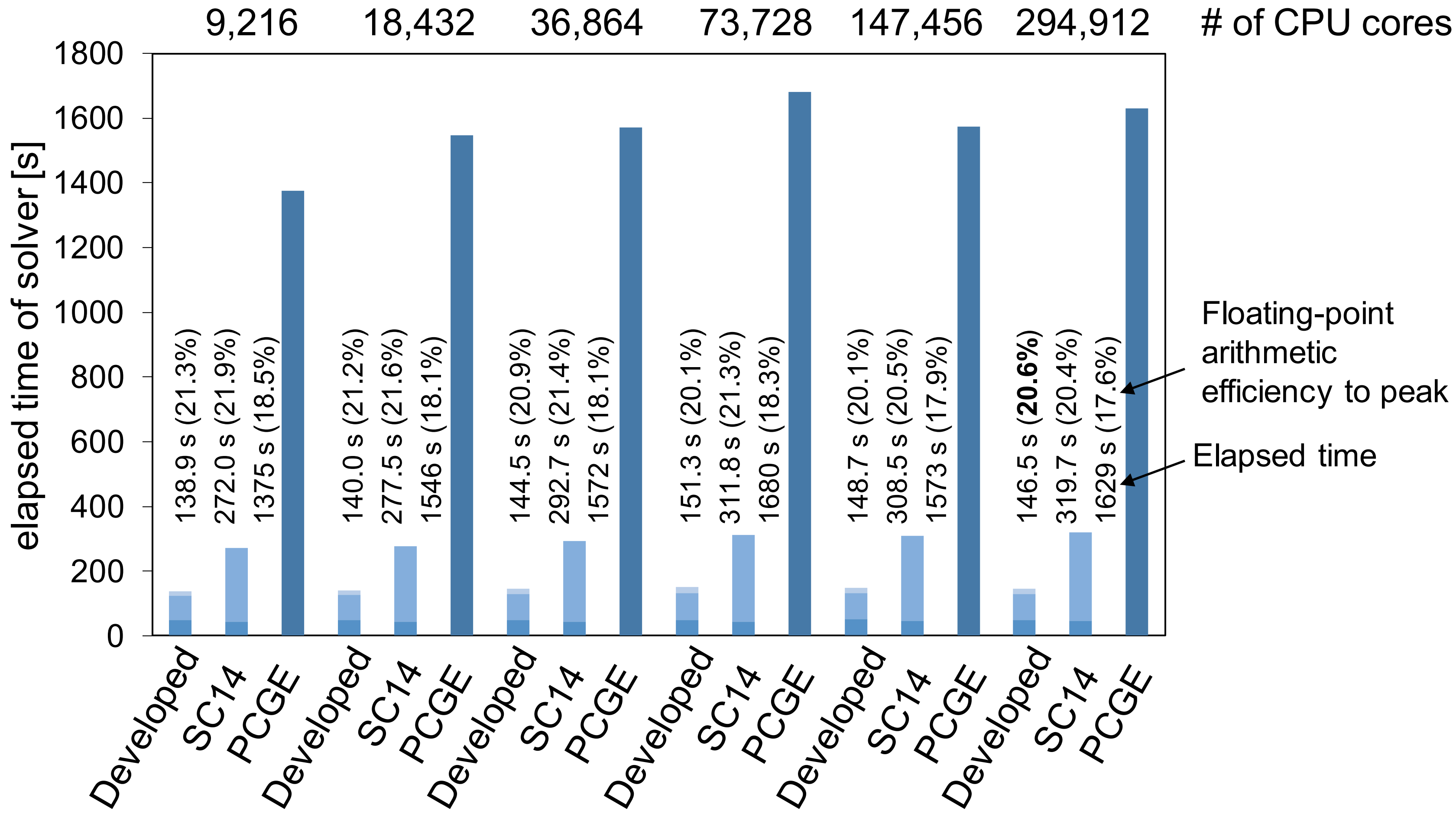



K. Fujita, T. Ichimura, K. Koyama, H. Inoue, M. Hori, L. Maddegedara, Proceedings of the Platform for Advanced Scientific Computing Conference (PASC), June 2017

# Performance on K computer

- Developed solver significantly faster than
  - PCGE (standard CG solver algorithm; preconditioning with 3x3 block diagonal matrix)
  - SC14 Gordon Bell Prize finalist solver (base solver for WACCPD 2016 GPU solver)
- Use this as a base for GPU solver




- 94.8% scalability from 9216 to 294912 cores
- 4 times peak performance of HPCG benchmark (HPCG on K computer: 5.3% in double precision)

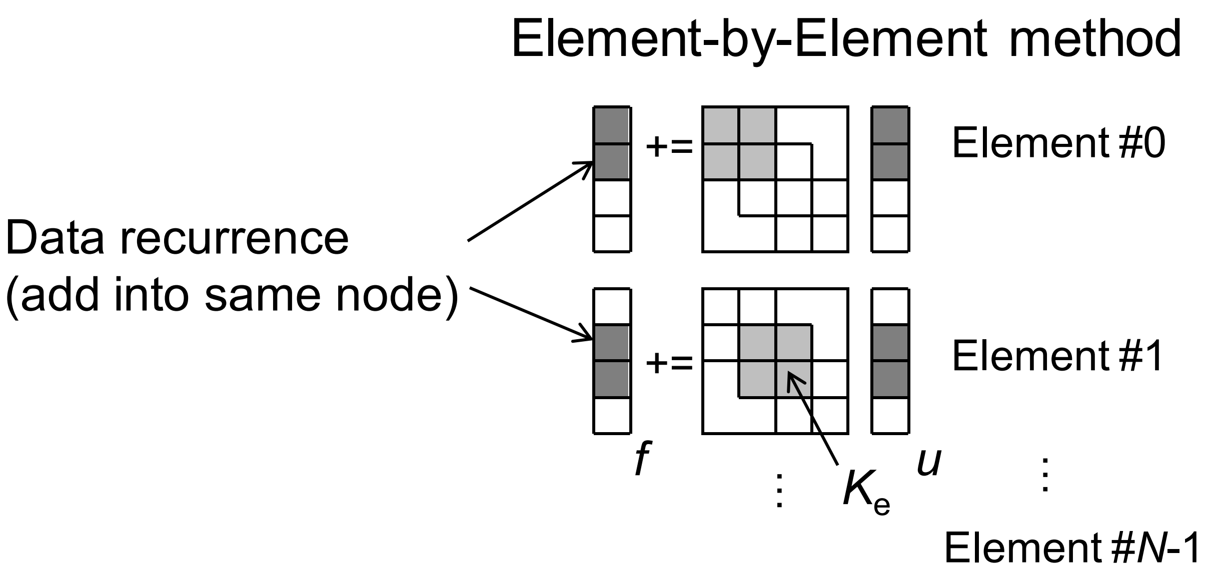
# Introduction of GPU computations

- Further speedup of the simulation by introducing GPUs
  - Good load balance, Reduced computation cost & data transfer size is also beneficial for GPUs
  - High performance can be obtained using OpenACC with low development cost
- GPU architecture is different from CPU architecture
  - Latency bound especially when we conduct random memory access
  - Relatively smaller cache size
- Finite-element applications tend to be memory bandwidth bound

➔Simple porting of the CPU code is not sufficient

# Key kernel: Element-by-Element kernel

- Most costly kernel; involves data recurrence
- Algorithm for avoiding data recurrence on **CPUs**
  - Use temporary buffers per core & per SIMD lane
  - Suitable for small core counts with large cache capacity
- Algorithm for avoiding data recurrence on **GPUs**
  - Last year, we developed algorithm using atomics and achieved high performance
  - However, random access becomes bottleneck…

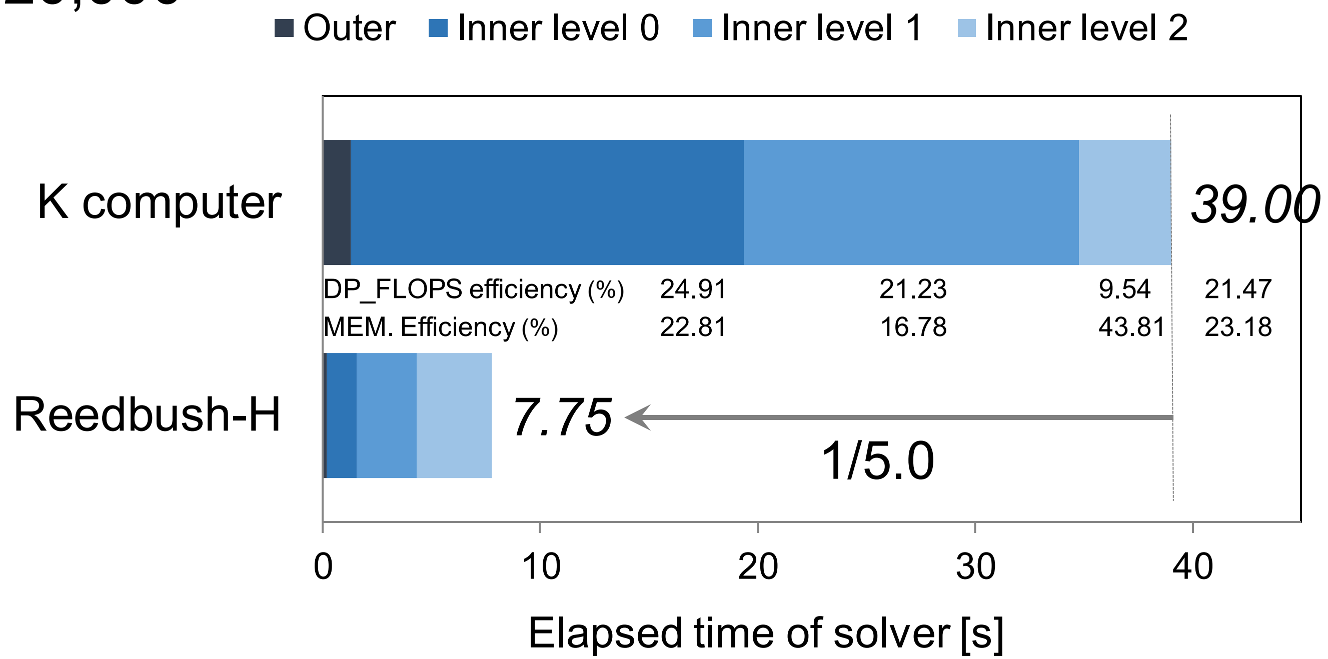


# Performance in simple porting

DOF: 125,177,217, # of elements: 30,720,000

Computational Environment

| | K computer | Reedbush-H |
|---|---|---|
| # of nodes | 20 | 10 |
| CPU/node | 1 x SPARC64 VIIIfx | 2 x Intel Xeon E5-2695 v4 |
| GPU/node | -- | 2 x NVIDIA P100 |
| Hardware peak FLOPS /process | 128 GFLOPS | 5.30 TFLOPS |
| Memory bandwidth /process | 64 GB/s | 732 GB/s |




- Simple porting achieved 5.0 times speedup
- However, there is some room for improvement
  - Memory bandwidth is 11 times larger

# Strategy for Introduction of OpenACC

- To attain optimal performance, algorithm/implementation suitable for GPUs should differ from that for CPUs

Thereby, we

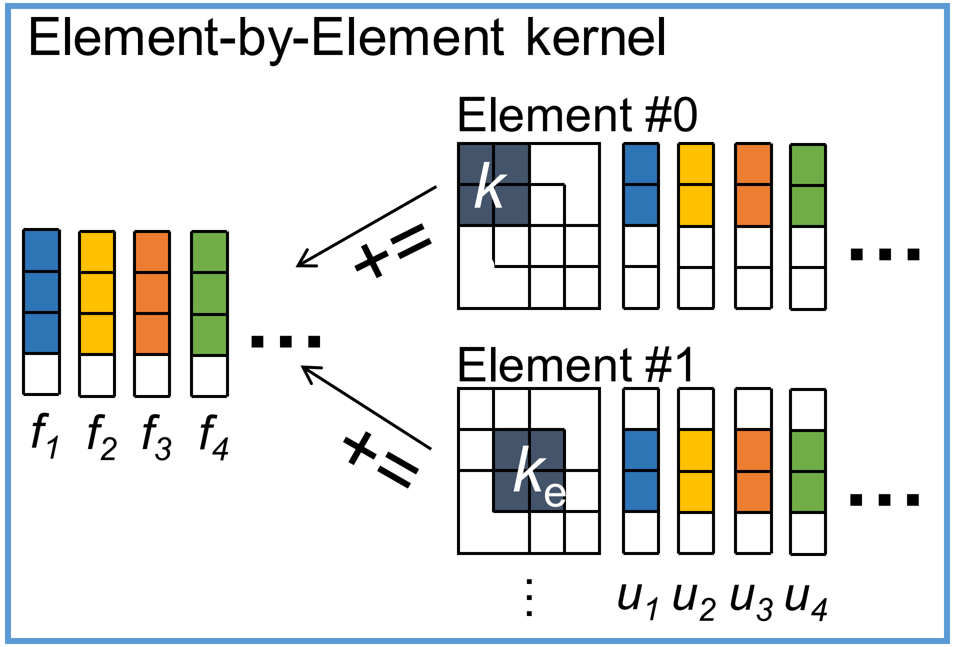
1. Modify the solver algorithm to suit the GPU architectures
2. Port the solver to GPUs using OpenACC

# Modification of Algorithm for GPUs

- Reduce random memory accesses
- Target applications (Inverse analyses, Monte Carlo method etc.) solve many systems of equations
  - Same stiffness matrix
  - Different right-hand side input vectors
- Multiple equations at the same time

$$\mathrm{K}[\mathrm{u}_1, \mathrm{u}_2, \mathrm{u}_3, \ldots, \mathrm{u}_{16}]^T = [\mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \ldots, \mathrm{f}_{16}]^T$$

Instead of $\mathrm{Ku}_1 = \mathrm{f}_1, \mathrm{Ku}_2 = \mathrm{f}_2, \mathrm{Ku}_3 = \mathrm{f}_3, \ldots$



# Introduction of OpenACC – 1/3

## Control of data transfer

Read data

Input vector **f**

CPU computation

***!$acc data copy(u, …) copyin(f, …)***

Solve $\mathbf{u}=\mathbf{K}^{-1}\mathbf{f}$

Computation (i.e, EBE)

***!$acc update host(err)***

Check convergence

***!$acc end data***

GPU computation

Output vector **u**

Data transfer between CPU and GPUs is minimized in the solver

- Only in convergence check part
- GPU Direct is used for MPI communication

# Introduction of OpenACC – 2/3

Insertion of some directives for parallel computation

Example for Element-by-Element multiplication

- Assign 16 threads for one element
- Introduce atomic functions to avoid data race

```
!$acc parallel loop collapse(2) present(…
do i_ele = 1, n_element
do i_vec = 1, 16
cny1 = connect(1, i_ele)
    ⋮
cny10 = connect(10, i_ele)

u0101 = u(i_vec, 1, cny1)
    ⋮
u1003 = u(i_vec, 3, cny10)

Ku01 = …
    ⋮
Ku30 = …

!$acc atomic
r(i_vec, 1, cny1) = r(i_vec, 1, cny1) + Ku01
    ⋮
!$acc atomic
r(i_vec, 3, cny10) = r(i_vec, 3, cny10) + Ku30
enddo
enddo
!$acc end parallel
```

# Introduction of OpenACC – 3/3

Minor tuning for OpenACC parameters

- The allocation of gang, worker and vector
- The length of vector

Optimize fine-grain control of parallelism

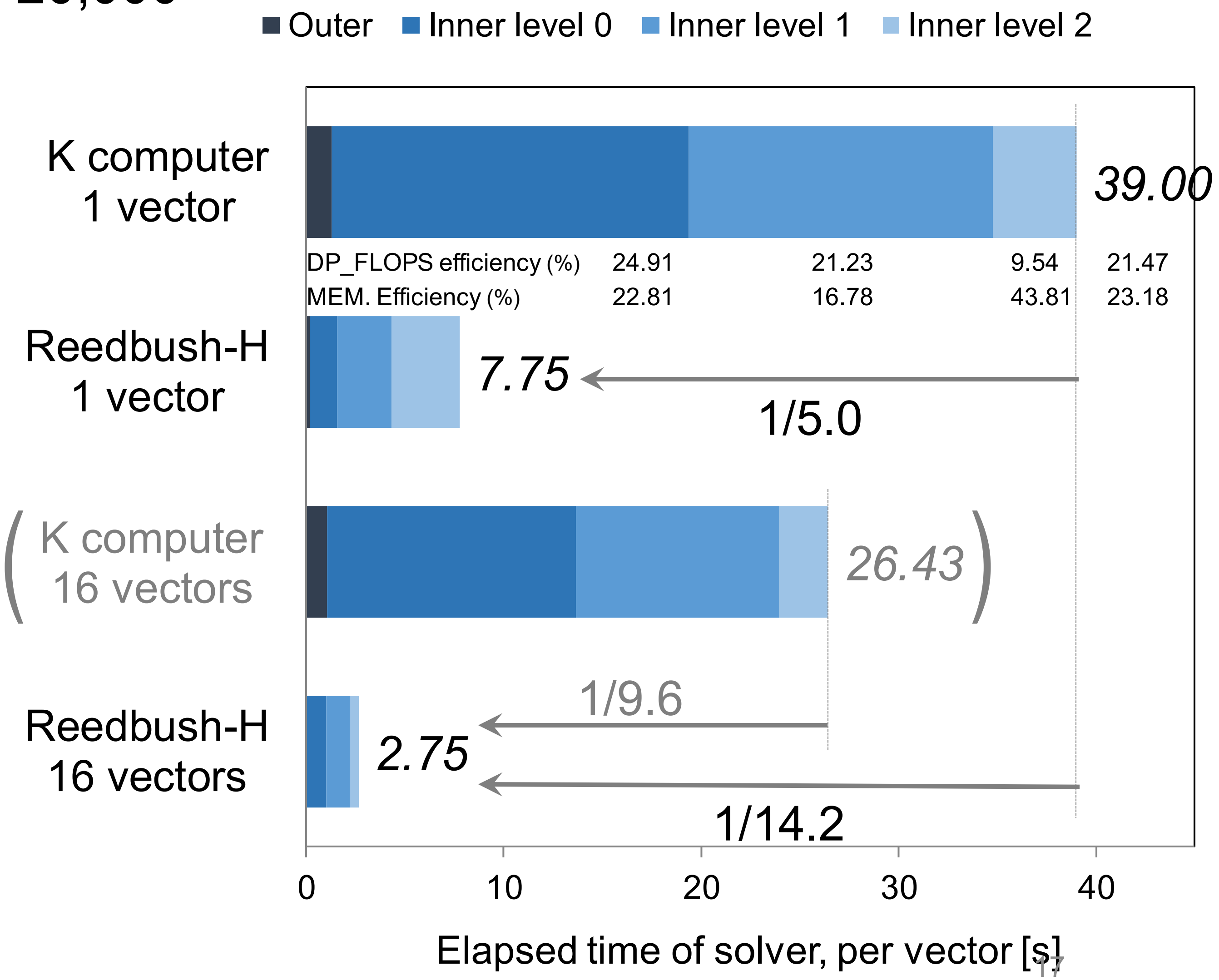
(Not large effect on performance)

# Performance of the proposed solver

DOF: 125,177,217, # of elements: 30,720,000

Computational Environment

| | K computer | Reedbush-H |
|---|---|---|
| # of nodes | 20 | 10 |
| CPU/node | 1 x SPARC64 VIIIfx | 2 x Intel Xeon E5-2695 v4 |
| GPU/node | -- | 2 x NVIDIA P100 |
| Hardware peak FLOPS /process | 128 GFLOPS | 5.30 TFLOPS |
| Memory bandwidth /process | 64 GB/s | 732 GB/s |



# The speedup of each kernel

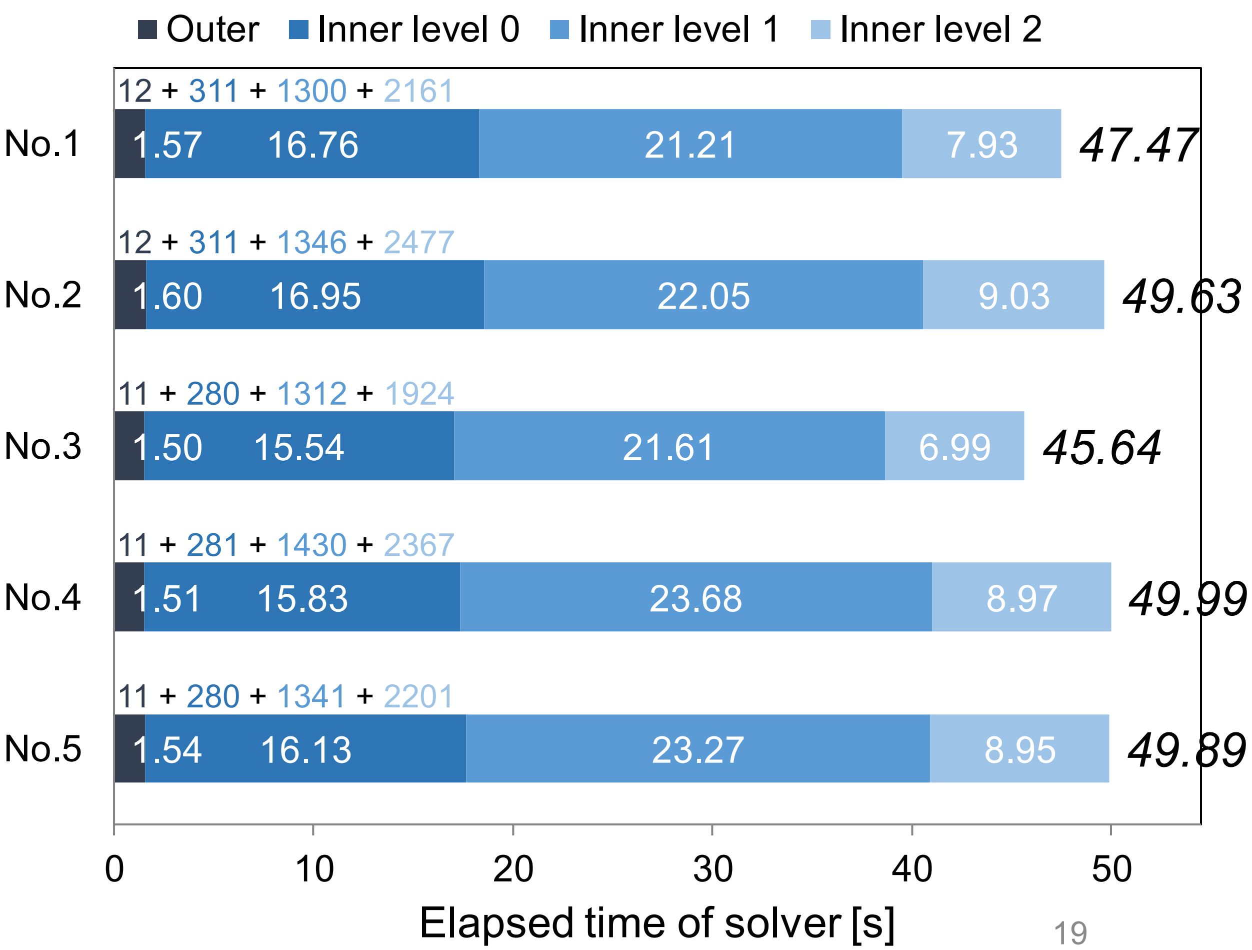
| Kernel | | Elapsed time per vector(s) 1 vector | 16 vectors | **Speedup** |
|---|---|---|---|---|
| Element-by-Element computation | $1^{st}$ order (FP32) | 0.948 | 0.584 | ***1.62*** |
| | $2^{nd}$ order (FP32) | 0.687 | 0.401 | ***1.71*** |
| | $2^{nd}$ order (FP64) | 0.044 | 0.025 | ***1.78*** |
| SpMV | | 1.465 | 0.091 | ***16.10*** |
| Dot product | | 0.213 | 0.522 | ***0.41*** |
| Total | | 7.75 | 2.75 | ***2.82*** |

- Reduction in random memory access in EBE kernels
- Total computation time for SpMV is constant
  - Bound by reading global matrix for memory
- Dot product is not efficiently computed
  - OpenACC cannot use arrays for reduction option
  - Using scalars (tmp1,tmp2,…,tmp16) causes stride memory access

# Weak Scaling

Reedbush-H: P100 GPU x 240

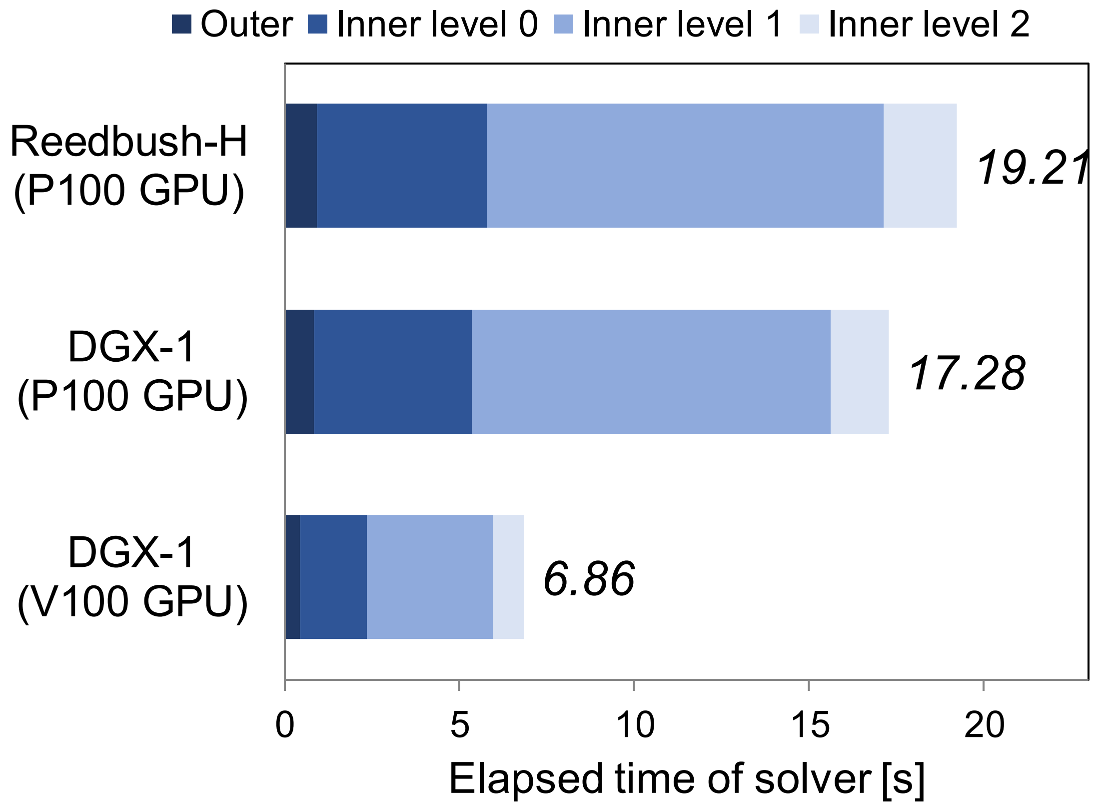
| Model | DOF | # of elements | PCGE iterations | # of GPUs |
|---|---|---|---|---|
| No. 1 | 125,177,217 | 30,720,000 | 4,928 | 20 |
| No. 2 | 249,640,977 | 61,440,000 | 4,943 | 40 |
| No. 3 | 496,736,817 | 122,880,000 | 4,901 | 80 |
| No. 4 | 992,038,737 | 245,760,000 | 4,905 | 160 |
| No. 5 | 1,484,953,857 | 368,640,000 | 4,877 | 240 |



# Performance in using V100 GPUs

DOF: 38,617,017, # of elements: 9,440,240

| | Computational Environment | | |
|---|---|---|---|
| | Reedbush-H | DGX-1 (P100/V100) | |
| # of nodes | 4 | 1 | |
| CPU/node | 2 x Intel Xeon E5-2695 v4 | 2 x Intel Xeon E5-2698 v4 | |
| GPU/node | 2 x NVIDIA P100 | 8 x NVIDIA P100 | 8 x NVIDIA V100 |
| Hardware peak FLOPS / process | 5.30 TFLOPS | 5.30 TFLOPS | 7.5 TFLOPS |
| Memory bandwidth /process | 732 GB/s | 732 GB/s | 900 GB/s |




Higher performance than expected from hardware capability

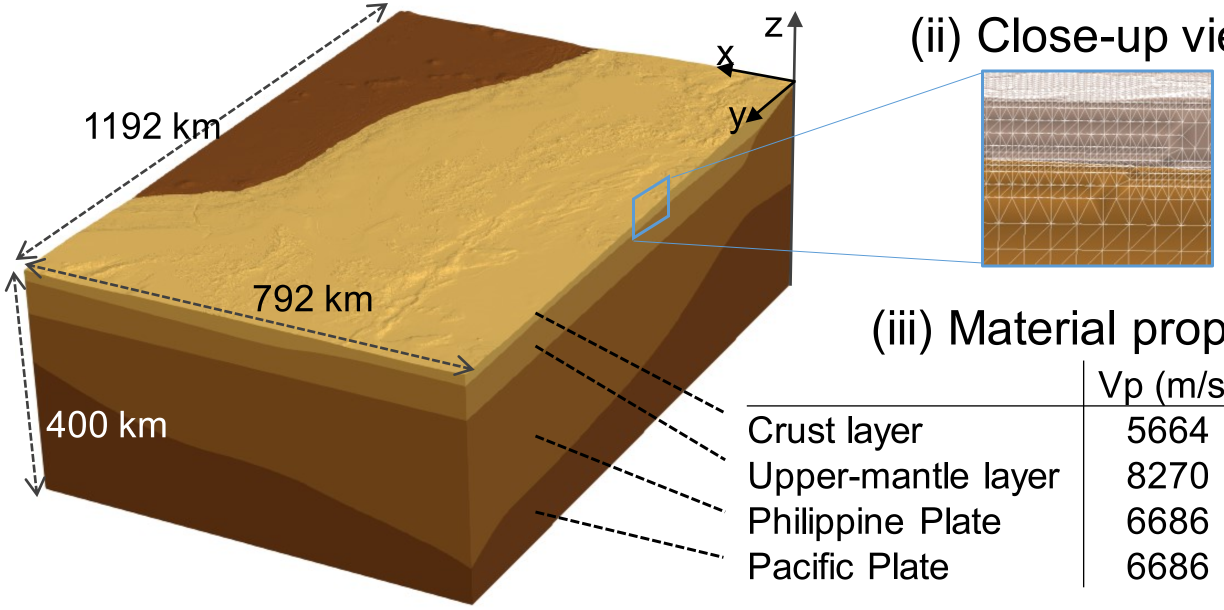
- improved L1/L2 caches

# Application Example

Estimation of coseismic slip distribution in 2011 Tohoku Earthquake
DOF: 409,649,580
# of input vectors: 368 = 23 sets × 16 vectors
Computation Environment: 64 x P100 GPUs (32 nodes of Reedbush-H)
Computation time: 828s for 23 sets of analyses
(29 times better in performance than previous studies)

(i) Whole FE model

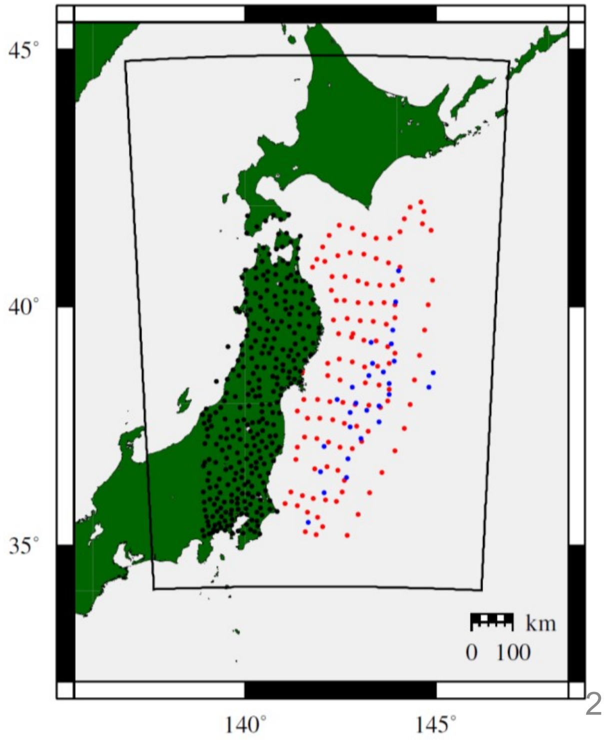
(ii) Close-up view




(iii) Material properties

| | Vp (m/s) | Vs (m/s) | ρ (kg/m$^3$) |
|---|---|---|---|
| Crust layer | 5664 | 3300 | 2670 |
| Upper-mantle layer | 8270 | 4535 | 3320 |
| Philippine Plate | 6686 | 3818 | 2600 |
| Pacific Plate | 6686 | 3818 | 2600 |

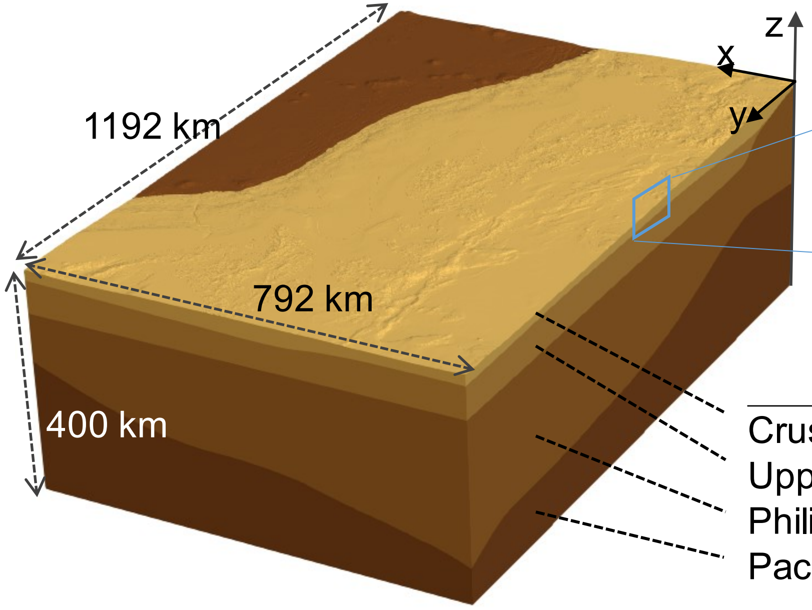
Target Domain



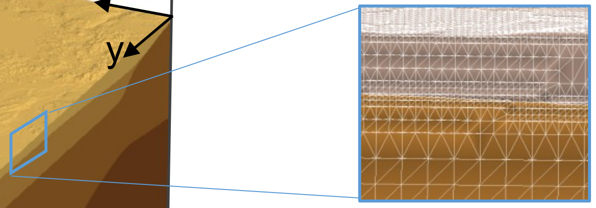
# Application Example

Estimation of coseismic slip distribution in 2011 Tohoku Earthquake
DOF: 409,649,580
\# of input vectors: 368 = 23 sets × 16 vectors
Computation Environment: 64 x P100 GPUs (32 nodes of Reedbush-H)
Computation time: 828s for 23 sets of analyses
(29 times better in performance than previous studies)

(i) Whole FE model



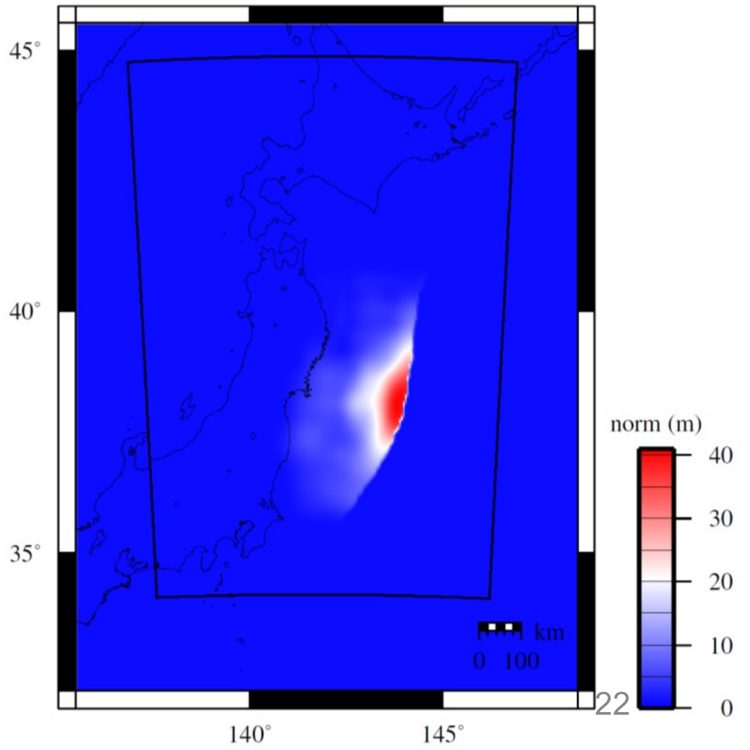

(ii) Close-up view



(iii) Material properties

| | Vp (m/s) | Vs (m/s) | ρ (kg/m$^3$) |
|---|---|---|---|
| Crust layer | 5664 | 3300 | 2670 |
| Upper-mantle layer | 8270 | 4535 | 3320 |
| Philippine Plate | 6686 | 3818 | 2600 |
| Pacific Plate | 6686 | 3818 | 2600 |

Estimated Distribution



# Conclusion

- Accelerate the unstructured low-order finite element solvers by OpenACC
  - Design the solver appropriate for GPU computations
  - Port the codes to GPUs
- Obtain high performance with low development costs
  - 14.2 times speedup on P100 GPUs from the original solver on CPU-based K computer
  - Computation in low power consumption
- Improvement in reliability of earthquake simulation
  - Many-case simulation within short time