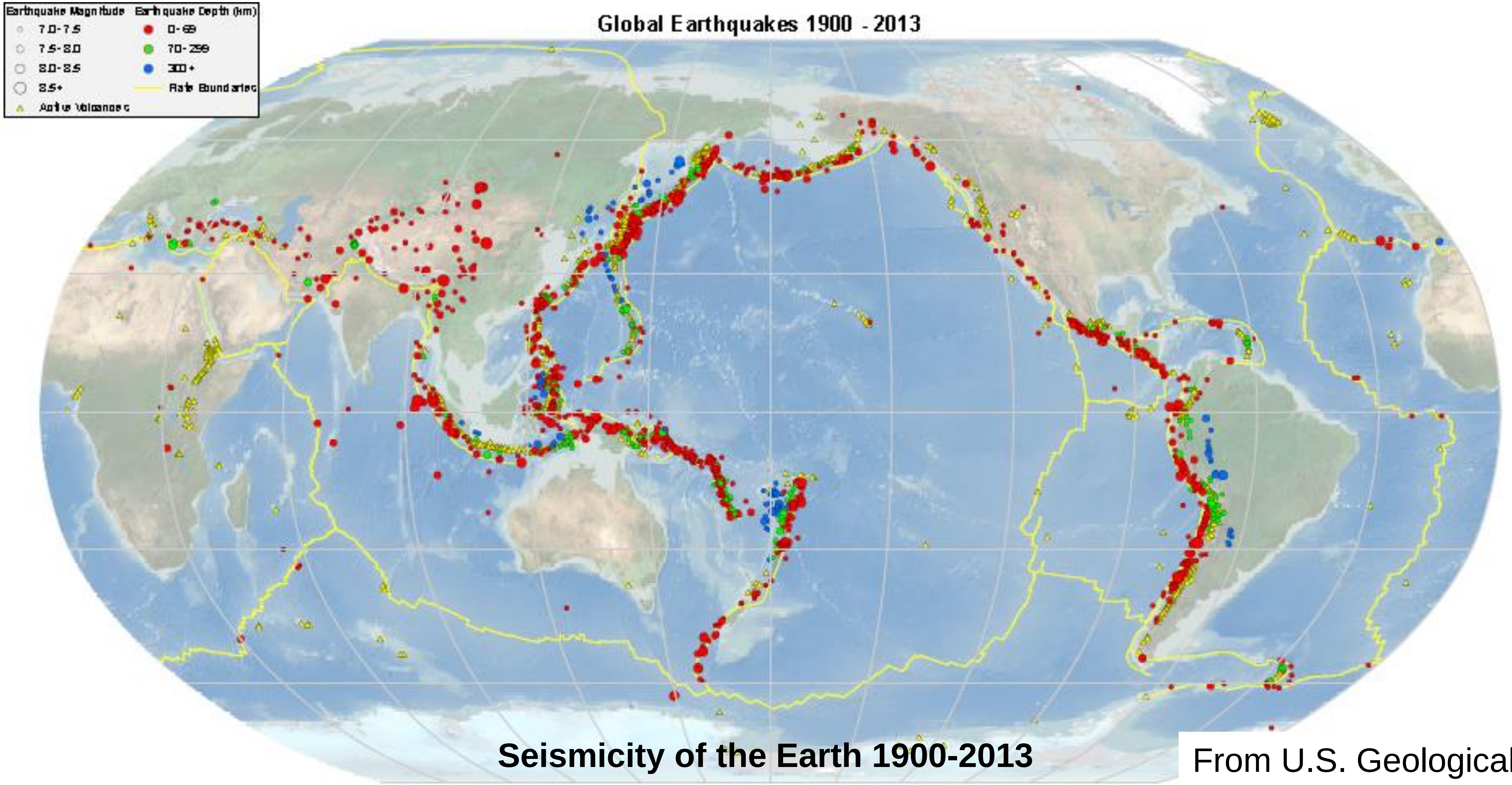
Third Workshop on Accelerator Programming Using Directives
(WACCPD2016), Nov. 14 2016

Acceleration of Element-by-Element Kernel in Unstructured Implicit Low-order Finite-element Earthquake Simulation using OpenACC on Pascal GPUs

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

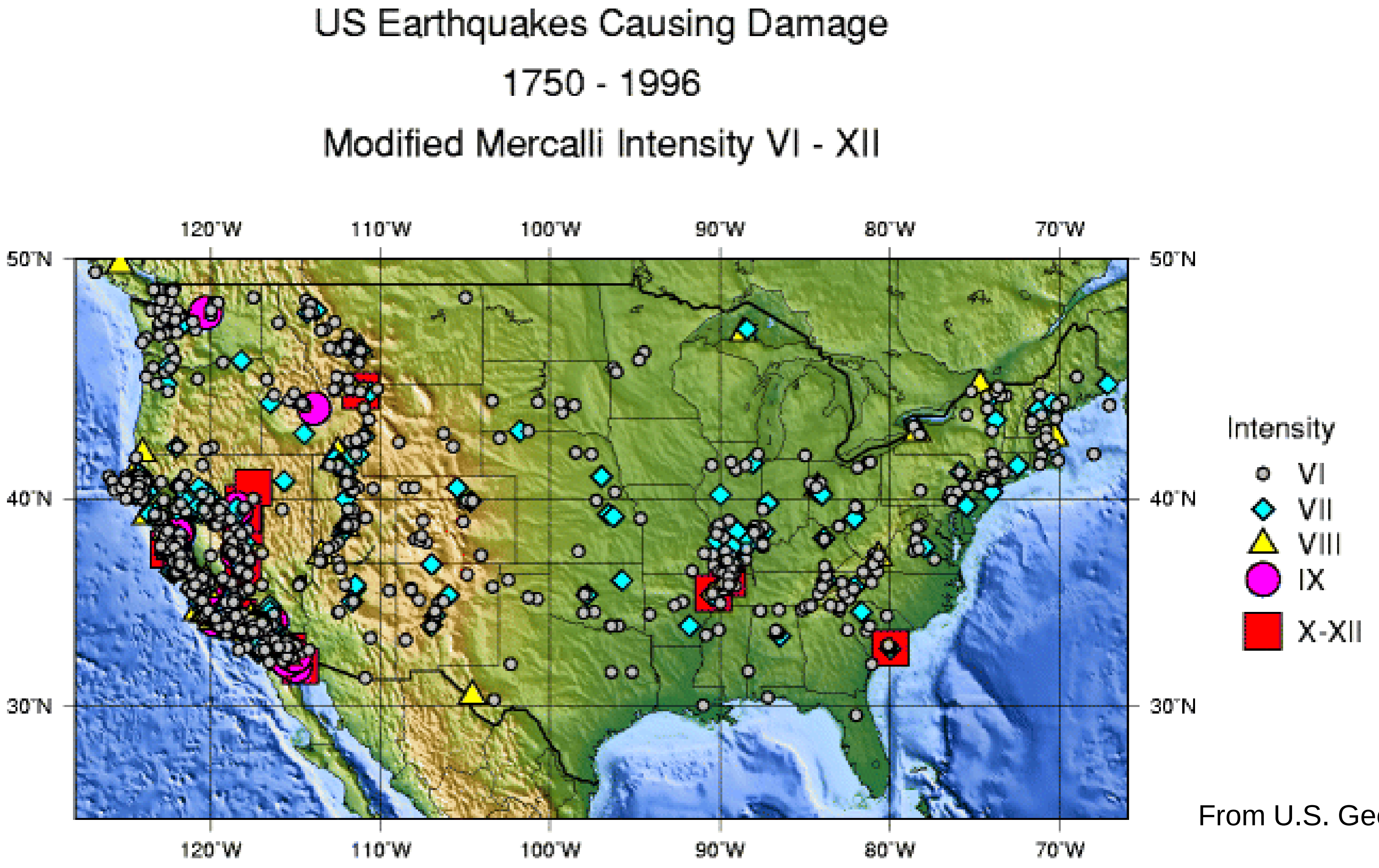
Many cities are prone to earthquakes



Seismicity of the Earth 1900-2013

From U.S. Geological Survey

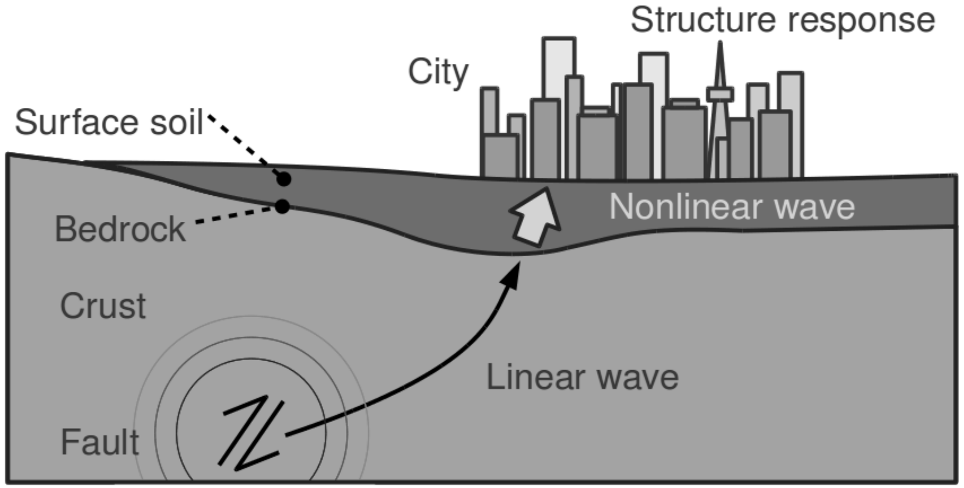
Many cities are prone to earthquakes



From U.S. Geological Survey

Introduction

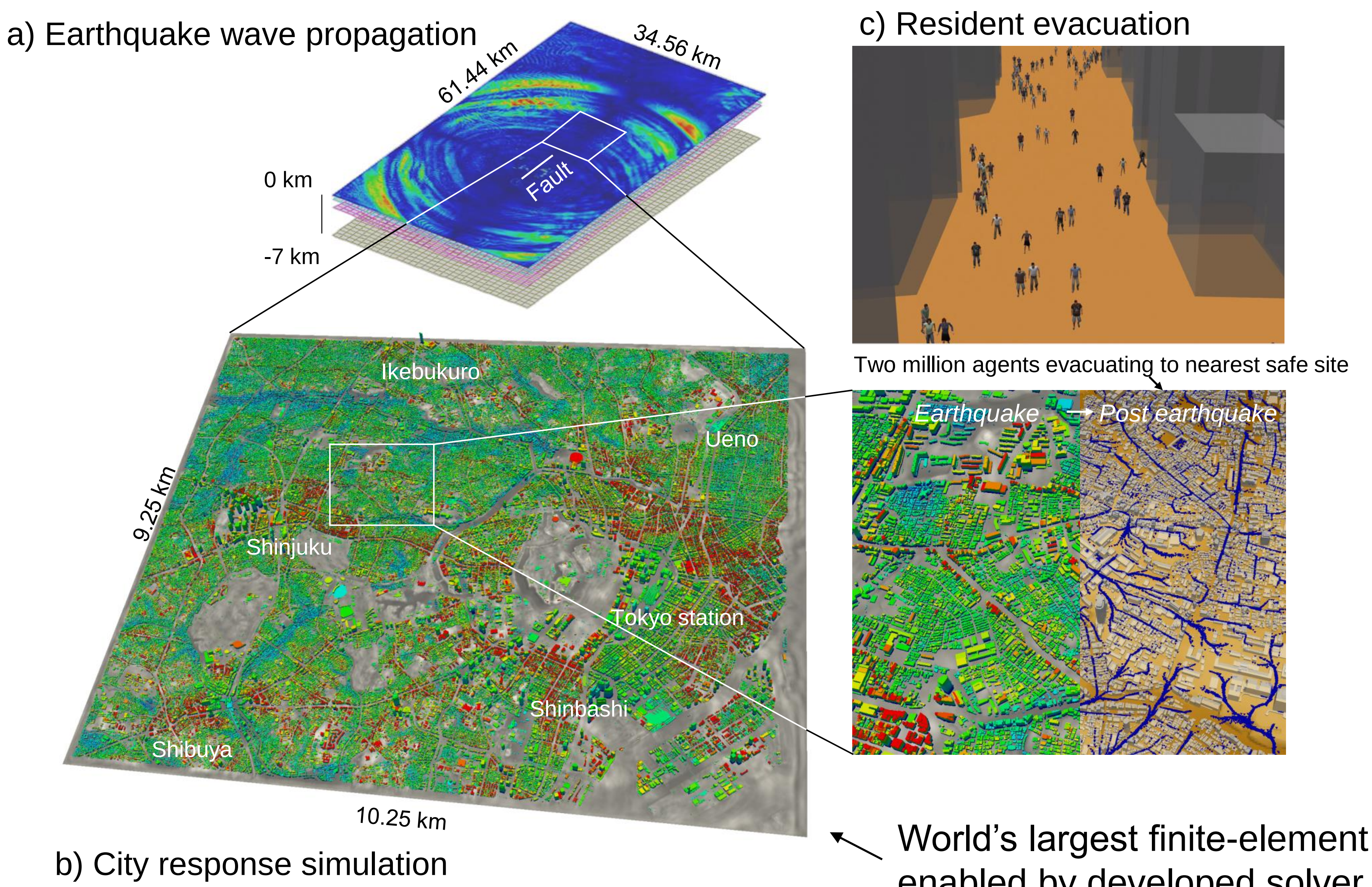
- Contribution of HPC to earthquake mitigation highly anticipated from society
- We are developing comprehensive earthquake simulation that simulate all phases of earthquake disaster by full use of K computer system
  - Simulate all phases of earthquake required by speeding up core solver
  - **Nominated for SC14 Gordon Bell Prize Finalist, SC15 Gordon Bell Prize Finalist & SC16 Best Poster Finalist**
- Today’s topic is porting this solver to GPU-CPU heterogeneous environment
  - Report performance on NVIDIA’s newest Pascal GPUs



Earthquake disaster process


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

Comprehensive earthquake simulation



World's largest finite-element simulation enabled by developed solver

City simulation

Visualized by CYBERNET SYSTEMS CO., LTD

Target problem

- Solve large matrix equation many times
  - Arises from unstructured finite-element analyses used in many components of comprehensive earthquake simulation
  - 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

$$Ku = f$$

Outer force vector

Unknown vector with 1 trillion degrees of freedom

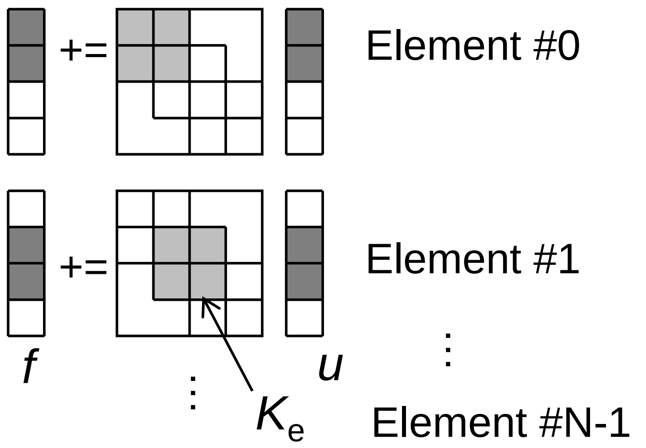
Sparse, symmetric positive definite matrix

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

Element-by-Element method

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



Designing scalable & fast finite-element solver

- Conjugate-Gradient method + Element-by-Element method + simple preconditioner
  - ➔ Scalability & peak-performance good, but poor convergency
  - ➔ Time-to-solution not good

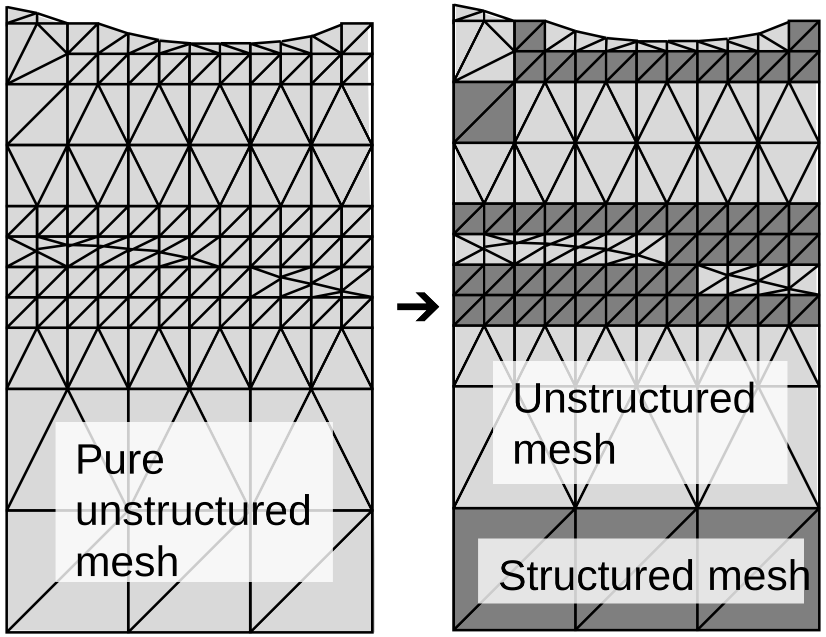
- Conjugate-Gradient method + sophisticated preconditioner
  - ➔ Convergency good, but scalability or peak-performance (sometimes both) not good
  - ➔ Time-to-solution not good

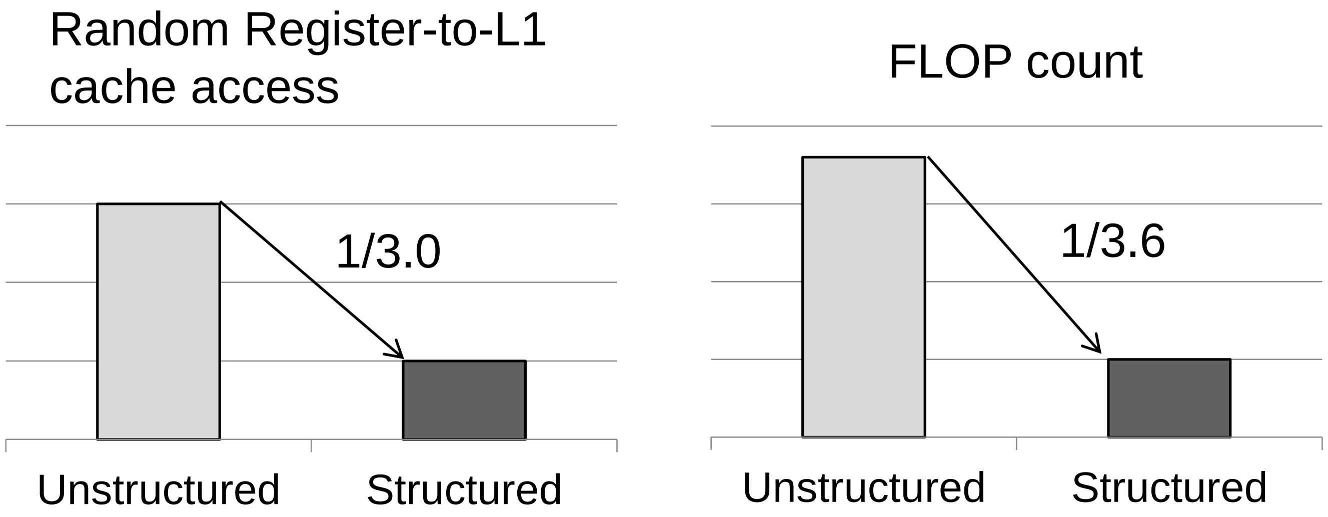
Designing scalable & fast finite-element solver

- Conjugate-Gradient method + Element-by-Element method + Multi-grid + Mixed-Precision + Adaptive preconditioner
  - ➔ Scalability & peak-performance good (all computation based on Element-by-Element), convergency good
  - ➔ Time-to-solution good
- Key to make this solver even faster:
  - Make Element-by-Element method super fast

Fast Element-by-Element method

- Element-by-Element method for unstructured mesh involves many random access & computation
  - Use structured mesh to reduce these costs
- Fast & scalable solver algorithm + fast Element-by-Element method
  - Enables very good scalability & peak-performance & convergency & time-to-solution on K computer
  - Nominated as Gordon Bell prize finalists for SC14 and SC15





Operation count for Element-by-Element kernel (linear elements)

Motivation & aim of this study

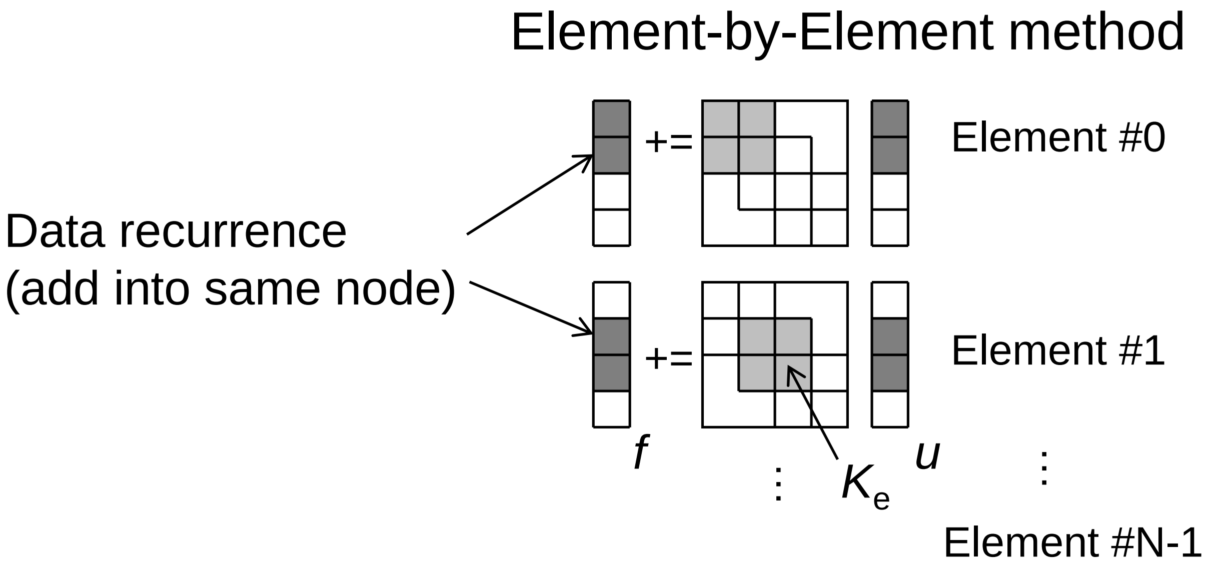
- Demand for conducting comprehensive earthquake simulations on variety of compute systems
  - Joint projects ongoing with government/companies for actual use in disaster mitigation
  - Users have access to different types of compute environment
- Advance in GPU accelerator systems
  - Improvement in compute capability & performance-per-watt
- We aim to port high-performance CPU based solver to GPU-CPU heterogeneous systems
  - Extend usability to wider range of compute systems & attain further speedup

Porting approach

- Same algorithm expected to be more effective on GPU-CPU heterogeneous systems
  - Use of mixed precision (most computation is done in single precision instead of double precision) more effective
  - Reducing random access by structured mesh more effective
- Developing high-performance Element-by-Element kernel for GPU becomes key for fast solver
- Our approach: attain high-performance with low porting cost
  - Directly port CPU code for simple kernels by OpenACC
  - Redesign algorithm of Element-by-Element kernel for GPU

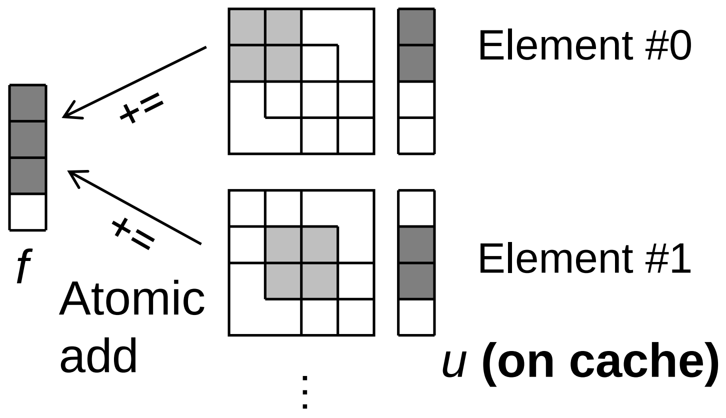
Element-by-Element kernel algorithm for CPUs

- Element-by-Element 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



Element-by-Element kernel algorithm for GPUs

- GPU: designed to hide latency by running many threads on 10^3 physical cores
  - Cannot allocate temporary buffers per thread on GPU memory
- Algorithm for adding up thread-wise results on GPUs
  - Coloring often used for previous GPUs
    - Algorithm independent of cache and atomics
  - Recent GPUs have improved cache and atomics
    - Using atomics expected to improve performance as data (*u*) can be reused on cache



Implementation of GPU computation

- OpenACC: Port to GPU by inserting a few directives
  - Parallelize
  - Atomically operate to avoid data race (atomic version)
  - Reduce CPU-GPU data transfer to the minimum
- Launch threads for the element loop *i*

a) Coloring add

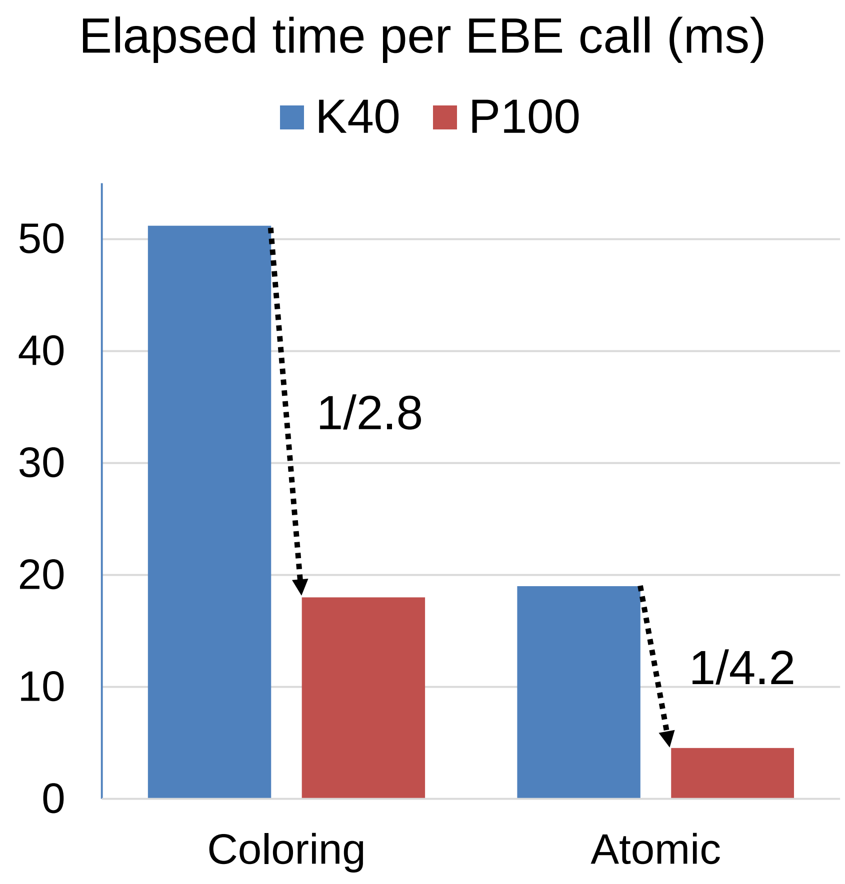
```
!$ACC DATA PRESENT(...)
    ...
    do icolor=1,ncolor
!$ACC PARALLEL LOOP
    do i=ns(icolor),ne(icolor)
     ! read arrays
     ...
     ! compute Ku
     Ku11=…
     Ku12=…
     ...
     ! add to global vector
     f(1,cny1)=Ku11+f(1,cny1)
     f(2,cny1)=Ku21+f(2,cny1)
        ...
     f(3,cny4)=Ku34+f(3,cny4)
    enddo
    enddo
!$ACC END DATA
```

b) Atomic add

```
!$ACC DATA PRESENT(...)
    …
!$ACC PARALLEL LOOP
  do i=1,ne
    ! read arrays
    ...
    ! compute Ku
    Ku11=…
    Ku12=…
    ...
    ! add to global vector
    !$ACC ATOMIC
    f(1,cny1)=Ku11+f(1,cny1)
    !$ACC ATOMIC
    f(2,cny1)=Ku21+f(2,cny1)
       ...
    !$ACC ATOMIC
    f(3,cny4)=Ku34+f(3,cny4)
  enddo
!$ACC END DATA
```

Comparison of algorithms

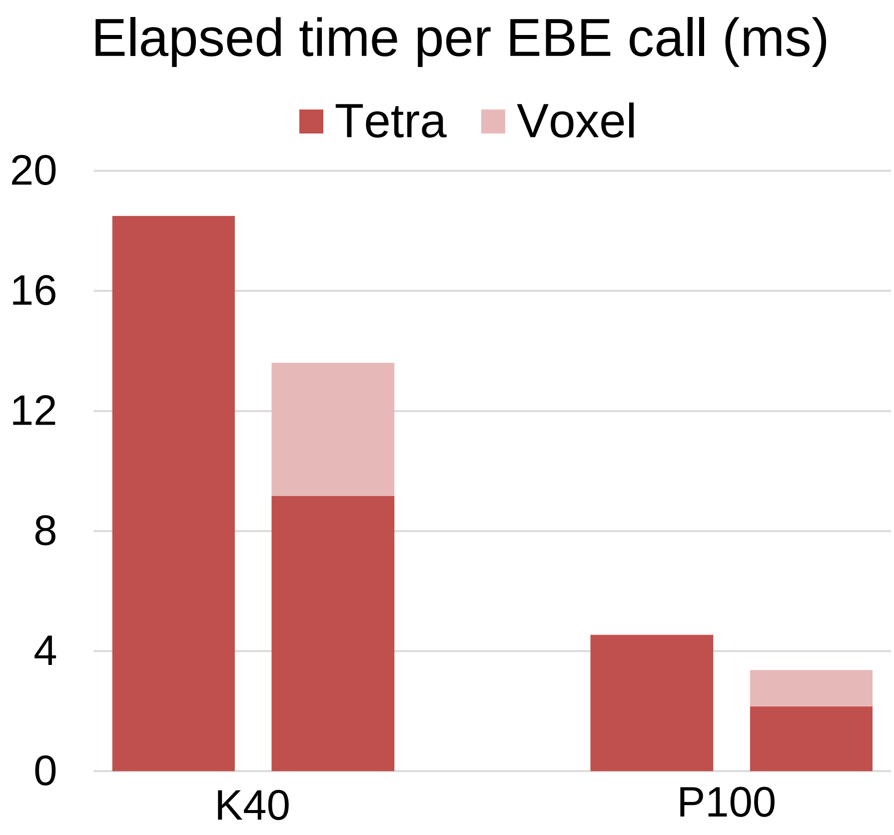
- Coloring and Atomics
  - With pure unstructured computation
  - NVIDIA K40 and P100 with OpenACC
    - K40: 4.29 TFLOPS (SP)
    - P100: 10.6 TFLOPS (SP)
  - 10,427,823 DOF and 2,519,867 elements
- Atomics is faster algorithm
  - High data locality and enhanced atomic function
  - P100 shows better speedup
  - Similar performance in CUDA



Performance in structured computation

- Effectiveness of mixed structured/unstructured computation
  - With mixed structured/unstructured computation
  - K40 and P100
  - 2,519,867 tetrahedral elements
    ➔ 204,185 voxels and 1,294,757 tetrahedral elements

- 1.81 times speedup in structured computation part

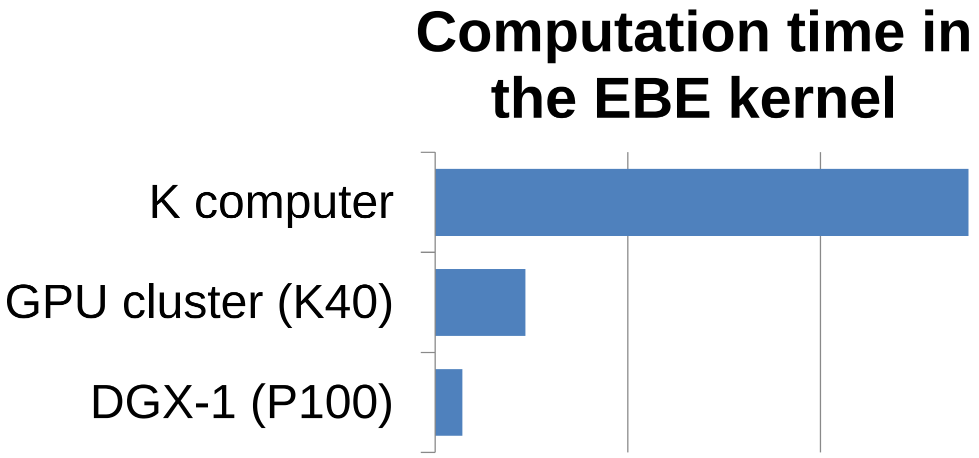


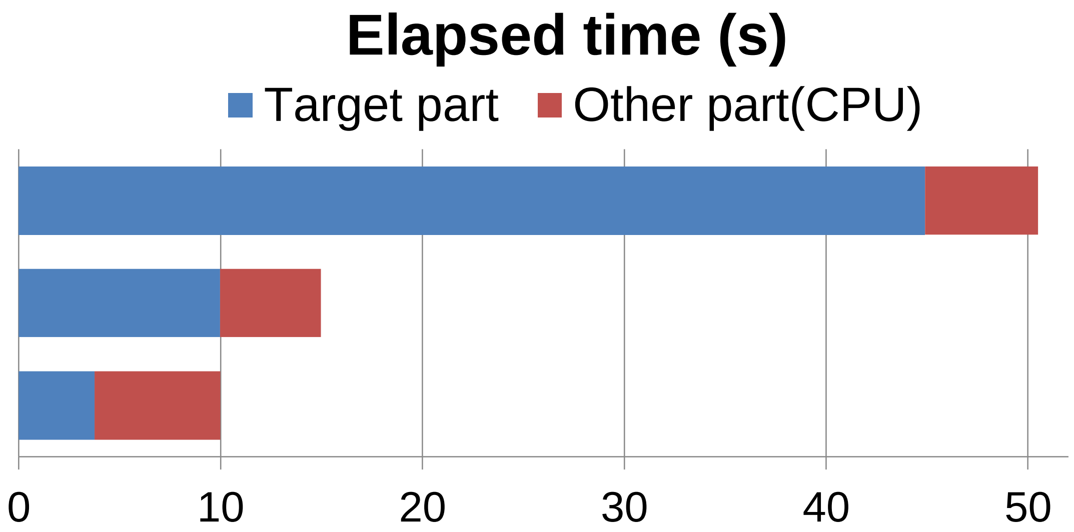
Performance in the solver

- 82,196,106 DOF and 19,921,530 elements

	# of nodes	CPU/node	GPU/node	Hardware peak FLOPS	Memory bandwidth
K computer	8	1 x SPARC64 IIIfx	-	1.02 TFLOPS	512 GB/s
GPU cluster	8	2 x Xeon E5-2695 v2	1 x K40	34.3 TFLOPS	2.30 TB/s
NVIDIA DGX-1	1	2 x Xeon E5-2698 v4	8 x P100	84.8 TFLOPS	5.76 TB/s

- 19.6 times speedup for DGX-1 in the EBE kernel





Conclusion

- Accelerate the EBE kernel on unstructured implicit low-order finite element solvers by OpenACC
  - Design the solver that attains equal granularity at many cores
  - Port GPUs to the key kernel
- Obtain high performance with low development costs
  - Computation in low power consumption
  - Many-case simulation within short time
- Expect good performance
  - With larger GPU-based arichitectures (100 million DOF per P100)
  - In other finite-element simulations