The Plastic Domain Decomposition

Boris Jeremić

Department of Civil and Environmental Engineering
University of California, Davis

MRCCS/NSF Summer School
High Performance Computing in Finite Element Analysis
1st - 5th September 2003,
University of Manchester

Supported in part by the NSF, PEER, Caltrans, and Cal–EPA.

Collaborators: Professors Zhaohui Yang (UAA), Sashi Kunnath (UCD), Gregory Fenves (UCB), Jacobo Bielak (CMU), Zhaojun Bai (UCD), George Karypis (UMN), Drs. Francis McKenna (UCB), Ingrid Hotz (UCD), and graduate students Xiaoyan Wu (UCD) Ritu Jain (UCD), Zhao Cheng (UCD), Kallol Sett (UCD), Qing Liu (UCD), Jinxiu Liao (UCD). Guanzhou Jie (UCD).

Motivation

- Create high fidelity models of constructed facilities (bridges, buildings, port structures, dams...).
- Models will live concurrently with the physical system they represent.
- Models to provide owners and operators with the capabilities to assess operations and future performance.
- Use observed performance to update and validate models through simulations.

Large Scale Numerical Simulation Goal

- Scalable parallel finite element method for inelastic computations (solid and structural elements)
- Based on state of the art computational mechanics theories and implementation
- Available for a range of sequential and parallel machines, including clusters, grids of machines and clusters (DMPs), and also multiprocessors (SMPs)
- Public domain, portable platform.

Incomplete Historical Background

- Substructuring to achieve partitions (Noor et al. [5], Utku et al. [7] and Storaasil and Bergan [6])
- Techniques to account for different types of elements but used substructures of same element types (non–balanced computations) (Fulton and Su [2]).
- dynamic analysis of framed structures with the objective of minimizing communications (Hajjar and Abel [3])
- parallel computational techniques for elastic–plastic problems but tied the algorithm to the specific multiprocessor computers used (and specific network connectivity architecture) (Klaas et al. [4])
- Greedy domain partitioning algorithm good for topological DD but does not redistributed the domains as a function of developed nonlinearities (Farhat [1])

Brief Methodology Background

- Static domain decomposition will lead to optimal parallel run for many problems
  - equal number of mesh elements to each processor, this will balance computational load (CL) on parallel machine,
  - minimize the size of subdomain boundaries, this will minimize the inter-processor communications overhead,
- This can be done many ways, recently a set of good graphs partitioner were developed (METIS family by Karypis et al.)
- This is indeed good for problems where the computational domain or discretization does not change.
- For inelastic computations using finite element method, the change will occur in internal state determination.

Inelastic Parallel Problem

- Presence of elastic and inelastic (plasticity, damage) computational domains.
- Difference in computational load for elastic and inelastic state determination leads to computational load imbalance
- This leads to imbalanced computations, very inefficient, not much gain from using parallelization
- Internal state determination can take as much as 80% of CL

Inelastic Computational Load Balancing Challenge: Adaptive Computations

- Dynamic computations, structure of elastic and elastic–plastic domains changes dynamically and unpredictably
- Periodic computational load-balancing is required during the course of the computation
- Computational load balancing cost similar to static DD (balance the mesh elements and minimize the inter-processor communications
- It also requires that the cost associated with redistributing the data in order to balance the computational load is minimized

Inelastic Computational Load Balancing Challenge: Multi-phase Computation

- Elastic–plastic computations follow up the elastic computations, there is a synchronization step between
- Each phase needs to be separately computational load balanced
  - Elastic phase computations (topological DD might be OK)
  - Elastic–plastic phase computations (PDD)
- System of equations needs to be CL balanced as well

Elastic Computations

$$ {}^{(m)}K_{IacJ} = \int_{V^m} H_{I,b}\, E_{abcd}\, H_{J,d}\, dV^m $$

$$ E_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu \left(\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk} \right) $$

Lamé (elastic) coefficients:

$$ \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \quad ; \quad \mu = \frac{E}{2(1+\nu)} $$

Applies to linear and nonlinear elasticity

Elastic–Plastic Computations

- Integration of constitutive equations at each integration point:
  - Explicit integration (Forward Euler (FE) algorithm)
  - Implicit integration (Backward Euler (BE) algorithm)
  - Mid–point integration (Crank–Nickolson (CN) algorithm)
- Single step (FE) and iterative algorithms (BE, CN)
- Elastic computations, computational load per element (integration point) known a priori
- Elastic–Plastic computations, computational load much larger
- Computations Load is not known a priori (the extent of plastic zone is not known prior to analysis)

Implicit Constitutive Integration

Iterative procedure, unknown number of steps

$$\Delta\sigma_{mn} = -\Big({}^{old}r_{ij} + \frac{{}^{n+1}F^{old} - {}^{n+1}n_{mn}\ {}^{old}r_{ij}\ {}^{n+1}T^{-1}_{ijmn}}{{}^{n+1}n_{mn}E_{ijkl}\ {}^{n+1}H_{kl}\ {}^{n+1}T^{-1}_{ijmn} - {}^{n+1}\xi_*\ h_*}\ E_{ijkl}\ {}^{n+1}H_{kl}\Big)\ {}^{n+1}T^{-1}_{ijmn}$$

$$\Delta q_* = \left(\frac{{}^{n+1}F^{old} - {}^{n+1}n_{mn}\ {}^{old}r_{ij}\ {}^{n+1}T^{-1}_{ijmn}}{{}^{n+1}n_{mn}E_{ijkl}\ {}^{n+1}H_{kl}\ {}^{n+1}T^{-1}_{ijmn} - {}^{n+1}\xi_*\ h_*}\right) h_*$$

$${}^{n+1}T_{ijmn} = \delta_{im}\delta_{nj} + \lambda\ E_{ijkl}\ \left.\frac{\partial m_{kl}}{\partial\sigma_{mn}}\right|_{n+1}\ ;\ {}^{n+1}H_{kl} = {}^{n+1}m_{kl} + \lambda\left.\frac{\partial m_{kl}}{\partial q_*}\right|_{n+1} h_*$$

$$n_{ij} = \frac{\partial F}{\partial\sigma_{ij}}\ ;\ m_{ij} = \frac{\partial Q}{\partial\sigma_{ij}}\ ;\ \xi_* = \frac{\partial F}{\partial q_*}$$

Computational Load Varies

- Load per integration point for each iteration/increment (4 CPU cycles (CPUc) for addition, 6 CPUc for multiplication, 13 CPUc for division...):
  - Elastic computation: approx. 3600 CPUc
  - Forward Euler single step: approx. 7000 CPUc
  - Backward Euler single step: approx. $12000 + n \times 15000$ CPUc
- Example #1: say 5 iteration steps $\rightarrow 87000$ CPUc (24 times more CPU load)
- Example #2: say 25 iteration steps $\rightarrow 387000$ CPUc (108 times more CPU load)
- This is all load per one integration point!
- Standard 20 node brick element should have at least 27 integration points $(3 \times 3 \times 3)$

Computational Load Imbalance

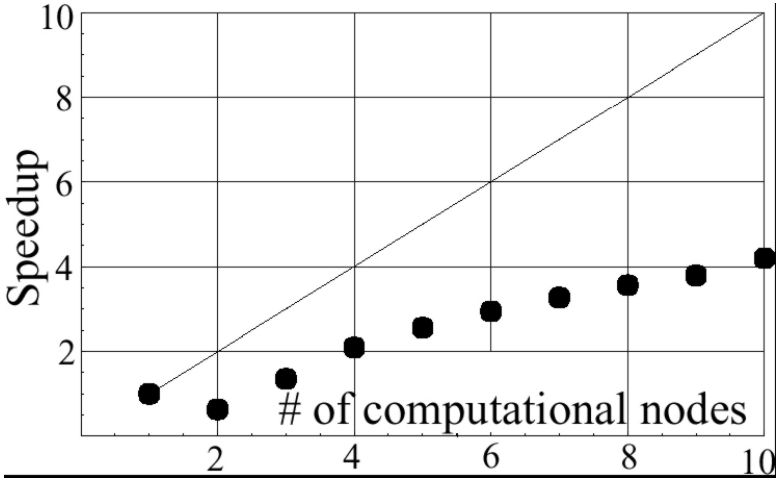
- Existing parallel computational algorithms: topological(preprocessing) DD into substructures with equal number of elements
- Topological DD assumes equal computational load per element
- Topological DD performed during a pre–processing phase
- No provisions for variation of computational load per element or for variation in connectivity speed (latency and bandwidth)
- Separate problem for internal state determination (elastic–plastic computations on the element level) and for the system of equations solution

Possible Solutions

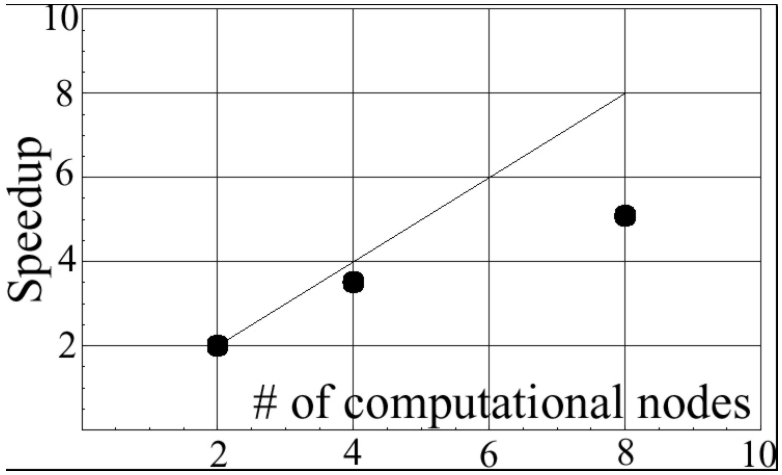
- Perform pre–processing stage DD, topological DD (equal computational load per element, pre–processing phase, no provisions for: variation of computational load per element, connectivity speed, latency and bandwidth)
- Distribute small subdomains to processor as they become idle, feed hungry CPUs (dynamic CL by design, expensive as it sends a lot of "small" data packets, internal state determination and system solution phases are totally separate, a lot of data moving around)
- Element by element family of methods in conjunction with iterative equation solvers
- Dynamic CL balancing $\rightarrow$ Plastic Domain Decomposition (currently being implemented into OpenSees)

Feed Hungry CPUs

- Distribute computations of element matrices to parallel nodes in small groups
- Solve the system in parallel using MP_SOLVE
- Both phases separate



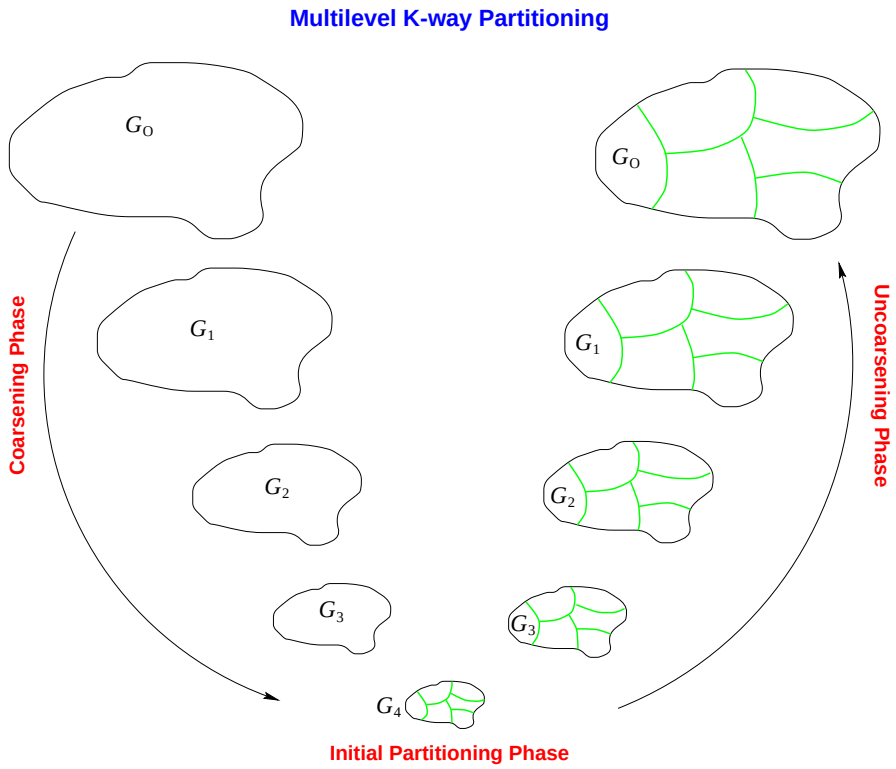
stiffness matrix formations



system solution

Development of the Plastic Domain Decomposition

- Based on work of Karypis et al.
- Multilevel Graph Partitioning
  - graph coarsening
  - initial partitioning
  - multilevel refinement
- Par–METIS system
- Weighted graphs edges and graph nodes
- Zoltan framework used as an interface layer for extensibility



Adaptive Graph Partitioning

- Dynamic computational load-balancing algorithms, based on the multilevel graph partitioning paradigm

- Attempt to minimize the data redistribution costs using one of the two methods:
  - compute a new partitioning from scratch and then intelligently map this back to the original partitioning
  - Perturb the original partitioning just enough so as to balance the computational load

Adaptive Graph Partitioning I

Compute a new partitioning from scratch and then intelligently map this back to the original partitioning:

- result in partitioning that do very well at minimizing the inter-processor communications costs,
- because when a new partitioning is computed from scratch, we can use a state-of-the-art partitioning method to do so,
- results in higher data redistribution costs compared to other methods,
- especially when the partitioning is only slightly imbalanced
- because it is often the case that the newly computed partitioning is substantially different from the original partitioning
- so even a good remapping of the new partitioning can still incur large data redistribution costs

Adaptive Graph Partitioning II

Perturb the original partitioning just enough so as to balance the computational load:

- extremely low data redistribution costs,
- result in higher inter-processor communications costs for highly imbalanced partitioning,
- because during the process of balancing the partitioning, locally optimum selections are made and the more that partitioning needs to be perturbed in order to balance it, the greater the likelihood that the effect of the locally optimum decisions that are made to balance the partitioning will be globally sub-optimal

Multi-phase Graph Partitioning

- Traditional graph partitioners CL balance only single phase
- Other phases might be seriously imbalanced
- Generalized formulation of the graph partitioning problem that is able to balance multiple phases simultaneously, while also minimizing the inter-processor communications costs
- Think of adaptive partitioning as a multi-objective optimization problem, that is minimize both the inter-processor communications, the data redistribution costs and create good partitions (CL balanced with minimal boundary)
- Unified adaptive partitioning–repartitioning algorithms
- Compute decompositions that take into account the relative costs of performing inter-processor communications and data redistributions

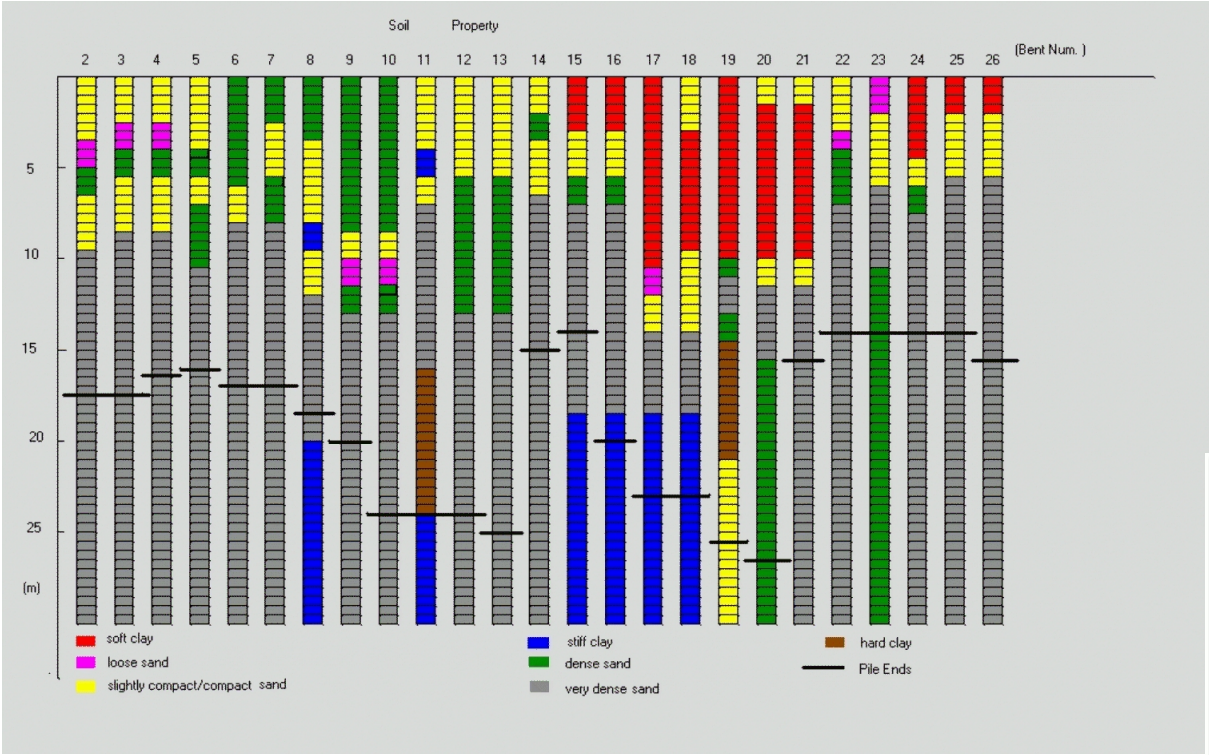
Plastic Domain Decomposition

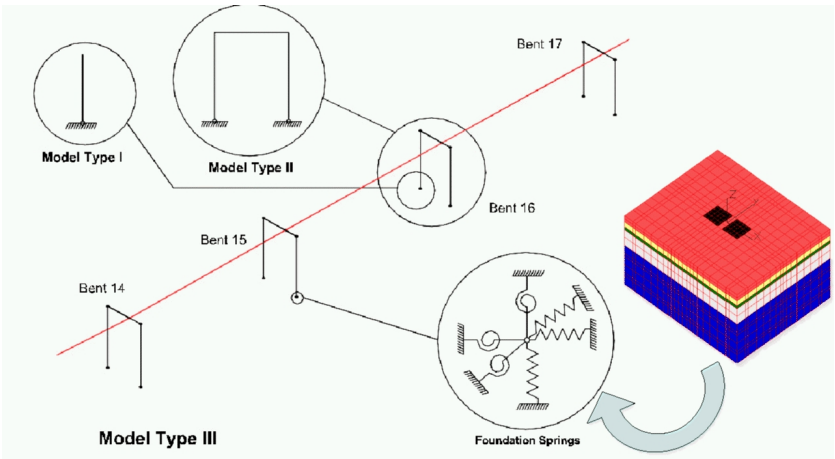
- Take into the account:
  - heterogeneous element loads that change in each iteration
  - heterogeneous processor performance (multiple generations nodes)
  - inter-processor communications
  - data redistribution costs
- Perform global optimization for both internal state determination and system solution phases
- Available for all elements (solid, structural) that provide the interface (sendSelf, RecvSelf, timer or CL weight estimate)
- Able to handle SMPs, local clusters, grids of computers

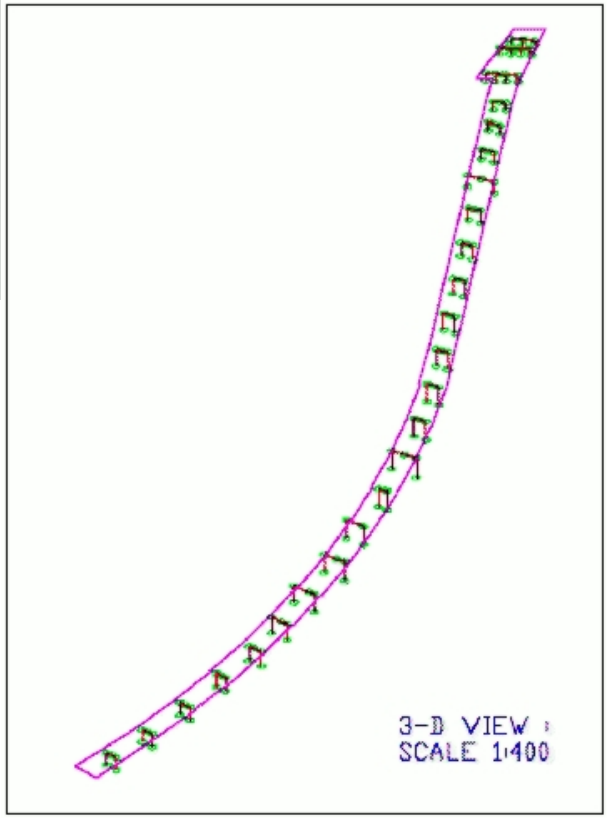
PDD Implementation

- Internal state determination:
  - Use lightweight measurement of each element performance (or provided information on CL)
  - Use lightweight measurement of network latency and bandwidth (including local are and wide area networks)
  - Use information from previous incremental step to make PDD decisions
  - Question: do we perform PDD for each iteration (or after few iterations) or for each increment (or after few increments)
- System of equations solution (SES):
  - Use previous decomposition as much as possible
  - might be somewhat imbalanced but it is OK if SES takes only 20% of time

PDD Goals: SFSI

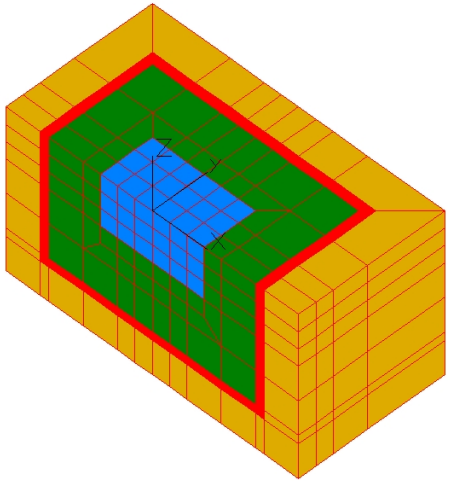


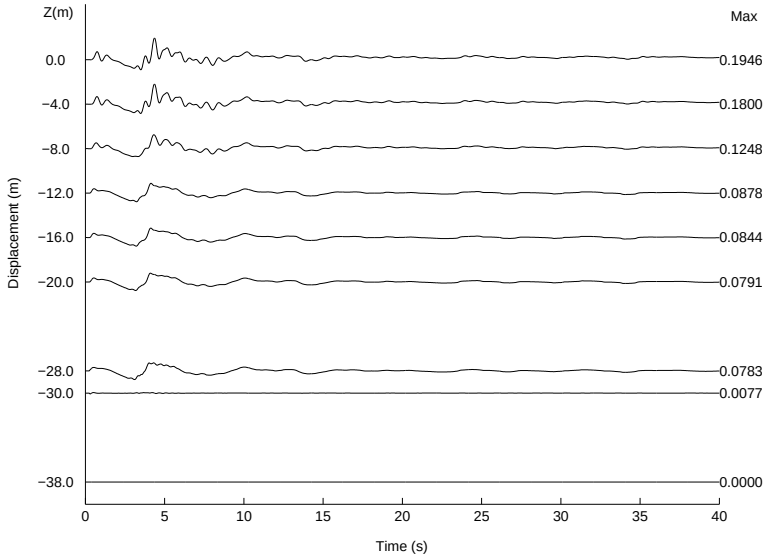




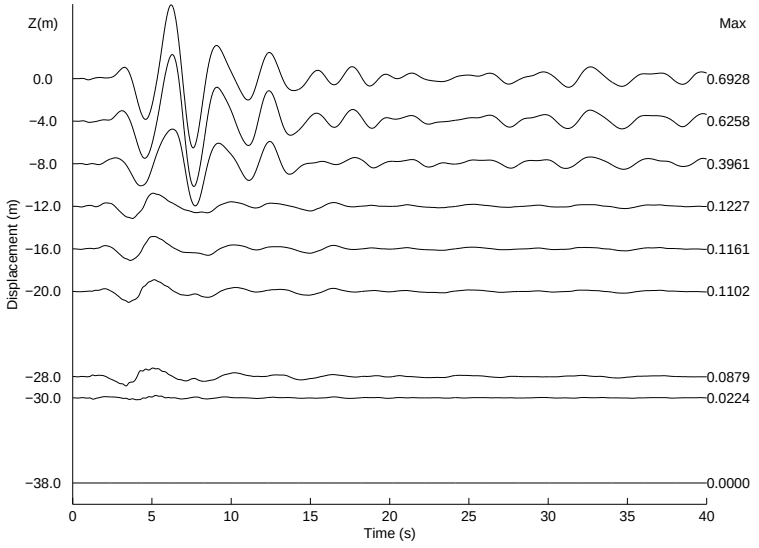
PDD Working Example

Pile, column and a mass on top





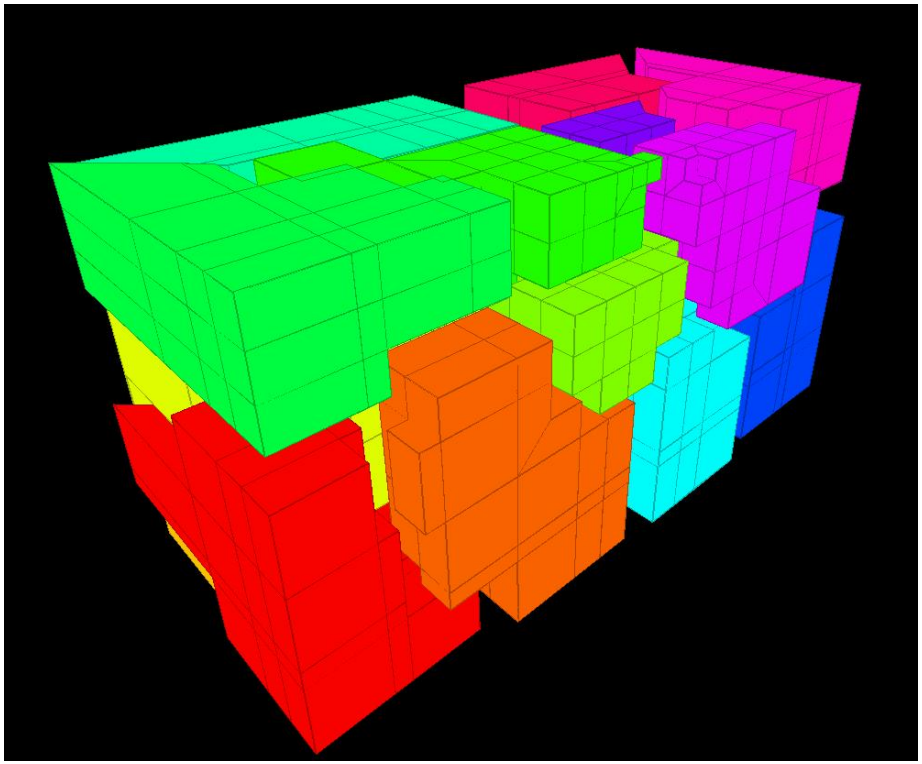
Stiff soil



Soft soil

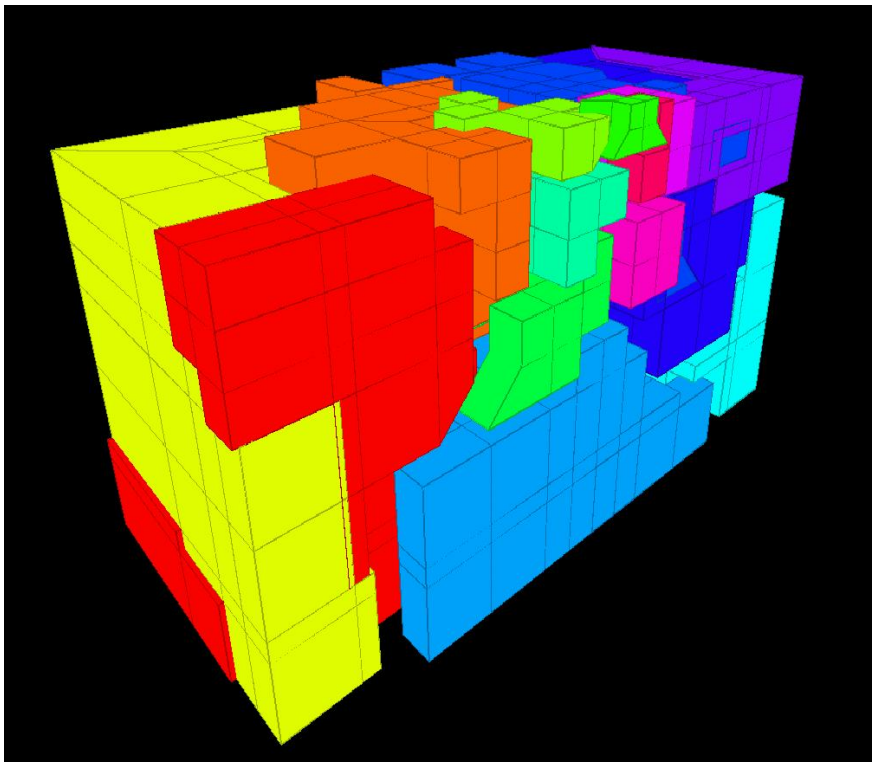
PDD: Elastic Decomposition

- 16 subdomains,
- approx. equal number of elastic elements per domain,
- minimized inter–domain boundary
- OpenSees $\Rightarrow$ Zoltan $\Rightarrow$ Par–METIS



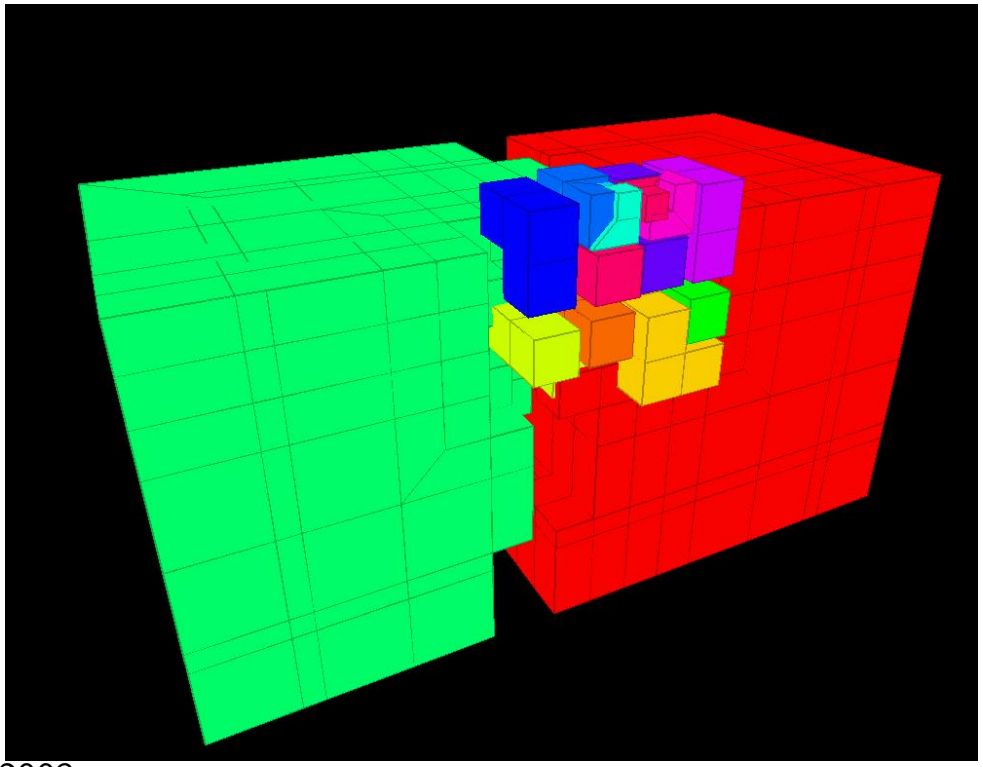
PDD: Mild Plasticity

- Small amount of plastic elements close to the pile
- Note smaller domain close to the pile
- Note also somewhat increased inter–domain boundary

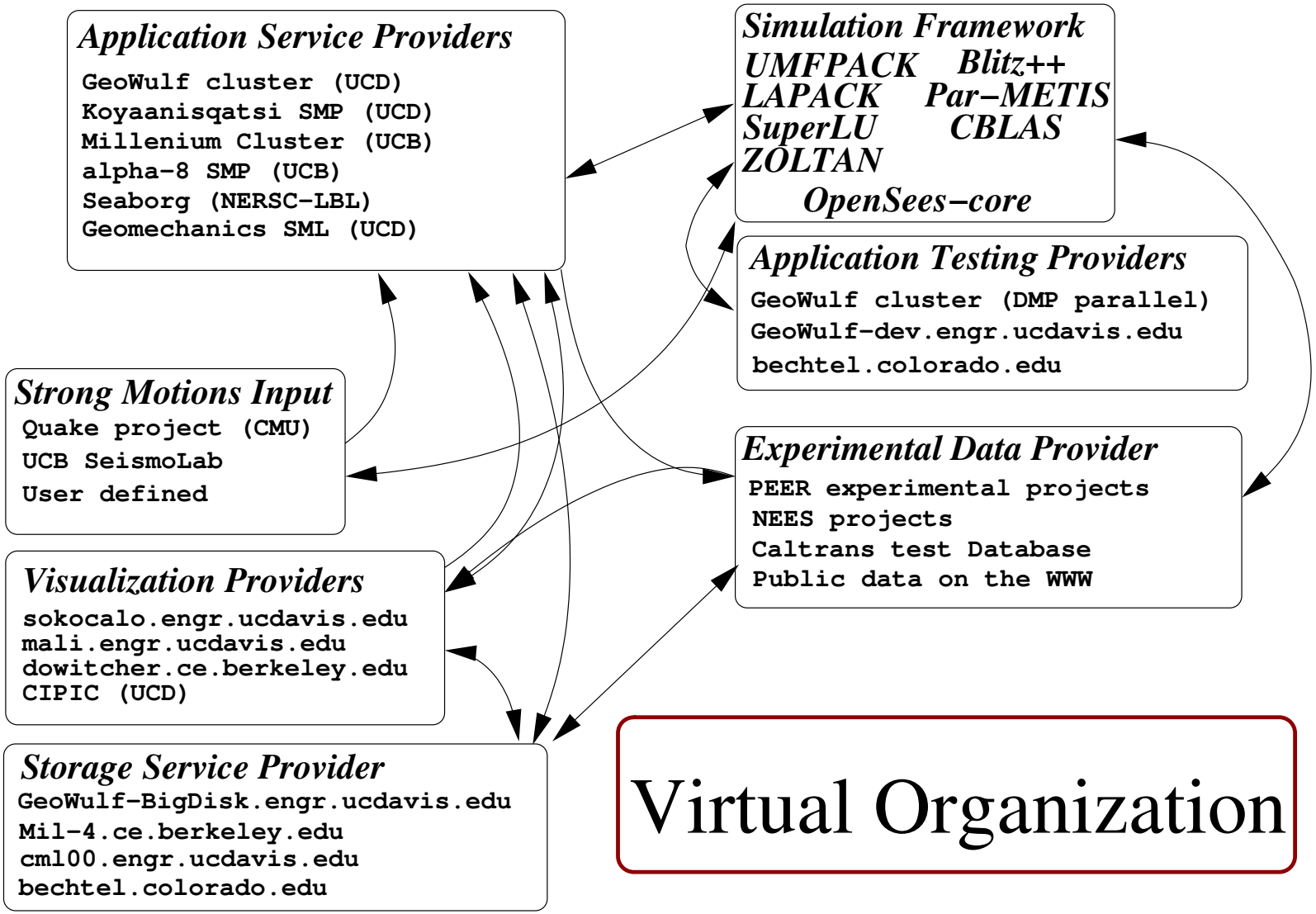


PDD: Severe Plasticity

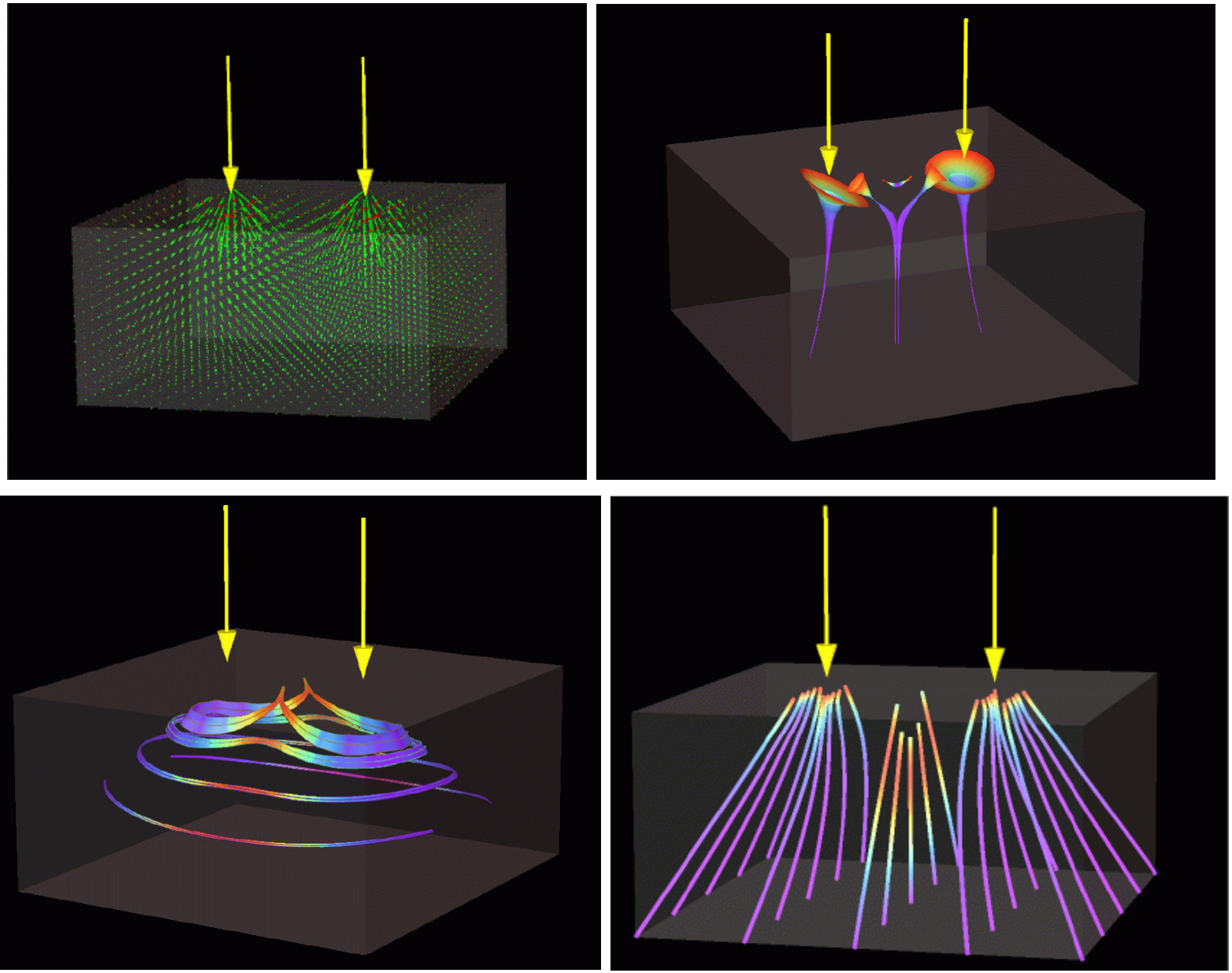
- Note small number of elements per CL heavy domains
- Note also two large, mostly elastic, subdomains
- Good DD for internal state determination, but bad for SES
- Use graph–partitioner to optimize SES phase using info on imbalance and network speed



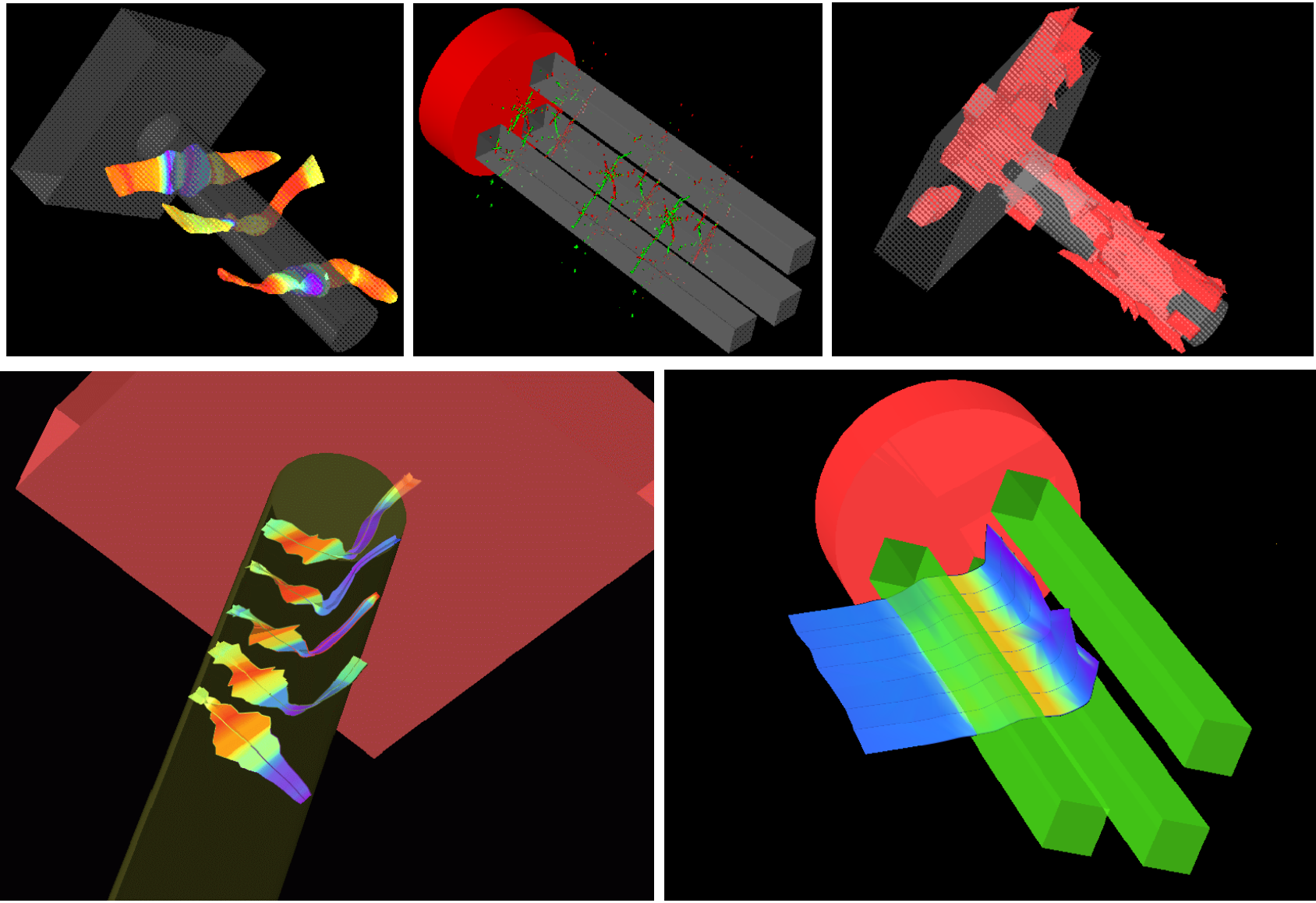
Current Grid Resources



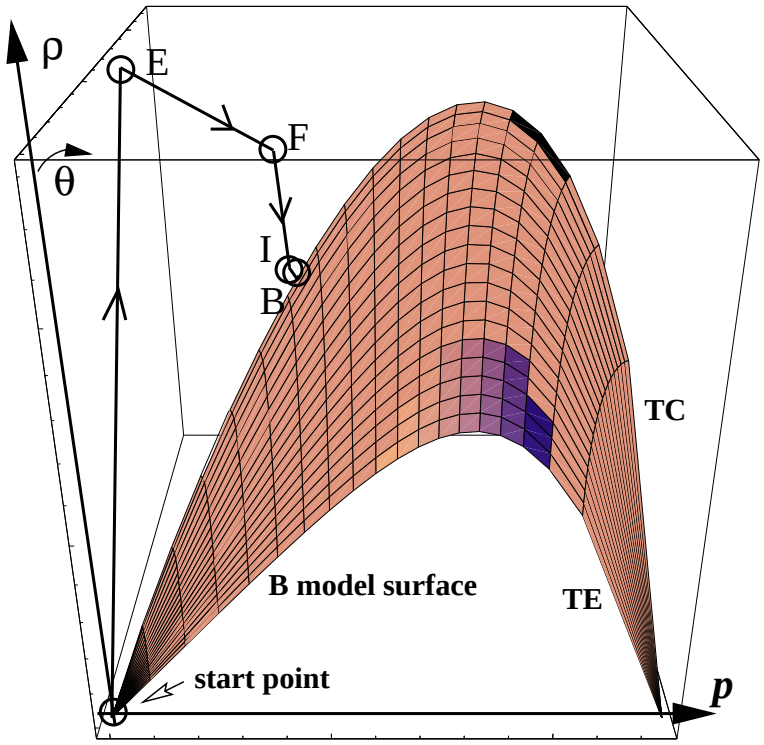
Stress Field Visualization



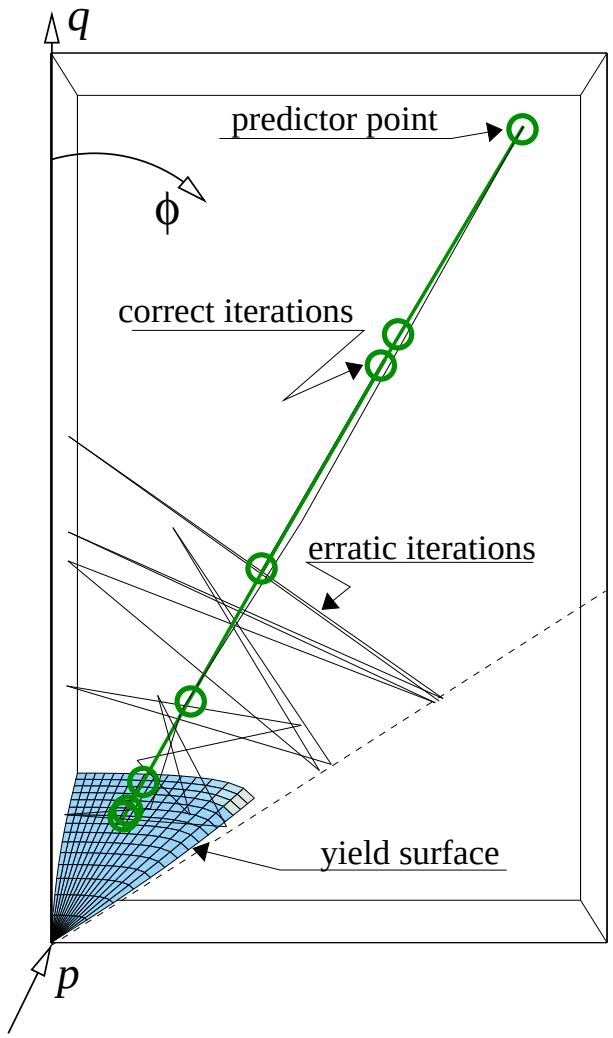
Pile Group Visualization



Constitutive Integration Visual Debugging



E: Elastic predictor
F: Forward Euler, elastoplastic corrector
I: Iterative steps
B: Backward Euler



Conclusions

- Plastic Domain Decomposition for tightly and loosely connected processors
- Can handle both computational load imbalance, heterogeneous processors and heterogeneous connectivity speeds
- Use of Par–METIS, Zoltan (and all that interfaces with Zoltan), Super–LU, MP_SOLVE
- Public domain within OpenSees framework

References

[1] Farhat, C. *Muliprocessors in Computational Mechanics*. PhD thesis, University of California, Berkeley, 1987.

[2] Fulton, R. E., and Su, P. S. Parallel substructure approach for massively parallel computers. *Computers in Engineering 2* (1992), 75–82.

[3] Hajjar, J. F., and Abel, J. F. Parallel processing for transient nonlinear strcutraul dynamics of three–dimensional framed structures using domain decomposition. *Computers & Structures 30*, 6 (1988), 1237–1254.

[4] Klaas, O., Kreienmeyer, M., and Stein, E. Elastoplastic finite element anslysis on a MIMD parallel–computer. *Engineering Computations 11* (1994), 347–355.

[5] Noor, A. K., Kamel, A., and Fulton, R. E. Substructuring techniques – status and projections. *Computers & Structures 8* (1978), 628–632.

[6] Storaasli, O. O., and Bergan, P. Nonlinear substructuring method for concurrent processing computers. *AIAA Journal 25*, 6 (1987), 871–876.

[7] Utku, S., Melosh, R., Islam, M., and Salama, M. On nonlinear finite element analysis in single– multi– and parallel processors. *Computers & Structures 15*, 1 (1982), 39–47.