Data and database requirements for the meta-partitioner

Jaideep Ray
Sandia National Labs, Livermore, CA
jairay@somnet.sandia.gov

Why do we need a database?

- The meta-partitioner is really a mechanism that chooses the correct partitioner settings {P}, given a problem (as characterized by a hierarchical mesh) which is represented by a set of grid parameters {G}
- The meta-partitioner identifies a mapping between {G} and a small collection (of size M) of the best partitioner settings $\{P\}_{best,i=1..M.}$ It then tries to establish which particular setting is the best.
- In order to find out what $\{P\}_{best}$ may be from a host of {P}, we need to do experiments (each with a different{P} & {G})and store each of the outcomes {O}. {O} is a set of performance metrics.
  - These are stored in a database.
  - We would like to cluster them and make a model.
- The set { {P}, {G}, {O} } forms one database record

What are {P}, {G} and {O}?

- {P} is an object that contains 14 partitioner settings. They are:
  - `actual_levels` (boolean), `good_enough` (ε $\mathcal{R}$), `white_space` (ε [0,1]), `bMode` (enumeration), `Q` (ε $\mathcal{N}$), `Mapping` (ε [0,100]), `smoothing` (ε $\mathcal{N}$), `growing` (ε $\mathcal{N}$), `mode Bit 1, 2` and `3` (boolean), `maxNRLoadImbalance` (ε $\mathcal{R}$), `maxVirtualProc` (ε $\mathcal{N}$), `atomicUnit` (ε $\mathcal{N}$)
- {G} is an object that contains 6 parameters that characterize an AMR grid. They are
  - Number of levels (`N`), amount of refined area per level (`std::vector<double> of length N`), amount of refined area on level *l* normalized by the area of level *l-1* (`std::vector<double> of length N`), number of patches per unit area normalized by base grid size (ε $\mathcal{R}$), patch area statistics (μ, σ, max, min; `std::vector<double> a [N]`), average aspect ratio of a patch (ε $\mathcal{R}$)
- {O} is an object that contains 4 outcome (performance) parameters
  - synchronization cost statistics, load imbalance statistics, communication cost statistics, data migration cost statistics (μ, σ, max, min; `std::vector<double>` in each case).

I/O operations

- INPUT: The set { {P}, {G}, {O} } will be written simultaneously
- OUTPUT/SEARCHING
  - Given {G} and $\{\varepsilon_G\}$, where $\{\varepsilon_G\}$ is a tolerance, supply the top Q partitioner settings $\{P\}_{i=1...Q}$
    - Will supply a `bool comparatorG({G1},{G2},`{ε}) that determines if 2 grid parameters are within {ε} of each other
    - Will supply a `comparatorPO({P1},{O1},{P2},`{O2}) which returns the better of 2 partitioner settings `{P1}` and `{P2}`
  - Given {G}, $\{\varepsilon_G\}$, {P} and $\{\varepsilon_P\}$, supply the top Q {O}, along with the corresponding {G} and {P}
    - Will supply a `comparatorP({P1},{P2},`{ε}) that determines if 2 partitioner settings are within {ε} of each other