Application Structure Aware Resiliency and Cost Model for Differentiated Recovery

Anshu Dubey

April 28, 2015
Fault Detection

- A recent idea (definitely not implemented)
  - Applications know the conservative bounds for change in the values between time-steps
  - The space of floating point numbers that falls within the acceptable values is orders of magnitude smaller than that all floating point numbers of the given precision
  - A simple check for $|x_n - x_{n-1}|$ can determine if the result belongs in the valid space
    - Both the quantities are in cache so minimal cost
  - If the result is dubious, recompute $x_n$
  - If the answer is different, initiate rollback within the patch
  - If the answer is the same, possibly the simulation is going badly
    - Apply more diagnostics
    - Abort if necessary

In either situation this approach could reduce wasteful computations
Why Structure Aware Resiliency

It does not appear that rollback recovery can be eliminated as an option for resiliency

- Traditionally - checkpoint the entire state and reconstruct for restart
  - Application determined state for transparent reconstruction
  - For some types of faults that might still be the right solution

- Full state checkpoint is still an overhead

Confining the recovery to a section of the problem domain can provide considerable cost saving – idea behind containment domains

- This talk is not about containment domain, though they can be used
Differentiated Recovery: Four Simple Ideas

- See if there is any hierarchy to be exploited
- Identify granularities in the application
- Map fault types to granularities
- Determine the right amount of state to be saved at each granularity

Necessary to have knobs for tuning and therefore cost models

Also necessary to have differentiated state saving provided by some software that gives control and flexibility to the application
Two key ideas
- Multi-version, multi-stream distributed arrays: preserves application critical data, enables flexible recovery from complex errors with fine-grain, localized manner
- Open resilience: cross-layer partnership for error handling, with unified error handling interface

We exploit the flexibility in GVR interface to implement our differentiated recovery strategy
AMR Basics

- Block Structured Local Refinement
- Refined regions are organized into logically-rectangular patches
- Refined grids are dynamically created and destroyed
- Refinement is performed in time as well as in space
- The depth of hierarchy depends upon the range of scales
- Mostly solve hyperbolics, some elliptic and some PIC
What do we have going for us?

- AMR needs to do load balance dynamically
- AMR has hierarchy
  - A level is a fairly self contained unit of computation
- Not all levels will always have boxes on all processors
Exploiting Granularities and Encapsulations

A Level provides the coarsest granularity in the framework

Convention – level below is coarser, level above is finer

- A level defines data on unions of rectangles

A level can do its own restart

Save meta-data, offsets for each box physical data of each box, in array for a level
Exploiting Granularities and Encapsulations

The next layer of granularity is box within a level

- Box with its surrounding halo of ghost cells is a completely defined computational region
- All boxes mapped to a processor may not be at the same level

The way we construct arrays gives us box level granularity for free

Save meta-data, offsets for each box, physical data of each box, in array for a level
Failure Scenarios and Recovery Modes

- Resource failure, recovery at one or more levels, or a full restart

- Data corruption, recovery at one level

- Meta-data corruption: read in the meta-data, rollback may not be needed

- Data corruption – recovery at multiple levels
  - Cascading effect- recovery may be needed in all the finer levels (may not always be needed though)
Cost Model

Definitions of the terms used in the cost model:

- \( R \) - ratio between two consecutive levels

- \( T_{\text{config}} \) - time to configure a level

- \( T_{\text{alloc}} \) - time for allocating an array,

- \( T_{\text{realloc}} \) – time for freeing and reallocating an array

- \( T_{\text{ver}} \) - time to increment a version,

- \( T_{\text{put}} \) - time to gather and put data into the array for the whole level

- \( T_{\text{get}} \) - time to get data from GDS for the whole level

\[
T_{\text{level}}(i) = T_{\text{ver/realloc}}(i) + T_{\text{put}}(i)
\]

\[
T_{\text{save}} = \sum_{i=0}^{N-1} R^i \times T_{\text{level}}(i)
\]

\[
T_{\text{save}}(\text{global}) = \sum_{i=0}^{N-1} T_{\text{level}}(i)
\]

\[
T_{\text{lost}} = \sum_{i=m}^{N-1} R^i \times T_{\text{step}}(i)
\]

\[
T_{\text{reconfig}} = \sum_{i=m}^{N-1} T_{\text{config}}(i) + T_{\text{get}}(i)
\]

\[
T_{\text{recovery}} = T_{\text{lost}} + T_{\text{save}} + T_{\text{reconfig}}
\]
Using the Cost Model

Example: \( R = 2 \), \( \text{Levels} = 10 \) (including coarsest level)

*For simplicity assume \( T_{\text{step}} \) and \( T_{\text{level}} \) are the same for all levels*

In case of resource failure

For global snapshot per time step
\[ T_{\text{save}} = 10xT_{\text{level}} \]
\[ T_{\text{lost}} = NxT \ (\text{where } N \text{ is all timesteps computed since save}) \]

For non-global snapshots saving per time step for every level
\[ T_{\text{save}} = 1024xT_{\text{level}} \]
\[ T_{\text{lost}} = mxT_{\text{step}} \ (\text{m is the number of affected levels}), \text{ or} \]
\[ T_{\text{lost}} = 2(10-n)xT_{\text{step}} \ (\text{where } n \text{ is the coarsest level affected}) \]

In case of data corruption \( T_{\text{save}} \) is unchanged

For global snapshot
\( T_{\text{lost}} \) may be reduced if corruption is in level \( n \), and occurs within the first timestep of level \( n-1 \) (with some caveats)

For non-global snapshots \( T_{\text{lost}} \) is \( T_{\text{step}} \)
Preliminary Measurements

Comparison between file and GVR for saving state snapshot

- write to file
- write to GDS

Tlost for different values of m

- m=0
- m=1
- m=2
- m=3

Recovery as a percentage of run time

- fault at m=0
- fault at m=3
Conclusions and Future Work

- Examining the structure and granularities in the application can lead to differentiated recovery, which can be significantly less costly.

- With a strategy, knobs, and the corresponding cost model one can examine the trade-offs.

- We need to generalize this work.

- Definitely follow up on the fault detection idea.