NCE Project Progress Report for ET International Inc.

DynAX: Innovations in Programming Models, Compilers and Runtime Systems for Dynamic Adaptive Event Driven Execution Models

Award Number: DESC0008716
Dates of Performance: 9/1/2015 to 12/31/2015
Report Date: 12/31/2015

Principal Investigator
Guang Gao, ET International. Inc.

CoPIs:
Benoit Meister, Reservoir Labs, Inc.
David Padua, University of Illinois Urbana Champaign
Andres Marquez, Pacific Northwest National Laboratories

Introduction

ETI sent an NCE request letter on August 14th. This report outlines the work performed by ET International Inc. during the no-cost extension period granted. During the period we worked on two types of tasks: (1) tasks related to the dissemination of STI, and (2) tasks related to coordination and organization.

Summary

To summarize our STI dissemination efforts, (1) ETI presented our work on containment domain based resiliency within SWARM at the 2015 International Parallel Computing Conference (ParCo'15) in Edinburgh, Scotland[1], and (2) based on feedback from the community, we revised and extended our paper which is to be included in the primary ParCo'15 proceedings.

To summarize our coordination and organization efforts, (1) ETI as the lead institution coordinated with other PI institutions for the completion of their NCE, (2) organized completion of the Y3 report, and (3) put together the final report.

Dissemination of STI

The following sections discuss CD based resilience and contain improvements based off of feedback from the community following our presentation at ParCo'15. These changes are also reflected in our extended paper.

Containment Domain Based Resilience

For completions sake we will repeat that at a high-level, a containment domain contains four components: data preservation, to save any necessary input data; a body function which performs algorithmic work; a detection function to identify hardware and software errors; and a recovery method, to restore preserved data and re-execute the body function. The detection function is a user defined function that will be run after the body. It may check for hardware faults by reading error counters, or for software errors by examining output data (e.g. using a checksum function). Since containment domains can be nested, the recovery function may also escalate the error to its parent. Since no coordination is needed, any number of

[1] S. Kaplan, S. Pino, A. Landwehr, G. Gao. **"Landing Containment Domains on SWARM: Toward a Robust Resiliency Solution on a Dynamic Adaptive Runtime Machine."** To Appear in the proceedings of the 2015 international Parallel Computing conference (ParCo'15). Edinburgh, Scotland, UK, September 1 – 4, 2015.

containment domains may be in existence, with multiple preserves and recoveries taking place simultaneously.

Containment Domains API within SWARM

The containment domain API has been modified from what we initially reported to allow for additional features. Specially, the preservation calls have an additional parameter called *type* allowing the user to specify into which domain to preserve data. Additionally, we have clarified the language used in the API descriptions. The API's are now as follows:

swarm_Containment_Domain create(parent): Create a new containment domain as a child of the specified parent domain.

swarm_Containment_Domain begin(THIS, body, body ctxt, check, check cxt, done, done ctxt): Begin execution of the current containment domain denoted by THIS by scheduling the codelet denoted by body ctxt. When the codelet finishes execution, the codelet denoted by check ctxt is scheduled to verify results. If the result of the execution is TRUE then the codelet denoted by done cxt is scheduled.

swarm_Containment_Domain preserve(THIS, data, length, id, type): In the containment domain denoted by THIS, do a memory copy of length bytes from data into a temporary location inside the CD. We support multiple preservations per CD (e.g. to allow preservation of tiles within a larger array, such that the individual tiles are non-contiguous in memory), by adding a user-selected id field. For each containment domain in SWARM, a boolean value is set based on its execution status.On the first execution, data is preserved normally. On subsequent executions, data is copied in reverse (i.e. from the internal preservation into the data pointer). The CD in which the data is preserved is denoted by type. This can either be the currently activated CD or the parent CD.

swarm_Containment_Domain finish(THIS): Close the current containment domain denoted by THIS, discard any preserved data, and make the parent domain active.

Experimental Results

We've expanded upon and clarified the results we initially obtained and provided a more thorough discussion from that of the quarterly reports. We evaluate our CD based approach through three primary means: feasibility, efficiency, and resilience and make a number of key observations. To show that our prototype implementation has sufficient functionality, we instrument a Cholesky decomposition program in SWARM to use containment domains. For the experiments, the program was run on a dual-processor Intel Xeon system, using 12 threads. The workload sizes were confirmed to not exhaust the physical memory of the machine.Though we found insignificant variance between runs, we have averaged all times

over 5 program runs due to the natural variation in run time due to extraneous system factors (such as scheduling differences).

Figure 1: Execution Time of Cholesky

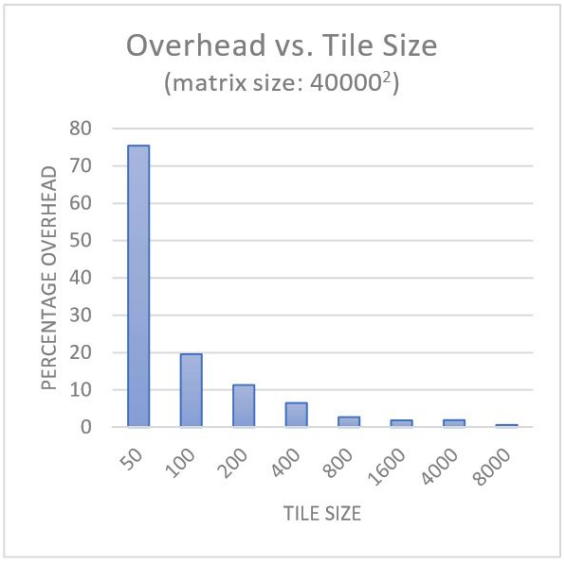

Figure 2: Percentage overhead

The Cholesky program has three main codelets, one for each linear algebra routine run on a tile (POTRF, TRSM, and GEMM/SYRK), and each of these is called a number of times for each tile. For our purposes, each of these is considered a containment domain. In our model, we only considered faults similar to arithmetic errors; that is, incorrectly calculated results. For this reason, we did not need to preserve input data unless it would be overwritten by an operation (e.g. the input/output tile for a POTRF operation). In order to simulate arithmetic errors, rather than relying on error counters from actual faulty hardware, a probabilistic random number generator is used. If a random number is below the specified configurable threshold, a fault is deemed to have occurred. Fault generation occurs within the check codelets causing checks to fail at random and the subsequent re-execution of the entire failed containment domain.

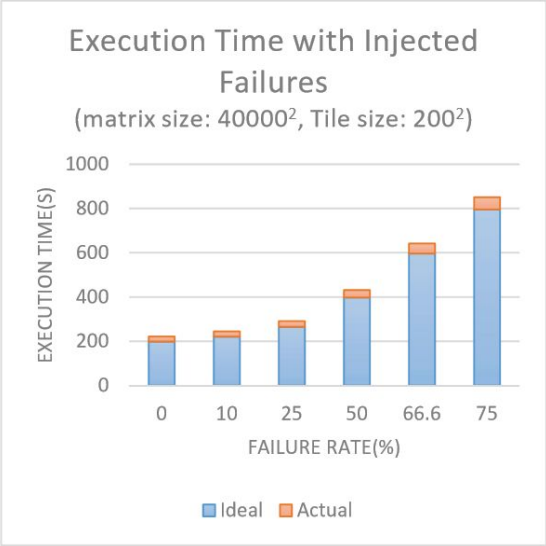

Figure 3: Execution time with simulated errors

Firstly, through the implementation of our framework in SWARM, and the working Cholesky application using said framework, we observe that it is feasible to adapt a codelet-based application to use containment domains. Secondly, Our implementation shows very low overhead. Figure 1 shows the execution time for various tile sizes executing Cholesky of size 40000x40000. Base denotes the Cholesky kernel runtime without containment domains or data preservation. CD denotes the time spent within CD related API calls without preservation. Preserve denotes the time spent preserving data. One can see that the API itself adds negligible overhead and that the only significant overhead comes from actual preservation of data. As with many tile based approaches to computation, choosing an inappropriate tile size will cause performance degradation due to inefficient use of caches or by limiting parallelism. In this particular case, tile sizes above 1600 limit the core utilization to 10 or 5 threads respectively, yielding the U-shape shown in the graph. Figure 2 shows the total overhead (preservation+API calls)relative to the base cholesky code without containment domains. The trends indicate that as tile sizes (workload per codelet) increase the overhead is mitigated and eventually becomes negligible.This trend is unsurprising given that the runtime overhead per API call is relatively constant and as the tile size increases less and increasingly larger data sized preservation calls are made to the runtime. Additionally, the cost of preservation (i.e. for data movement) increases at a much slower rate than the cost of the Cholesky computation as tile sizes are increased. Overall, this trend shows that there is a sweet spot in terms of granularity and that proper decomposition is key to mitigate preservation overheads and maximize performance.

Figure 3 shows simulated injected failures that result in codelet re-execution within the SWARM framework.The idealized case is computed by taking the average execution time without faults for a Cholesky of size 40000x40000 and tile size of 200x200, and computing the expected execution time for various fault rates using the geometric distribution. In order to

accurately access the overhead of our implementation, this projected base execution time includes neither CD or preservation overhead.The actual case shows the execution times actually obtained from running SWARM with injected failures. We note that there is around 11% overhead without failures and that this overhead decreases to 6% at a failure rate of 75%. This is because some allocation and API overheads are not present upon re-execution. Trend wise, we note that maximal overhead occurs when faults are not present in the system. We additionally note that the execution time follows reasonably well to that of the idealized case and that it is possible to project with reasonable accuracy the execution time in the event of failures using data from actual runs without failure.

Coordination and Organization Efforts

During the NCE period, we prepared the Y3 and Final reports to be submitted to the DOE. Activities included coordinating and communicating with each collaborating institution for the content and publications to submit as well as interfacing with the other institutions to make sure that the final report contained a smooth interface referencing the NCE reports by each institution which were submitted to the program manager as well as ETI. With respect to this, we received Reservoir Lab's NCE report dated 10/31/2015; PNNL's NCE report dated 12/23/2015. UIUC provided their report directly to the DOE on 8/31/2015. UIUC did not additional work after this date and will not be using the remaining funds. Additionally, we posted the NCE reports to the XStack Project Wiki. Direct Links to reports are included at the end of this document.

Status

We have completed the Y3 and Final reports and are in the process of submitting all documentation to the DOE and placing all documentation on the XStack Modelado wiki.

Report Links

The following URLs lead directly to other reports that have been submitted directly to the DOE for the DynAX project:

- PNNL NCE Report
- Reservoir Labs NCE Report
- Year 3 Report