Milestone 9 Status Report

Award #: **DE-SC0008717**
Recipient: **Intel Federal LLC**
Project Title: **TRALEIKA GLACIER X-STACK**
PI: **Shekhar Borkar**
Report Date: **December 8, 2014**
Period Covered by Report: **September 1, 2014 to November 30, 2014**

Acknowledgment: This material is based upon work supported by the Department of Energy [Office of Science] under Award Number DE-SC0008717.

Contents

Executive Summary

Open Community Runtime project made significant progress; it runs on all targeted platforms: distributed memory with MPI (or GasNet) implementation, as well as multi-block FSim implementation targeting TG architecture, and successfully tested for up to 256 XE's (Execution Engines), showing good scalability. We have established a roadmap for OCR for the remainder of the TG project.

Several DOE applications are now coded in CnC (Concurrent collections). LULESH coded in CnC works on FSIM, HPGMG is under development, CoMD runs on several x86 targets, and several others, such as CoHMM, AMR, MD are being coded using CnC. We have also started to develop an early prototype to demonstrate interoperability between CnC and HTA (Hierarchical Tiled Arrays). R-Stream optimizer, with enhanced OCR data-block support, demonstrates performance and productivity benefits by automatically producing efficient OCR code for HPGMG kernels (starting from few lines of C code) that gives comparable or better performance than OpenMP.

We completed development and testing of multiple versions of CoMD, with two different algorithms, each programed using MPI, MPI+OpenMP, OpenMP, MPI+PThreads, PThread, and OCR. The results were presented at TG technical meetings, SC14 BOF, and at the Co-HPC workshop (Codesign in HPC, held at SC14).

The architecture group has started evaluation of the TG system architecture with OCR and software-managed (incoherent) caches. Results show that with carefully orchestrated data movement, incoherent caches within a cluster provide about the same system performance as with hardware managed coherent caches. We plan to confirm these results using FSim.

For community outreach, we are working with the X-Stack PIs on design and development of a unified open community runtime and APIs that takes the best parts of their collective runtime research including OCR, HPX and combines them with mature environments like Charm++, OpenMP and MPI to offer portability, interoperability and legacy support. Following successful program with Oregon State University (OSU), we have started 3 new projects with undergraduate students from OSU related to OCR.

TG held a 3-day application workshop during the first week in October. We structured this workshop to be "hands-on" and used the last day to jointly code with our DOE colleagues on the proxy applications. And this quarter we held the 6th annual CnC workshop with excellent participation.

1	11/30/12	Architecture V2 spec & preliminary apps kernal identified for evaluation	Intel
2	3/1/13	Simulators V2 functional, tools (C + binutils) in place, IRR V1 identified	ETI, Reservoir
3	5/31/13	Selected kernels evaluated for 0(compute)	Intel, PNNL
4	8/30/13	Basic timing in simulator, intelligent scheduling in Exec model, tools (LLVM, etc)	ETI, Rice, Reservoir
5	11/27/13	Selected kernels evaluated for 0(com), select apps coded with PGM system for IRR	UCSD, PNNL
6	2/28/14	Architecture V2.5 spec, system evaluation of V2.0	Intel, UIUC (Josep)
7	5/30/14	Simulators V2.5 functional, tools for V2.5 released	ETI, Reservoir
8	8/29/14	System evaluation of V2.5	UIUC (Josep)
9	11/26/14	Arch V3.0 spec (ISA 4.1.0), selected apps evaluation with execution model and programming system for V2.5	Intel (with all)
10	2/27/15	Simulators V3.0 functional, tools for V3.0 released	Intel, Reservoir
11	5/29/15	Release OCR (Open Collaboration Runtime) V1.0	Rice
12	8/28/15	Evaluation of all X-Stack technologies and report	Intel

Intel

FSIM and OCR (Romain Cledat)

Introduction

During this quarter, Intel worked on the following areas:

- Following successes in similar projects last year with Oregon State University (OSU), Intel started 3 new projects with undergraduate students from OSU related to OCR. We will be working to a) develop an interactive web-based tutorial for OCR b) improve on and evaluate methodologies to identify performance bottlenecks in OCR particularly at scale, and c) port OCR to the Xeon Phi platform.
- OCR now runs correctly on all targeted platforms. Rice contributed the MPI implementation and Intel was responsible for developing the multi-block FSim implementation. The FSim implementation has been successfully tested for up to 32 blocks (256 XEs). We will improve performance for these implementations in the following quarters.
- The third Application Workshop was hosted at Intel with participation from our DOE partners. An initial draft of the OCR specification was released at that meeting.
- The specification for the v3 of the architecture was finalized. It includes new op codes for operations identified as common for DOE applications and features a 64 bit ISA. A whole new tool chain is being developed for this new architecture; we currently have the assembler and compiler and a port of libc is in the works. The simulator is also being updated for this new architecture.
- Several issues with OCR were fixed. The roadmap for OCR was also laid out internally.

Details of these points can be found in the 'Accomplishments' section directly below.

Accomplishments

OCR on FSim is now functional and has been tested with up to 32 blocks. We have identified several implementation issues that affect performance and will be tackling them in the next quarters.

Now that we have basic support for OCR on all our targeted platforms (x86, simulated TG on x86, TG and clusters), we also finalized and presented to our partners a simplified method to build and run codes written in OCR. This approach was demonstrated during our third Application Workshop with applications from DOE. Several issues were identified during the workshop and have since been fixed.

Based on feedback received during the workshop, a roadmap for OCR development for the remainder of the X-Stack project has been established. We intend to freeze most features by the end of 2014 and focus on implementation and performance optimizations in 2015. Our current focus is revamping the scheduler implementation, improving memory allocation (in particular on the TG platform) and adding programmer hints (in collaboration with Rice). A few additional API features will also be implemented based on requests from application developers.

Development for the simulator has been brought in house and we are actively working on finalizing the modifications required for v3 of the architecture. Most op codes were previously implemented by ETI and work has focused on implementing some of the newer features like queue and memory engines. Significant work has also gone in properly defining the memory map.

Plans

For the next milestone, we plan to:

- Freeze OCR features and start focusing on performance improvements.
- Wrap up development on the v3 tool chain and simulator.
- Evaluate applications using v3 of the architecture. OCR will also be "ported" to this new architecture to make use of some of new features (particularly queue and memory engines).
- Host another application workshop at the end of February.

Issues

None.

Applications (Bill Feiereisen)

Introduction

We introduced a unified build architecture for all the applications and kernels into the Traleika Glacier repository. This repository will contain all versions of all of the proxy applications and kernels in a uniform build procedure. It will also contain the scripts and datasets for running all the versions. We used this repository to hold the third Applications Workshop with our DOE colleagues in early October.

Accomplishments

Application Repository: This repository structure now allows us to run regular app regression tests upon the Traleika Glacier software and hardware infrastructure, providing a powerful tool for co-development of the architecture, the simulator and the OCR runtime.

Our goal is to provide this repository fully populated to the DOE for assessment and for use as teaching examples for refactoring codes in the high level programming notations. In preparation for our recent Applications Workshop we implemented the entire set of kernels in the build format and as of this

writing we now have one of the full proxy applications CoMD. The additional proxy applications in a selection of the programming notations are in work and will populate the repository in the next quarter.

Applications Workshop: We held the third Traleika Glacier 3-day Workshop during the first week in October. We structured this workshop to be "hands-on" and used the last day to jointly code with our DOE colleagues on the proxy applications. The software tool suite supporting these applications was implemented and proved sufficiently robust. In addition we invited our DOE colleagues to "bring their own codes" and we were able to support the partial refactoring into CnC on the Thursday of the workshop. This format proved very useful to our DOE colleagues and to our developments on the Traleika Glacier team. We received very favorable feedback from our Lab colleagues and from Intel management – a group achievement award was granted. We will follow this format for the next workshop in February.

Plans

During Q1-2015 we will fully populate the applications repository with the initial versions of all of the proxy applications, including the original versions for comparison. And we will expand the implementations of the proxies beyond the current OCR and CnC→ OCR versions.

Issues

None.

Community Development and Coordination (Wilf Pinfold)

Introduction

Intel is working with the X-Stack PIs on design and development of a unified open community runtime and API that takes the best parts of their collective runtime research including OCR, HPX and combines them with mature environments like Charm++, OpenMP and MPI to offer portability, interoperability and legacy support. We are exploring ways this unified runtime can be developed and maintained by the community through an independent foundation and how we can provide disciplined release engineering and continuous integration support in a way that allows vendors and software developers to collaborate without limiting innovation.

Accomplishments

Programming Environments Document: Started work on a document that will articulate how the technologies in the XPRESS project will integrate with DEGAS, D-TEC, and Traleika Glacier and the other X-Stack programs.

Runtime System Report: Put tools in place to enable team development of a document based on the April runtime summit that outlined a roadmap for achieving a unified runtime systems architecture for Exascale systems.

PI meetings: We have had monthly phone meetings between X-Stack PIs to coordinate research and a face to face meeting at SC14 where we discussed the context for exascale hardware changes and the comparative analysis of parallel environments.

Plans

- Interim Programming Environments Document target date: December 19, 2014
- Runtime System Report target date: January 30th 2015
- Runtime Systems Workshop: March 11-13, 2015

Issues

None.

Inventions

None.

Publications

None.

Reservoir Labs (Richard Lethin)

Introduction

This research memo describes the contributions of the Reservoir Labs X-Stack team during the period of September 15, 2014 through December 15, 2014. A summary of our contributions during this period includes:

- Demonstration of performance and productivity benefits of R-Stream by automatically producing efficient OCR code for HPGMG kernels (starting from few lines of C code) that gives comparable or better performance than OpenMP;
- Continued support for low-level tools (LLVM and binutils) for 32-bit and 64-bit versions of the Traleika Glacier (TG) ISA;
- Enhanced OCR datablocks support in R-Stream;
- Making the R-Stream generated HPGMG OCR code available to the TG team members through the Traleika Glacier X-Stack git repository. This is the only existing OCR code for HPGMG kernels.

Accomplishments

This section details our contributions during this reporting period.

LLVM and Binutils for TG ISA

We worked on an LLVM bug -- the compiler does not output debug information sections even when invoked with the debug flag (-g). The fix for the bug required changes in the LLVM compiler to enable generation of debug information and also in the assembler as it has to process additional information that it receives from the compiler.

We developed a fix to output debug information from the compiler by programmatically turning on a certain flag in the compiler and also re-writing a certain portion of the pretty printer to output 8 byte labels (as opposed to 2- or 4-byte labels, which was the default behavior).

The assembler (a part of binutils) was also updated to enhance relocation support which was necessitated by the debug information laden compiler output.

Datablocks Support in R-Stream

We worked on extending the OCR datablocks support in R-Stream. The existing capability in R-Stream creates one large datablock for each array and then creates smaller datablocks within EDTs that fit in the local scratchpad (for e.g. XE scratchpad in TG architecture) based on the data accessed within an EDT.

The new capability takes in a data partitioning (aka data tiling) specification from the user and creates initial datablocks according to the specification. R-Stream automatically figures out the datablocks that each EDT needs and creates an input slot for each datablock. Within an EDT, the data needed by the EDT from each of its input datablocks is automatically copied on to temporary local arrays that collectively fit in the local scratchpad attached to the processing element. This capability eliminates the need to create one large datablock for each array and provides a pathway to achieve scalable performance.

HPGMG Study, Analysis, and Mapping

Reservoir Labs investigated configurations with one of the HPGMG authors, and found that modern GMG solvers can best be proxied with the finite volume code configured with V-cycles, a biconjugate gradient stabilized (BiCgStab) bottom solver and Gauss-Seidel Red-Black (GSRB) smoothers operating at each level of the multigrid code. Future "exascale" GMG solvers are best proxied with finite volume code configured with full multigrid F-cycles, a BiCgStab bottom solver, and Chebyshev smoothers at each level.

Identification of Performance Critical Sections

Examination of HPGMG profiling data showed that performance critical sections include smoother, restriction, interpolation, and ghost zone exchange. Smoother was identified as a bottleneck and examined in more depth. Existing hierarchical parallelism was identified. Coarse grain parallelism was identified at the MPI layer where the problem domain was equally divided amongst ranks. Fine grain parallelism was identified where subsets of the per-rank set were assigned to individual OpenMP threads. The finest grain parallelism was found inside each box where subsets of the box were assigned to be computed by individual OpenMP threads.

For our initial experiments, we chose to map and optimize the finest grain parallel region. In future, we will be expanding the scope of R-Stream optimizations to include more coarse-grained parallel regions. Further, we will be also enabling broader fusion of smoother, restriction, residual computation, interpolation, and ghost zone exchange in different levels of the multigrid solve.

R-Stream Optimization

After identifying the smoother as a bottleneck area, the GSRB and Chebyshev smoothers were modified to enable automatic parallelization and optimization, and OCR code generation by the R-Stream compiler. As mentioned earlier, we mapped the finest grain parallel region and produced OCR code for it. The R-Stream generated OCR code was compared against the OpenMP code for the finest grain parallel region. Initial results showed that OCR performance is better than or comparable to the OpenMP performance.

These parallelized OCR versions have been code reviewed, committed to the X-Stack git repository, and successfully built on the X-Stack cluster. Preliminary performance data was presented at the most recent Traleika Glacier Applications Workshop at Intel's Hillsboro facilities.

CnC Collaboration

Reservoir has been collaborating with the CnC team at Intel/Rice and the PNNL team. A guide to building and running HPGMG was presented and subsequent collaborations have detailed a plan to enable the use of R-Stream optimized portions of HPGMG as CnC steps. Reservoir has been working with PNNL to define software interfaces between CnC steps and R-Stream optimized components of HPGMG including smoothers, restriction, residual computation, and interpolation operations.

SDSC Collaboration

An initial meeting with Laura Carrington's group at the San Diego Supercomputing Center has been set up to discuss HPGMG development.

Intel Collaboration

Reservoir Labs has been providing guidance to Gabriele Jost from Intel Corporation as she ramps up on the project. We have assisted with configuration, build, problem sizing, and environment setup.

Plans

For the next milestone, we currently have the following key ongoing tasks:

- Support LLVM and binutils for TG ISA;
- Extend the scope of R-Stream optimizations for HPGMG, specifically, enable broader cross kernel fusion and map more coarse-grained parallel regions in different levels of the multigrid solve;
- Enhance performance-scalable optimizations in R-Stream-OCR code generation.

Issues

None.

Inventions

None.

Publications

None.

Conclusion

During this quarter, we demonstrated the capability of R-Stream to automatically generate efficient OCR code for performance critical sections in the HPGMG benchmark. R-Stream takes few lines of (HPGMG) C code as input and produces efficient OCR code that gives comparable or better performance than OpenMP; this highlights the performance and productivity benefits that R-Stream offers to the exascale software stack. R-Stream generated OCR code is the only existing OCR code for HPGMG kernels and was made available to all TG team members through the X-stack git repository. Further, we continued development and maintenance of LLVM and binutils tools to support the 32-bit and 64-bit TG ISA. Furthermore, we made progress in extending the high-level compiler optimizations for TG through R-

Stream. Specifically, we developed capability in R-Stream for automatic creation and management of datablocks within EDTs according to a user-specified partitioning of data.

Rice University (Vivek Sarkar)

Accomplishments

- Distributed OCR: We have completed the implementation of OCR that works on distributed memory systems. This implementation can use MPI or GasNet for communication between shared memory nodes. We have tested and verified this implementation, and in the process discovered and fixed several correctness and performance bugs in OCR.
    - OCR implementations of applications or kernels of interest to DOE:
        - Conjugate Gradient
        - CoMD
        - LULESH
        - FFT
- OCR hints: we have identified several classes of hints that the user can provide to the runtime in order to achieve better performance and/or energy consumption. We have proposed several API changes and additions to OCR that will allow these mechanisms.
- OCR power API: we began the addition and integration of power-related APIs to OCR that will allow the OCR runtime to communicate with the underlying hardware power monitoring mechanisms, and to react to user-specified policies and hints regarding power.
- This quarter we held CnC'14 (6th annual CnC workshop) Sept 18-19. Chairs: John Feo (PNNL) and Sanjay Chatterjee (Intel) For details see: http://cass-mt.pnnl.gov/cnc2014/
- CnC implementations of applications of interest to DOE:
    - LULESH – (With PNNL): Currently works on FSIM but there is an FSIM issue that constrains the tile size of one element. See PNNL section for details.
    - HPGMG – (With PNNL): The CnC domain spec is under development. See PNNL section for details.
    - COMD – (With USCD): COMD in CnC - runs on several x86 targets. See UCSD section for more details
    - There are other DOE applications that have been implemented in CnC outside of the X-Stack project:
        - COHMM – (With LANL)
        - Adaptive Mesh Refinement – (With LANL)
        - Molecular dynamics – (With LLNL and UCLA)
- Other CnC accomplishments (infrastructure, features and additional applications):
    - Use of cancellation and priority mechanisms for convergence processing in Jacobi2d - this combination of facilities supports convergence tests without barriers and has potential for more general use.
    - Gene Sequencing - Vrije Universiteit (Yves Vandriessche et al): Here CnC is being used to coordinate among applications, not just within an application.

    - Ray tracing in CnC - PNNL (Ellen Porter et al)
- OCR implementation of CnC: We have completed the implementation of CnC on top of OCR. This implementation works on shared-memory OCR on x86, distributed-memory OCR using MPI or GasNet for communication, and on FSIM implementation of OCR.
- Unified CnC: We have started the efforts of unifying the different front-end implementations of CnC into a single CnC framework. This will allow programs written in CnC to execute on top of different CnC implementations (CnC-OCR, CnC-HC, Intel's CnC on C++, etc.) without any change in the CnC graph specification.
- Integration of CnC and HTA (With UIUC): We are developing an early prototype to demonstrate the interoperability of CnC and HTA. As a proof of concept, we code within CnC steps can invoke operations implemented in HTA. CnC manages the coarse-grain asynchronous task-based computation steps and HTA is responsible for the fine-grain data parallel computation within each coarse-grain step. For more details see the UIUC section.

Plans

- Power APIs: identify and define OCR APIs that will allow power/energy tuning within the OCR runtime
- Further performance and correctness improvements to the OCR implementations
- Integrated the Unified CnC approach into the X-Stack repositories
- Tuning for CnC on OCR

Issues

None.

Inventions

None.

Publications

None.

UCSD (Laura Carrington)

Proposed milestone

Work on experiments with CoMD on OCR-FSIM to explore energy and performance tradeoffs. Work on porting CoMD to CNC/OCR to compare to the hand coded version.

Interpretation: 1) Describe preliminary trade-offs in CoMD design: propose metrics, apply metrics, show quantitative results of metrics. 2) Work with Bill Feiereisen and DOE to identify Proxy App for next Q milestone by end of this Q; provide a written statement of Proxy App to be used in next Q efforts.

Issues and Limitations Encountered

The CnC version of CoMD is complete and we are still waiting on Nick Vivrlo to get a CnC->OCR-FSim working. The issues are not well defined and are currently being investigated, but it appears a problem in the translator. Without that version no experiments on the simulator can be performed.

There is currently a problem that prevents FSim to produce energy numbers (https://exascale-tech.com/trac/ticket/252) and until that is solved we will not be able to compare different code variants.

Progress

We completed development and testing of multiple versions of CoMD two different algorithms each programed using MPI, MPI+OpenMP, OpenMP, MPI+PThreads, PThread, and OCR. Results have been presented at TG technical meetings, SC14 BOF, and at the Co-HPC workshop (Codesign in HPC, held at SC14)[1]. We completed a series of performance experiments to analyze the performance features and bottlenecks of the different algorithms and programming environments. In addition we completed experiments to highlight how current bulk synchronous programming models are not capable of handling the anticipated dynamic environment of Exascale systems. This data was written up in a paper submitted to Co-HPC a workshop on Co-Design for SC14. The paper was also sent to the ExMatEx team and we have been in email discussion about the results.

We are experimenting with FSIM to define the methodology to measure performance and energy consumption to compare different variants of CoMD.

Both CG and CoMD have been adapted and committed to the repository in the “apps” directory according to the new format and are now integral part of the source tree. In addition, we contributed several standard C header files and functions to enable porting of other apps to TG/FSIM.

We also contributed several versions of CoMD to the ExMatEx github repository.

Next steps

We will continue experimenting with CoMD on FSIM and compare different variants for their performance and energy efficiency.

We are currently working on porting the HPGMG code to OCR. The data-flow below shows the different phases of the full multi-grid solver with a Chebyshev’s smoother and BICGstab iterative solver (Figure 1). We completed the basic structure of the graph and we are now adding the code of the various operators and the iterative solver.

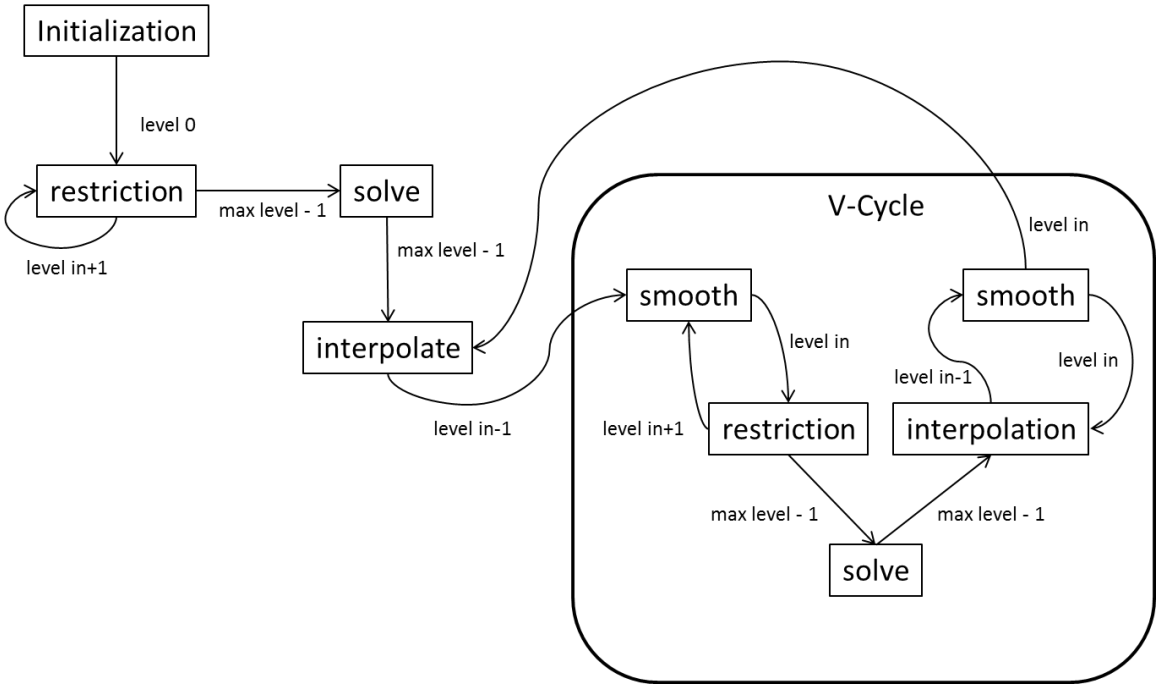

Figure 1 - HPGMG data flow

Publications

[1] P. Cicotti, S. Mniszewski, and L. Carrington, "An Evaluation of Threaded Models for a Classical MD Proxy Application."

University of Illinois Urbana Campus (David Padua)

Accomplishments

SPMD Extensions to HTA

This quarter we completed our first task parallelism implementation for HTAs. We are currently using this work to implement SPMD parallelism in HTAs to reduce synchronization overhead through the reduction of barriers required by our previous fork/join data parallel model. Significant changes need to be made to the HTA implementation to support SPMD parallelism, and these changes are underway.

Integration of CnC and HTA

We started to develop an early prototype to demonstrate the interoperability between CnC and HTA. We worked out a way to invoke matrix-vector multiplication operation implemented in HTA (running on OpenMP) within Cholesky factorization code written in CnC as a proof of concept. In this scenario, CnC manages the coarse-grain data-driven parallel computation on tiled matrices and the HTA operation is responsible for the fine-grain data parallel computation within each tile. The scenario provides programmers an easier way to manipulate fine-grain parallelism and multi-level tiling.

Graph Extensions for HTAs

Some graph algorithms can be written as linear algebra calculations, and linear algebra calculations have been well-studied in previous HTA related work. We studied the recent work of Saeed Melaki et al., "Tiled Linear Algebra: A System for Parallel Graph Algorithms'' (to appear in LCPC 2014), which proposed a new notation – Tiled Linear Algebra (TLA) to express Single Source Shortest Path (SSSP) algorithms in an elegant way. The TLA notation can be converted into HTA code easily and naturally.

One issue of TLA SSSP algorithms is the need of the global synchronizations, and a pipelined algorithm was proposed to reduce global synchronizations in implementation on distributed memory machines. We are investigating whether it is possible to utilize HTA implementation in SPMD execution mode to minimize the need to synchronize and improve performance on shared memory machines.

Plans

Graph Extensions for HTAs

We plan to continue our work on the graph extensions for HTAs, including an implementation of the SSSP algorithm previously mentioned. This implementation will take into account the optimizations we have been studying.

HTA and CnC Integration

We will continue our effort of the integration of HTAs with CnC.

HTA Implementation Improvements

Next quarter we plan to complete the SPMD implementation for HTAs. We also plan to study any chances of optimizations in the implementation of the HTA library as well as the generated OCR code.

Issues

We currently have no issues for this quarter.

Inventions

None.

Publications

None.

University of Illinois Urbana Campus (Josep Torrellas)

Accomplishments

In this quarter, Wooil Kim and Josep Torrellas accomplished two main things. We have initiated the evaluation of the TG system with OCR and further evaluated Software-Managed (i.e., Incoherent) caches. We describe each in turn.

TG system with OCR

We have initiated the evaluation of the TG system (architecture and OCR runtime system) on the FSim simulator. This effort continues the work we did in the last milestone, when we evaluated the TG architecture with an earlier, non-OCR runtime system.

However, the FSim simulator does not yet completely support the OCR system. Consequently, we have only been able to obtain limited results. We have been able to run a Cholesky matrix factorization program and obtain performance numbers as we change the number of blocks and the matrix tile size. Moreover, all memory allocation is done in the off-chip DRAM. Hence, remote memory accesses mostly go to DRAM. We expect that, in the future, OCR will support memory allocation in the scratch-pad, BSM, and USM.

Cholesky Program

Cholesky is a decomposition method of a positive, definite matrix into a product of a lower triangular matrix and its conjugate transpose. The algorithm is iterative, and values produced in the previous iteration are used in the current iteration. The algorithm has O(N^3) complexity because it iterates N (the matrix size in one dimension) times for matrix elements (N x N). The simplified sequential algorithm is written as follows:

```
for (k = 0; k < N; k++) {
  A[k][k] = sqrt(A[k][k]);
   for (j = k+1; j < N; j++) {
       A[j][k] = A[j][k] / A[k][k];
       for (i = k+1; i < N; i++) {
           A[i][j] = A[i][j] - A[i][k] * A[j][k];
       }
   }
}
```

The parallel version of Cholesky divides an entire input matrix into t x t tiles, where t (the number of tiles in one dimension) is N (the matrix size in one dimension) divided by T (the tile size in one dimension). An external program transforms an original matrix into a tiled lower triangular form, and the Cholesky program starts from it. The breaking of the program into different tasks (called EDTs or Event-Driven Tasks) was described in the previous milestone report.

All EDTs are created early on, but they wait until their dependences are resolved. The degree of parallelism in Cholesky is affected by two main factors. The first one is the number of tiles. Given a fixed problem size, more tiles with smaller tile size enable more parallelism. However, increasing the number of tiles also increases the overhead. The second factor is related to the runtime. When there are many EDTs that are ready to execute, the runtime picks one among them and assigns it to an idle XE. If the runtime chooses an EDT with more dependent EDTs, the completion of the EDT increases the number of executable EDTs. Overall, Cholesky shows a highly-variable degree of parallelism.

Performance as We Change the Matrix Tile Size

Figure 2 shows the execution time (in billions of cycles) of Cholesky with matrix size 500 running on 2 blocks with 8 XEs each. We vary the tile size from 250 to 100, 50, and 25. As we decrease tile size, more EDTs are generated, and more numbers of concurrent EDTs are available. Consequently, more parallelism is exposed. We see speed-up as the number of EDTs increases. Hence, when the programmer exposes more parallelism in the problem, the current runtime exploits it well.

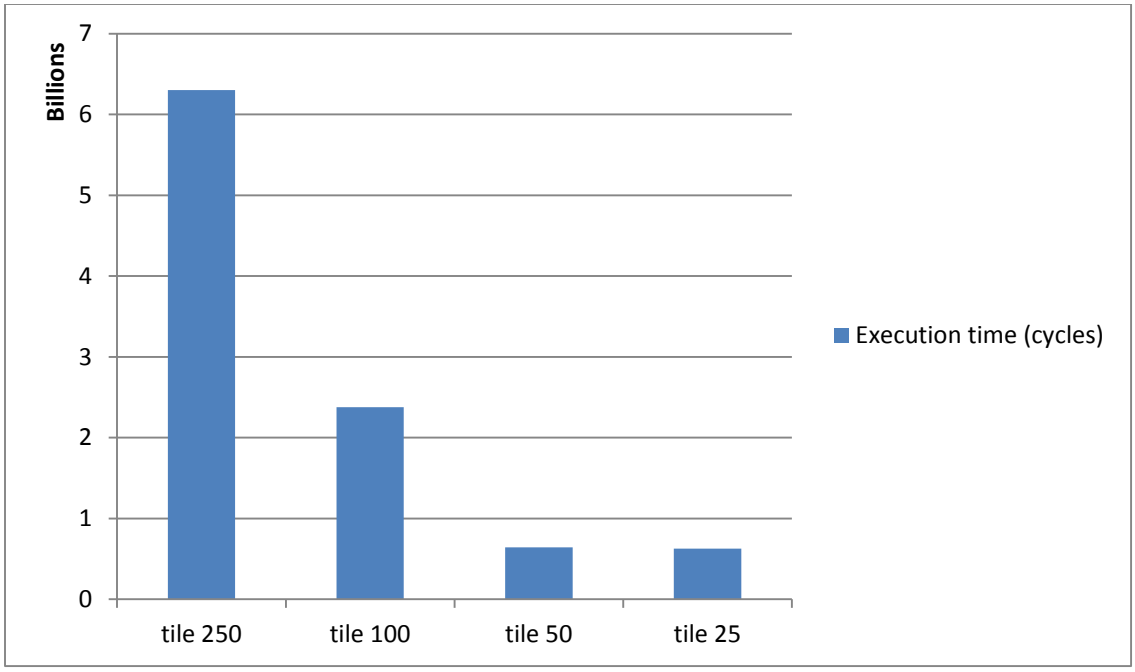

Figure 2 - Execution time (in billions of cycles) of Cholesky for matrix 500 and varying tile size, for 2 blocks with 8 XEs each.

Performance as We Change the Number of Blocks

Figure 3 shows the execution time (in billions of cycles) of Cholesky with matrix size 500 for different tile sizes and different number of blocks and XEs per block. We have three tile sizes: 100, 50, and 25. For each, we run it on 1 block (8XEs), 2 blocks (16XEs), and 4 blocks (32XEs).

We can see that there are slowdown effects when we add more blocks. With the fixed size of the matrix and the tile, increasing the number of blocks does not improve the performance but harms it. This might be a synchronization issue in the multi-block simulation, or it might be due an inefficiency in the OCR.

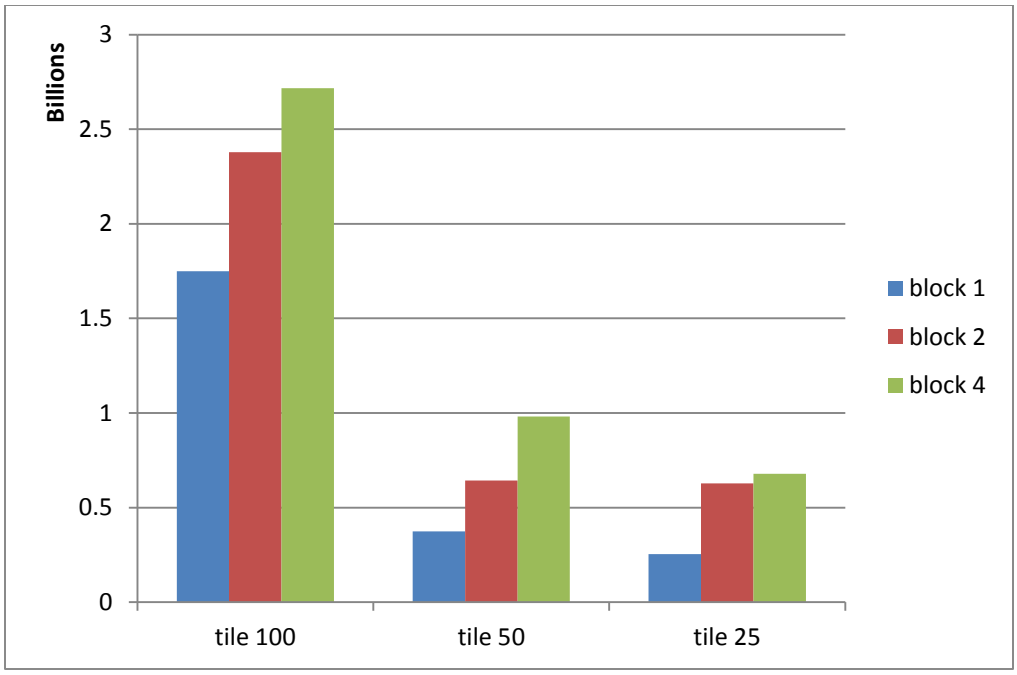

Figure 3 - Execution time (in billions of cycles) of Cholesky for matrix 500, varying tile size and varying number of blocks used (one, two or four blocks, of 8 XEs each).

Load Balance

In terms of number of instructions, loads and stores, the program is very load balanced. We run Cholesky for matrix 500 and tile size 50. In Figure 4, the program runs one 1 block. The figure shows the breakdown of instructions, remote reads and writes, and local reads and writes across the 8 XEs. In Figure 5, the same program runs on 4 blocks. The figure shows the breakdown of instructions, remote reads and writes, and local reads and writes across the 4 blocks. We can see that the system is well load-balanced.

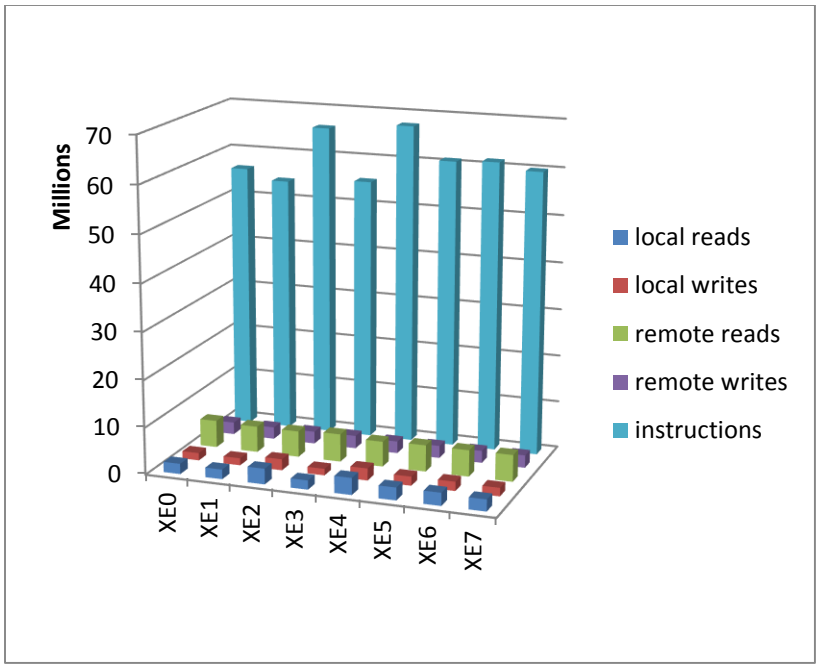

Figure 4 - Breakdown of instructions, remote reads and writes, and local reads and writes, across the 8 XEs of a block.

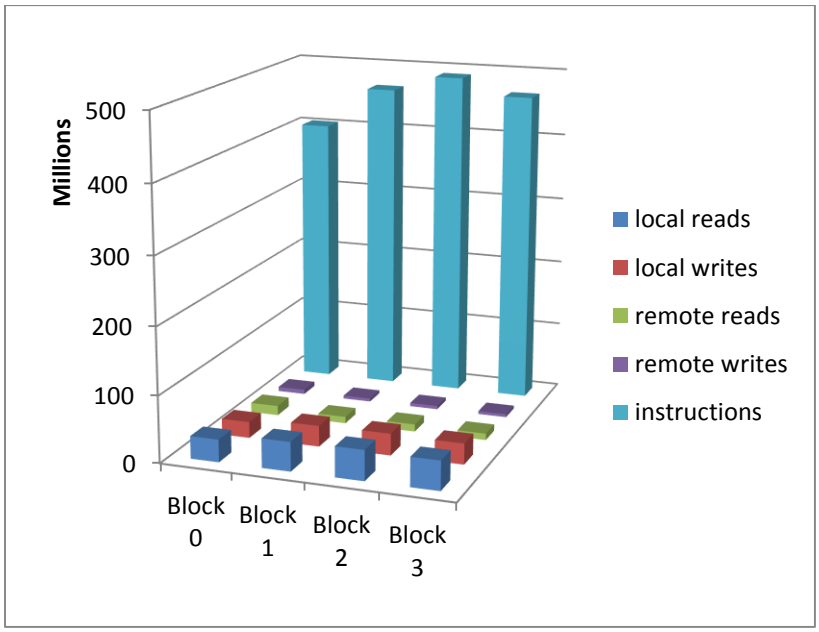

Figure 5 - Breakdown of instructions, remote reads and writes, and local reads and writes, across the 4 blocks of a 4-block run.

We expect to continue debugging the evaluation of the TG system.

Evaluation of Software Managed Caches (In Preparation for FSIM)

We have continued to evaluate the API for software managed caches (i.e., incoherent caches). Since the FSIM support for this API is not fully debugged yet, we have created a simpler simulator that allows us to perform an initial evaluation and get ready for when FSIM's support is available.

Recall that our ISA support to manage an incoherent cache hierarchy is based on *writeback* and *self-invalidation* instructions. We studied the subtle issues that these instructions need to face, including reordering in the pipeline, and effective use of caches organized in blocks for energy efficiency.

We took a set of applications from the SPLASH-2 benchmark suite, and developed a simple approach that the programmer can use to orchestrate the movement of data. It is based on exploiting the synchronization points in the program and the type of synchronization operations. Moreover, if the programmer provides additional communication pattern information, we can improve the performance. Figure 6 shows the annotations we use for communication patterns enabled by barriers (a), critical sections (b), flags (c), and dynamic happens-before epoch orderings (d). In the figure, WB stands for writeback and INV for self-invalidation instructions.

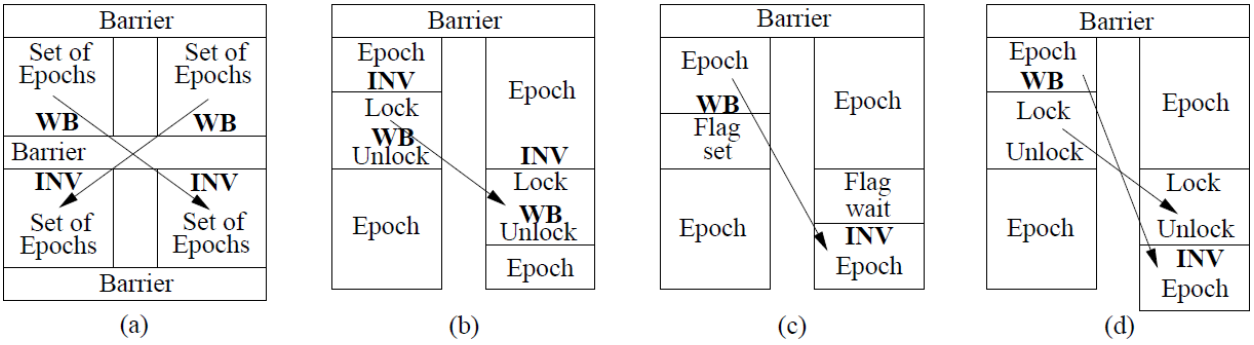

Figure 6 - Annotations for different communication patterns.

Our simulation results show that the execution of applications on incoherent cache hierarchies can deliver reasonable performance. Figure 7 shows the normalized execution time of the parallel phase of the applications with different levels of hardware support. For each application, and for the average, we show 5 bars. Starting from the left, the first bar shows execution time for directory-based hardware cache coherence, and all bars are normalized to this configuration.

The rest of the bars show: our baseline incoherent caches, and then three bars after applying various combinations of our optimizations on the baseline incoherent caches. The last bar in each application is the best optimization possible. Each bar is broken down into 5 categories: busy time plus memory access stall (execution), stall due to barrier (barrier stall) and lock (lock stall) synchronization, and stall time due to writeback (WB stall) and self-invalidation (INV stall).

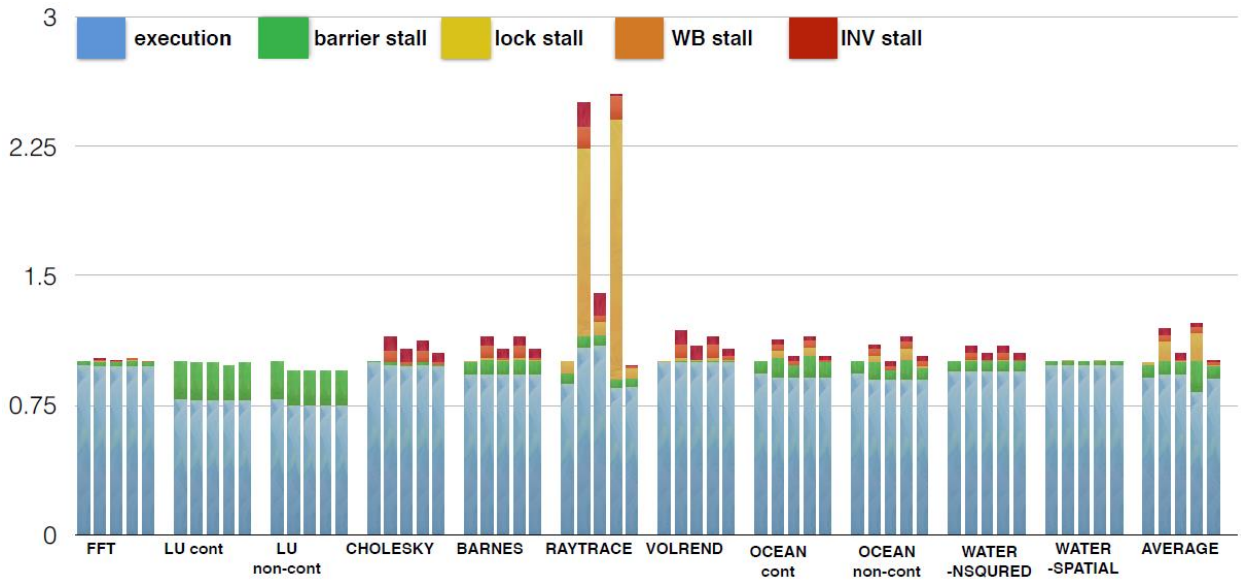

Figure 7 - Execution time of the applications.

From these figures, we show that, if we carefully orchestrate the data movement, the execution using incoherent caches within a cluster is not much slower than using hardware cache coherence in these applications. We look forward to being able to try these experiments on FSIM.

Issues

We need to work with the rest of the team members so that FSIM fully supports OCR for different numbers of processors and allocating the data in the block memories. We also need to have the support for incoherent caches fully debugged in FSim.

Plans for next milestone

Continue to evaluate the TG architecture on FSIM, focusing on having OCR fully implemented and evaluated, and then evaluate incoherent caches in FSim.

Inventions

None.

Publications

Currently working on:

Title: Architecting and Programming an Incoherent Multiprocessor Cache Hierarchy
Authors: Wooil Kim, Sanket Tavarageri, Josep Torrellas, and P Sadayappan

Pacific Northwest National Laboratory (John Feo)

Accomplishments

Application Work on CNC

Three versions of Lulesh have been pushed to the X-Stack repository: C, CnC-Intel, CnC-OCR. Lulesh in CnC-OCR form has successfully run on FSim with 1 and 2 element domain sizes. We will present Lulesh characterization data in the near future. Our current efforts focus on HPGMG. Our current HPGMG implementation is a follows: The first step was to build the HPGMG control in CnC and implement its different cycles (v, f and w). Under the CnC programming model, keeping the cycle code separate from the step code becomes trivial. Thus allowing us to reuse the steps regardless of which cycle is running. The second step was then to implement and test this theory. The first two steps are complete and we are currently working on the third step; integrating the CnC control into the existing HPGMG code.

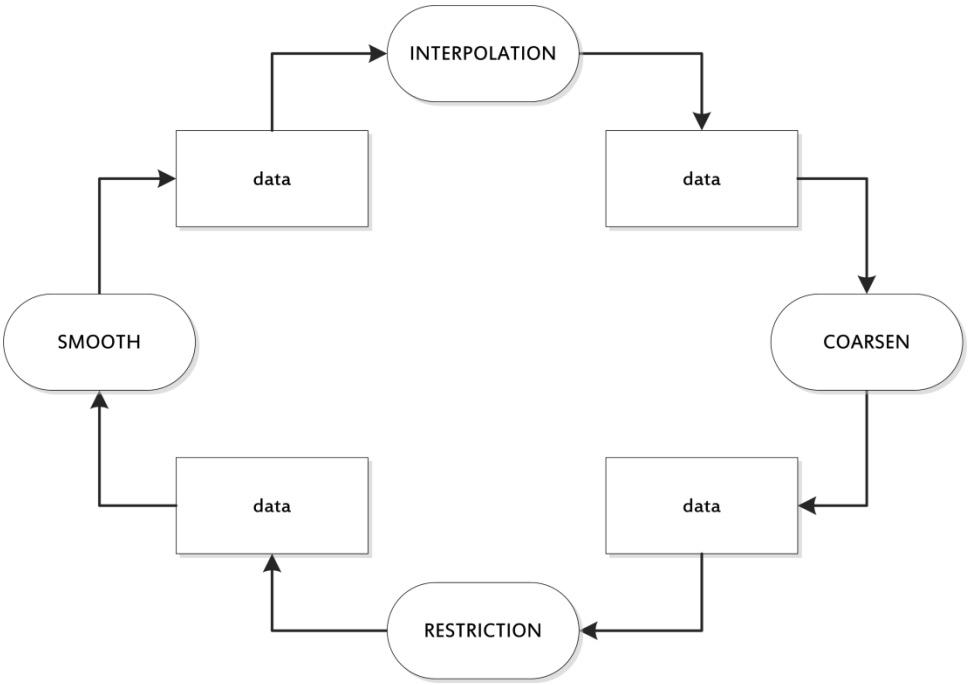

Figure 8 - The application flow for the HPGMG mini app

ACDTF on OCR

As part of the XSTACK Traleika Project, PNNL is developing a prototype of a dynamic self-aware lightweight system [1], the Architected Composite Data Type Framework (ACDTF) – originally called the Rescinded Primitive Data Type (RPDT) -- running on top of the Open Community Runtime (OCR) interface (ACDTF-OCR). This prototype will feature, once completed, real time decision-based compression, dynamic range adjustment and algebraic invariant operations that use real-time sampling.

[1] Marquez A, JB Manzano Franco, S Song, B Meister, S Shrestha, T St. John, and GR Gao. 2014. "ACDT: Architected Composite Data Types. Trading-in Unfettered Data Access for Improved Execution." To Appear In *The 20th IEEE International Conference on Parallel and Distributed Systems*.

The current prototype has been implemented across three hardware platforms: *a Linux based cluster*, *a shared memory node* and *the Traleika Glacier Simulator (FSIM)*. The prototype supports two compression algorithms (The Floating Point Compression and an in-situ Run Length Encoding)[2], an enhanced compressed format (based on the sparse compressed block format) and two sampling methods (based on block and differential run-length sparseness).

In addition, the prototype offers wrappers to the OCR's interfaces and expands upon them. It adds typing information and lets users define block composites, decision thresholds, invariant operations, sampling points, and macros to transform code segments into single/double precision. Governors that make the decisions to compress, sample, and actually do compression are configurable to run on dedicated cores, at OCR interface points, or across all cores. Furthermore, these governors can run asynchronously or synchronously. Thanks to these features, we can do exploration studies on the best prototype configurations across various platforms.

As part of our experiments, we have concentrated on the Cholesky kernel and characterized its performance and its ACDTF overhead.

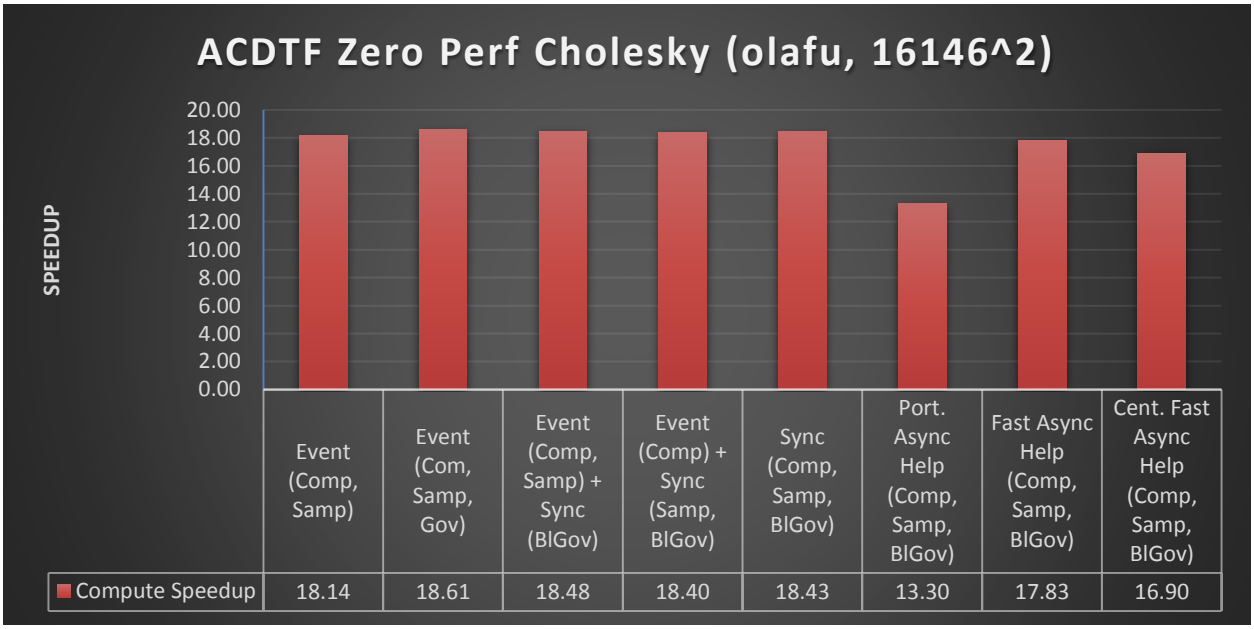

Figure 9 - ACDFT-OCR shows sparse matrix compression improvements for "olafu" at event points across its different configurations -- synchronous vs. (fast-) asynchronous, sampling vs. non-sampling, various governor strategies -. (Comp = Compression, Samp = Sampling, Gov = Governor). Furthermore, the asynchronous version has a portable implementation that works on any version of OCR, and a fast version which requires pointers for the platform to be able to address any memory location. ACDTF supports a centralized core to handle requests (Cent). Block size is ~300.

[2] Peter Lindstrom and Martin Isenburg. 2006. Fast and Efficient Compression of Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (September 2006), 1245-1250. DOI=10.1109/TVCG.2006.143 http://dx.doi.org/10.1109/TVCG.2006.143

Preliminary results show that we are able to achieve performance rates between 0.99x to 18x vs. non-enhanced OCR across various sparse matrices using a 16 core Xeon system regardless of the configuration of our system, as long as we detect invariant operations and use our compressed zero-configuration. All others configurations show no improvement or less than 1% performance degradation due to overhead being low for the coarseness of the work (~300^2 block size). We expect other configurations to excel, once we exercise them on improved distributed ACDT-OCR implementations.

Issues

Salient limitations to characterize ACDTF-OCR performance on other platforms include limited FSIM accuracy as well as only global data allocation support. Poor performance of the distributed OCR version is another major culprit.

The lack of application and kernels ported and optimized to OCR.

Goals for Next Quarter

Further integration of ACDTF-OCR, including and adding better multi-level memory support in the simulator and on X86. However, some of these features require OCR modifications.

We will be exploring other applications.

Continue developing HPGMG in CnC.

Inventions

None.

Publications

None.