Milestone 11 Status Report

Award #: **DE-SC0008717**
Recipient: **Intel Federal LLC**
Project Title: **TRALEIKA GLACIER X-STACK**
PI: **Shekhar Borkar**
Report Date: June 2, 2015
Period Covered by Report: **March 1, 2015 to May 31, 2015**

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

Contents

Executive Summary

Our major accomplishment this quarter is the release of Open Community Runtime, OCR 1.0, with all the supporting documentation, including specification and development support (Wiki, code repository, code-review, bug tracker, etc.). We have also started exploring how OCR could benefit from LEGION, more specifically, from the underlying task-driven runtime called REALM.

We started discussions with the contributing vendors of the ECI program (IBM, Cray, AMD, ARM, nVidia) about joint support of a collaborative interaction and development capability supported by a neutral party, and hosting or linking to publicly available services for source repository, regression testing, code review and bug tracking. This effort will help converge the community towards the common Exascale goal.

We hosted a two and a half day application workshop including an OCR tutorial, in an informal setting to foster impromptu conversations and hands-on application coding. The group discussed applications, refactoring, and targeting them for TG architecture. This workshop attracted over 65 attendees, with about 30 from DOE lab application developers. R-stream generated OCR code for HPGMG kernels was one of the highlights of the workshop, demonstrating capability for automatic high-performance OCR code generation.

The team is now feverishly working toward the final milestone to put the SW stack together, and evaluate the TG architecture.

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 – Shekhar Borkar

FSIM and OCR (Romain Cledat)

Accomplishments

Our major accomplishment this quarter is the release of OCR 1.0 as well as the supporting documentation, examples, specification and development support (Wiki, code repository, code-review, bug tracker, etc.). More information on the release can be found here: https://xstack.exascale-tech.com/wiki. OCR 1.0's specification is also appended to this report . This release focuses on a functional implementation of OCR on x86 as well as cluster machines. During the final quarter of this project, we will be focused on **(a)** improving the x86/cluster implementations and **(b)** releasing the Traleika Glacier implementation.

During this quarter we also worked on the following aspects of OCR (more details below):

- Started a discussion about possible collaboration with Pat McCormick's team on LEGION
- Preliminary work on standardizing runtime profiling data
- Hosted a workshop on application development on OCR
- Added preliminary support for introspection to the OCR runtime on x86
- Several "under the hood" performance improvements

OCR and LEGION
We started exploring how OCR could benefit from LEGION or, more specifically, the underlying task-driven runtime called REALM. We had a preliminary discussion with Pat McCormick's team during the application workshop and followed up with 2 deep-dives on REALM and LEGION. We intend to continue with a 2 day F2F with Alex Aiken's team (the creators of REALM and LEGION) to understand how we can integrate some of their ideas to OCR.

Runtime profiling
Members of the OCR team travelled to LANL to meet with Martin Schulz to discuss standard ways to collect profiling information on task-based runtimes. Collaboration is ongoing.

Application workshop
We hosted a 2.5 days application workshop including an OCR tutorial. The format of the workshop was very informal to foster impromptu conversations and hands-on application coding. Feedback from the DoE and other participants was very positive and we will continue with this format going forward. The workshop led to the two previously discussed collaborations (LEGION and runtime profiling).

Introspection support
Preliminary support for introspection was added to the OCR implementation on x86. It currently allows the user to "pause" the execution of an OCR program and query some basic information about the number of tasks in flights, etc.

Performance improvements
We continued to identify and remedy performance bottlenecks.

Plans

For the next milestone, we plan to:

- The port of OCR to v3 of the architecture is still not complete but we plan to have it by the end of the program
- Focus on improving performance and refining OCR's internal design to make it easier for others to use and contribute

- Report out on our learnings (both negative and positive)

Issues

None

Inventions

To be reported by Jun 30.

Publications

OCR 1.0 Specification, located on the public wiki - https://xstack.exascale-tech.com/git/public?p=xstack.git;a=blob_plain;f=ocr/spec/ocr-1.0.0.pdf

Applications (Bill Feiereisen)

Introduction

During the period March 15, 2014 - May 15, 2015 we made significant progress in the refactorization of proxy applications and "teaching" kernels into the revolutionary programming models that are based upon the OCR dynamic runtime. We also built upon the unified build structure of the application and runtime repository that we set up last quarter, by adding a formal regression test system. This system allows for nightly tests of all the proxies and kernels upon the runtime. This assists in the evaluation of the initial public release of the system V1.0 that is the major deliverable for this period. It has proven its worth in runtime debugging and necessary subtle changes in applications refactoring.

Accomplishments

Proxies and Kernels: During the course of this quarter we implemented a large selection of proxies, kernels and teaching examples. These are now implemented in publicly accessible form in our git repository. In addition we now have refactored forms of the current proxies and kernels implemented into the Jenkins regression system. The git repository itself, containing all of the refactored applications has now been made public accessible and will be open for public contribution at the release of OCR v1.0. There is now an extensive set of documentation included in the Xstack wiki detailing progress and findings.

For the Release of OCR V1.0 this is the set of example proxies and kernels

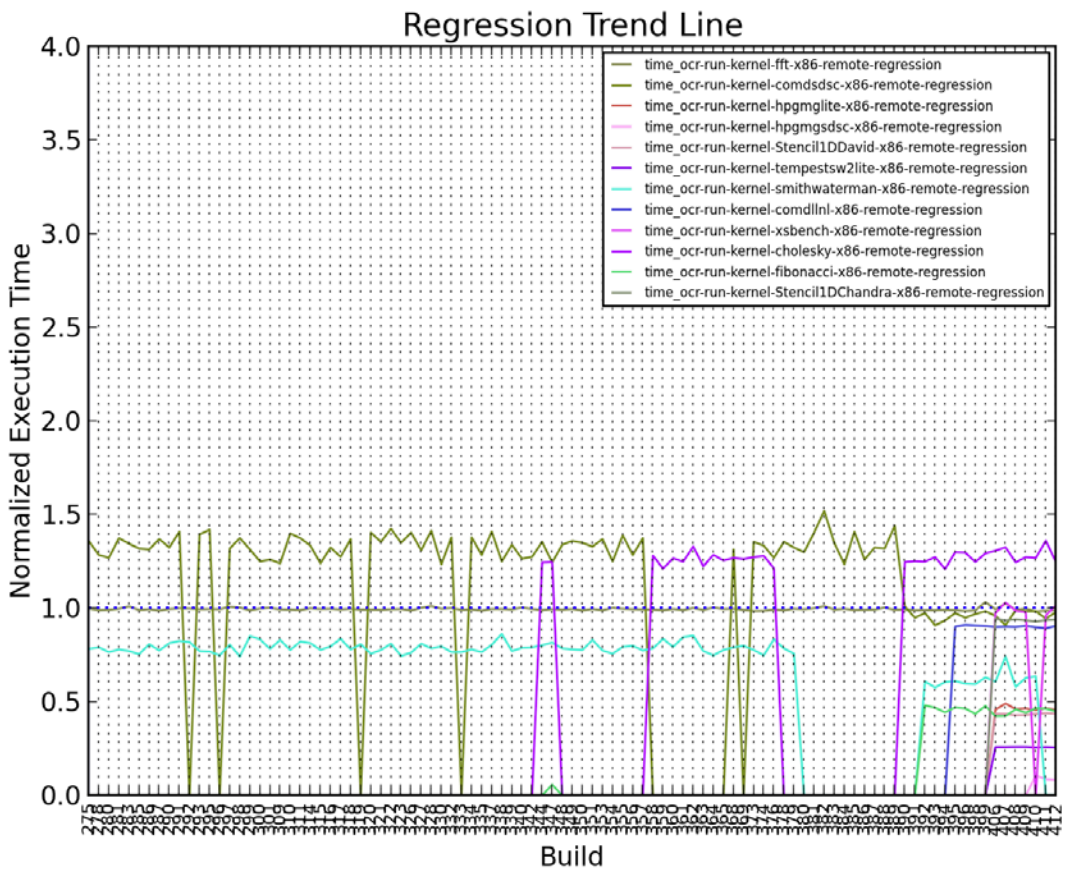
Table 1: Proxies, Kernels and Teaching examples implemented
Multiple entries indicate alternative implementations with varying algorithms

Application	**Programming Model**
CoMD	Baseline MPI+OpenMP
CoMD	Baseline MPI
CoMD	Baseline OpenMP
CoMD	Legacy serial on OCR, (identical to the original doe application except that it relies on newlib which then runs on top of OCR)
CoMD	MPI-Lite

CoMD	CnC on OCR
CoMD	OCR
HPGMG	Baseline DOE Original in MPI+OpenMP
HPGMG	MPI-Lite
HPGMG	ROCR (R Stream --> OCR)
HPGMG	OCR
LULESH	Baseline MPI+OpenMP
LULESH	Intel CnC
LULESH	iCnC
LULESH	CnC on OCR
miniAMR	Baseline DOE Original in OpenMP
SNAP	Baseline Translated into C from the DOE Original
SNAP	MPI-Lite
Tempest	Baseline DOE Original in MPI
Tempest	MPI-Lite
RSBench	Baseline in OpenMP
XSBench	Baseline MPI+OpenMP
XSBench	MPI-Lite
XSBench	OCR
Hello World	OCR
Stencil1D	OCR
Stencil1D	OCR
Stencil1D	MPI
Stencil1D	MPI-Lite
Stencil1D	Serial
Cholesky	OCR
Smith Waterman	OCR
FFT	OCR
Fibonacci	OCR
Unbalanced Tree Search (UTS)	OCR
Synthetic Aperture Radar (SAR)	OCR
Global Sum	OCR
triangle	Serial
triangle	OCR
Synch_p2p	OCR

Regression testing is quite extensive. Figure 1 shows the extent of regression on the refactored proxies and kernels:

Figure 1: Night Regression testing of the proxies and kernels



Applications Workshop #4 was held in Hillsboro, OR April 7-8. This workshop attracted over 65 people of which about 30 were DOE laboratory application developers. We structured the workshop to accommodate both experience colleagues who were able to dive right in and, those who had not programmed in our systems. We held a set of tutorials to help our new colleagues get up to speed. Most of the time was spent in joint coding and discussion. We received very good feedback and will use this template to structure future workshops.

Plans

During Q3-2015 we will continue to populate the applications repository with the final documented 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. The documentation detailing how-to's and best practices will be documented for the end of the program

Issues

None

Inventions
None

Publications
None

Community Development and Coordination (Wilf Pinfold)

Introduction
We started discussions with the contributing vendors of the ECI program (IBM, Cray, AMD, ARM, nVidia) about joint support of a collaborative interaction and development capability supported by a neutral party and hosting or linking to publicly available services for source repository, regression testing, code review and bug tracking. There is also possibly a need for binary package hosting. Our objective is to get letters of support in early June, funding in July and services operational by end of year 2015.

Discussions continued with Sandia National Labs (Ron Brightwell and Robert Clay) and Office of Science (Sonia Sachs) about runtime convergence. OCR is now seen as an Intel technology and thus competitive to HPX, Realm, and Charm. Ron Brightwell has been asked to set up a Labs steering committee to look for the best way to consolidate. The starting proposal is to set up an MPI forum like structure.

Accomplishments
Discussions aligning the five vendors on the structure and cost of collaborative services complete

Letters drafted soliciting financial support for common collaborative services.

Plans
Send letter of solicitation to the five vendors and secure funding

Issues
None.

Inventions
None.

Publications

Reservoir Labs - Richard Lethin

Introduction

This research memo describes the contributions of the Reservoir Labs X-Stack team during the period of March 15, 2015 through June 15, 2015. A summary of our contributions during this period includes:

- Demonstrating scalable performance of R-Stream-generated OCR code for HPGMG kernels at Intel's Exascale Applications Workshop in April
- Demonstrating R-Stream capability for automatic high-performance OCR code generation to participants from X-Stack TG project and DOE Labs at Intel's Exascale Applications Workshop in April
- Completing R-Stream-OCR task items for the public release of OCRv1.0
- Supporting the integration of R-Stream and HTA/PIL
- Supporting users using R-Stream installed in the Intel X-Stack cluster

Accomplishments

This section details our contributions during this reporting period.

HPGMG Mapping

During this quarter, we continued our efforts on optimizing coarser regions of the HPGMG smoother kernels (especially the Chebyshev smoother) to exploit the opportunities in executing these regions in a more asynchronous fashion in an EDT-based runtime such as OCR. We isolated a coarse grain Chebyshev kernel representing the entire smooth function and parallelized with multiple methods (namely, hand parallelized OpenMP, R-Stream-generated OpenMP, and R-Stream-generated OCR), as shown in Figure 1.

R-Stream-generated OCR code enables a scalable asynchronous EDT-based execution. R-Stream has a lightweight runtime layer on top of OCR that enables on-the-fly just-in-time creation of EDTs and datablocks, and enables dynamic creation and handling of dependence events between EDTs. This avoids any unnecessary runtime overhead and enables scalable performance. This turned out be key for the Chebyshev kernel.

Further, R-Stream applies key compiler optimizations and generates OCR code. For the Chebyshev kernel, the optimizations included - (1) smart fusion of Chebyshev smoother loops, (2) tiling across multiple smooth steps, and (3) autotuned tile dimensions. Due to the afore-mentioned compiler and runtime optimizations, R-Stream OCR code turned out to be the fastest among all parallelized codes. R-Stream OCR code was up to 5x faster than hand parallelized OpenMP code. Further, R-Stream OCR code was also faster than fused, tiled and autotuned R-Stream OpenMP code.

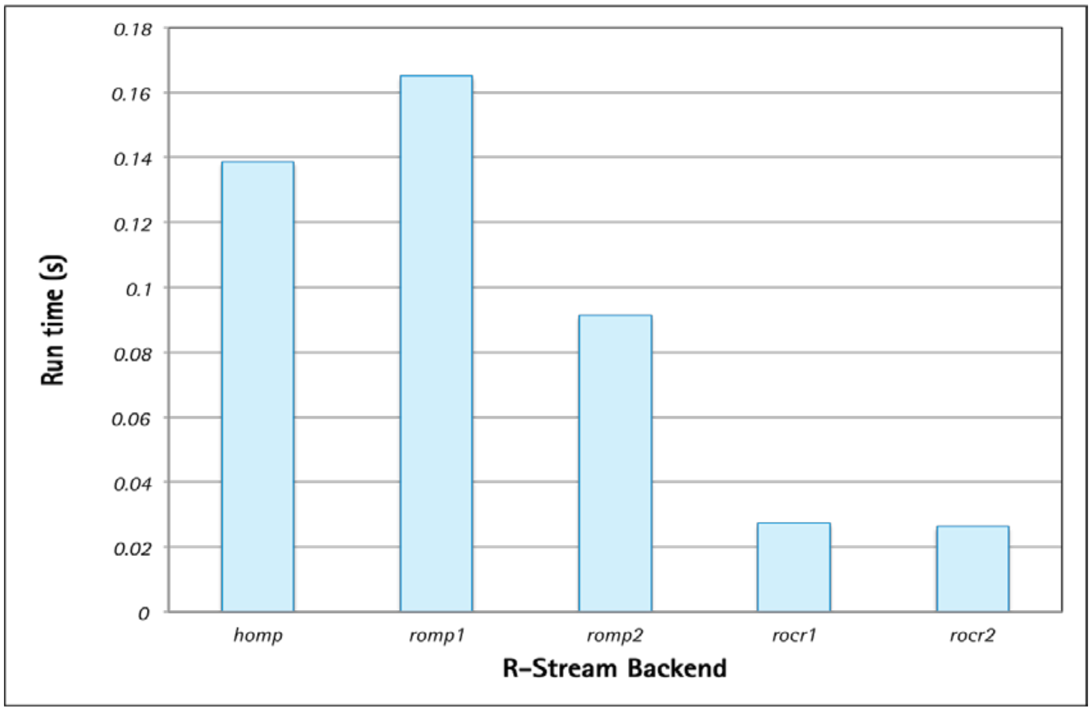


Figure 1: Chebyshev smoother performance on 64^3 array using multiple parallelization techniques. R-Stream OCR shows upt to 5x performance increase vs. hand parallelized OpenMP. homp: hand parallelized OpenMP, romp1: R-Stream OpenMP (loop fusion), romp2: R-Stream OpenMP (loop fission), rocr1: R-Stream OCR (loop fusion), rocr2: R-Stream OCR (loop fission)

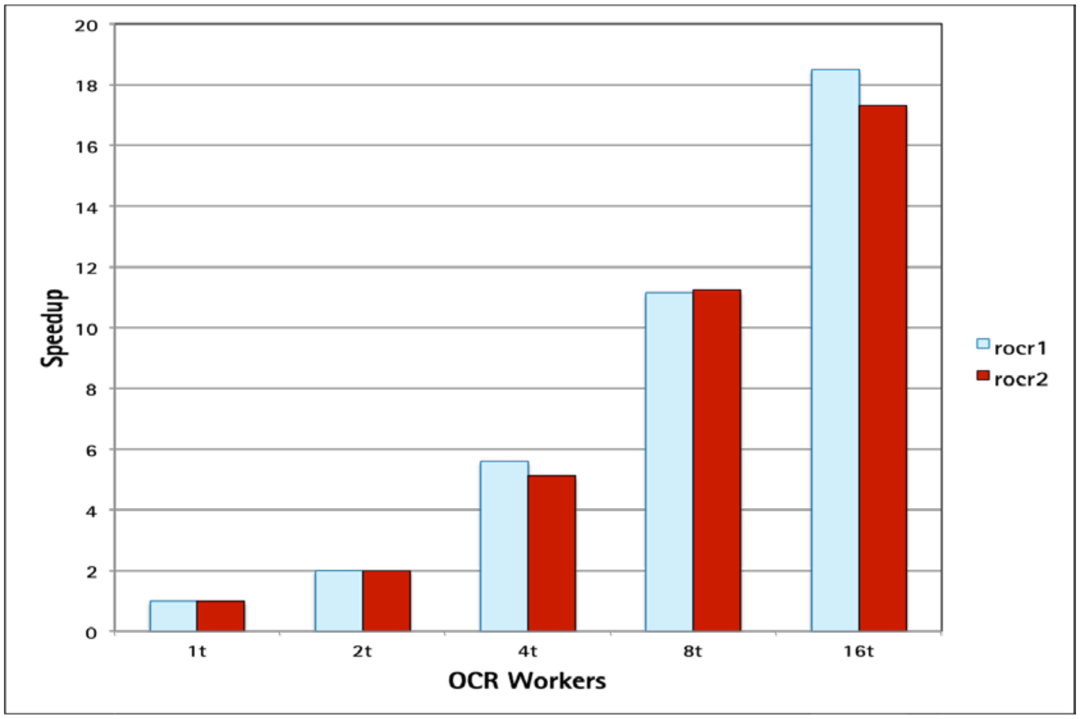


Figure 2: R-Stream OCR scalability. R-Stream OCR shows over 18x performance increase at 16 worker threads.

We also performed experiments that showed superlinear speedups when scaling R-Stream OCR versions of the Chebyshev smoother from 1 to 16 OCR workers, as shown in Figure 2. We generated tiled code and autotuned the tile sizes for each worker count. Superlinear speedup was observed at 4, 8, and 16 workers. This is due to better cache utilization of the tiled code tuned for each worker count.

In total, using R-Stream we generated approximately 8.75 million lines of OCR Chebyshev smoother code consisting of approximately 3500 variants with 2500 lines of code each. The Chebyshev smoother code inputted to R-Stream has less than 100 lines of code.

SDSC Collaboration

We discussed with Laura Carrington's group at the San Diego Supercomputing Center to discuss our individual HPGMG progress and plan of action to plug-in R-Stream optimized components in their hand-written OCR code.

Intel Collaboration

Reservoir Labs has been providing guidance to Gabriele Jost from Intel on HPGMG optimizations (e.g. communication-avoidance optimizations) and usage of R-Stream for simple kernels (e.g. stencil kernels) and HPGMG.

Integration of R-Stream and HTA/PIL

API enhancements

Reservoir Labs has been providing support to Adam Smith from UIUC regarding the interaction of HTA/PIL with R-Stream. In this context, both parties agreed on a new interaction protocol between both tools.

In the context of the collaboration between UIUC and Reservoir Labs, HTA/PIL and R-Stream are combined to parallelize programs using the OCR backend. While HTA is in charge of the coarse grain parallelization, R-Stream parallelizes and optimizes the computations at a finer grain. More precisely, HTA parallelizes the program, targeting coarse grain parallelism, and calls a sequential function to perform the actual computation. Then, R-Stream is used to parallelize the sequential function in order to benefit from the fine grain parallelism available on the cluster. Because the function is initially a regular sequential function, HTA is expecting the function to be synchronous and to only return when the computation is done. Thus, when R-Stream optimizes and parallelizes the function, it has to guarantee this synchronicity and an OCR worker has to be blocked, waiting for all the parallel subtasks to finish. If a single worker is waiting for numerous other running tasks, the performance loss is negligible. However, HTA is not generating a single parallel task on each machine but several of them and each one is calling the function parallelized by R-Stream. As a consequence, several workers end up being blocked, waiting for subtasks. In the worst case, a starvation occurs as all the running OCR workers are waiting for other tasks that cannot run.

The issue has been identified and acknowledged by both parties and we agreed to solve the issue by moving from a synchronous model to an asynchronous scheme. In this new scheme, HTA does not assume anymore that the call to the function parallelized by R-Stream will return only when the computation ends. Instead, it is now assumed that the function is called asynchronously and can return at any time after having scheduled all the parallel subtasks. An event object, created before calling the function, is triggered when all the parallel subtasks are done. This new contract between HTA and R-Stream is expected to improve the performance of the generated programs and will solve the starvation situation described earlier.

Currently, the R-Stream runtime layer used by the OCR programs generated by R-Stream has been modified to allow asynchronous function calls. The event triggering is being developed and will be deployed on the X-Stack cluster after testing.

OCR update

As OCR evolves, new constructs and APIs are being added and existing APIs are being modified. In order to reflect these changes and remain compatible with the latest release of OCR, the R-Stream runtime layer has been updated. This update is particularly relevant in the context of the collaboration between Reservoir Labs and UIUC because the latest version of OCR needs to be used by both HTA/PIL and R-Stream.

Supporting Irregular Computations

We have been extending R-Stream optimizations for handling irregular computations. Specifically, we are working on generating optimized OCR code for the sparse matrix matrix multiply kernel of the CoSP2 proxy application from ExMatEx. Optimizations are being developed to exploit the sparse data structures, reduce the overhead in supporting the irregularity in computations, and improve data locality.

Plans

For the next milestone, we have the following plan of action:

- Delivery of R-Stream optimized HPGMG code (The code is available in the Intel X-Stack git repository)
- Delivery of R-Stream including all performance-scalable optimizations in R-Stream-OCR code generation targeted towards Exascale architecture like TG (R-Stream is made available to all interested users – TG members and DOE users – through an installation in the Intel X-Stack cluster)
- Delivery of final technical report

Issues

None.

Inventions

None.

Publications

None.

Conclusion

During this quarter, we demonstrated the performance and productivity benefits that R-Stream offers to the Exascale software stack through R-Stream generated OCR code for HPGMG kernels. The code is made available to all TG project members through the Intel X-stack git repository. Further, we extended the integration of R-Stream and HTA/PIL to introduce more asynchrony in the R-Stream-HTA integrated programs. Furthermore, we completed necessary task items from R-Stream-OCR perspective to support the public release of OCR.

Pacific Northwest National Laboratory – Andres Marquez

Accomplishments

The ACDT Framework / OCR testbed

In Q11, PNNL worked on improving our distributed OCR (Open Community Runtime) testbed. A list of achievements is presented below:

- Support for Infiniband via the rsocket protocol [REF]
- Support for resource distribution hints
- Redesign of the GUID allocator
- Implementation of a memory coherence protocol for distributed OCR that is almost compliant with the latest OCR spec (v.99a)
- Improved support for machines using the SLURM scheduler
- Inclusion of several code optimizations
- Test of the framework running on 128 nodes (4096 cores) with a selected kernel

Our OCR framework was initially based on TCP sockets between nodes and shared memory within a node. However TCP scalability is limited, so we opted to support a streaming protocol that is written on top of Infiniband's RDMA protocol to help achieve better scalability. The protocol chosen, rsockets, is developed by Intel and is part of the OpenFabrics' librdmacm.

We began working towards providing memory movement hints for kernels/applications that spawn work predominately on a single node. These hints allow OCR internally to create memory on the current node, allocate a GUID for a different node that will point to that memory in the future, and redistribute that memory to the other node while the current process continues to execute. In the future, OCR will redistribute memory across nodes. In support of this, we created a new GUID allocator/address space that is decentralized where every node can allocate a GUIDthat points to any other node.

In our OCR framework, a GUID stores destination node information. The destination node has a routing table pinpointing the owner of that data block. Initially, it will always point to itself. During program execution, data block ownership can change during write operations. Data block ownership discovery during execution is hereby more efficient as compared to an overall network discovery.

Moreover, because data blocks can be written at any time by any node, we needed to develop a memory consitency protocol. This protocol supports OCR's acquire/release memory consistency. The difference between OCR's memory model and "standard" release consistency is that the release semantics are operating on data blocks and those data blocks can only be acquired once per node internally by OCR before that node begins running. Releases can happen at any point, once a node begins executing. So for our protocol, a data block is copied to a node when an EDT requires it as an input. The data block copy exists until it is invalidated. To invalidate a data block, the data block needs to be released or be in a Read Write (RW) or Exclusive Write(EW) mode. If a data block is in those modes, we send an invalidate request to the data block router who in turn sends the requests to anyone who has copies. The router identifies the new owner as the node that sent the invalidate request and asynchronously waits for acknowledgements (ACK) from whoever has currently a copy. Once the router receives all the ACK, it acknowledges the initial requester. Meanwhile, the requester continues executing its EDT. If it encounters any event satisfies during an ACK absence, the requester will buffer those calls and continue executing asynchronously. Once, the ACK comes, it will execute those satisfies. Similar semantics can happen on a release but release functionality has not been implemented yet.

These enhancements to PNNL's testbed allowed us to achieve good scalability on the Cholesky kernel as shown in Figures 1-3. For this kernel, the main EDT creates a static computation graph and partitions the matrix into data blocks. We use hints to redistribute the memory for these EDTs, events, and data blocks across nodes. These initial distribution movements are accounted for in our timing for the benchmarks below: thus these times are not strictly computation time. For each benchmark, we used a 4 socket 32 core AMD Opteron 6272, 32 threads per node, 64 GBs of memory per node, Infiniband, rsockets, GCC 4.8.2, the SLURM launcher, and a 400^2 tile size. The current data presented here is preliminary. We are still in the development process, adding new features and testing all parts of the testbed. The results presented here are a good indication of the progress of the testbed but they require deeper analysis and experimentation.

As a strong scaling test, Figure 1 shows Cholesky performance with a fixed sized matrix of 80000^2 as node count increases.. For each node, we used 32 cores. The max relative speedup we were able to achieve was 81X on 128 nodes. Likely, the performance drop-off from 64 to 128 nodes is due to a combination of processors starving for data and the communication overhead. Figure 2 shows a weak scaling test where we fixed the computation per node to be the same as we scaled the nodes. The graph shows very close to ideal scalability up to 64 nodes. The sudden decrease in scalability from 64 to 128 nodes is likely due to the fact that we are timing the initial memory distribution of the 160K^2 matrix movement as well as other memory movement. The graph of the static OCR Cholesky can have a large memory footprint. Case in point, the matrix alone is 204GBs of memory, not including the EDT or event memory. Our initial assumptions lead us to believe that Infiniband saturation and congestion cause the startup of EDTs to be delayed. Finally, Figure 3 just computes the gigaflops based on Figure 2's timing information. The current numbers reflects a Cholesky kernel that has not used any optimized libraries (e.g. MKL) for its internal operations, which will increase the performance further. Nonetheless, we have significantly improved OCR's performance for distributed systems.

Thanks to all these enhancements and features, we have vastly improved distributed OCR compared to our initial implementation. Future work includes adding finish EDTs and exclusive writes into the distributed OCR, finishing up the distributed ACDTF (Architected Composite Data Types Framework), bug fixes, optimizations, and a public release.

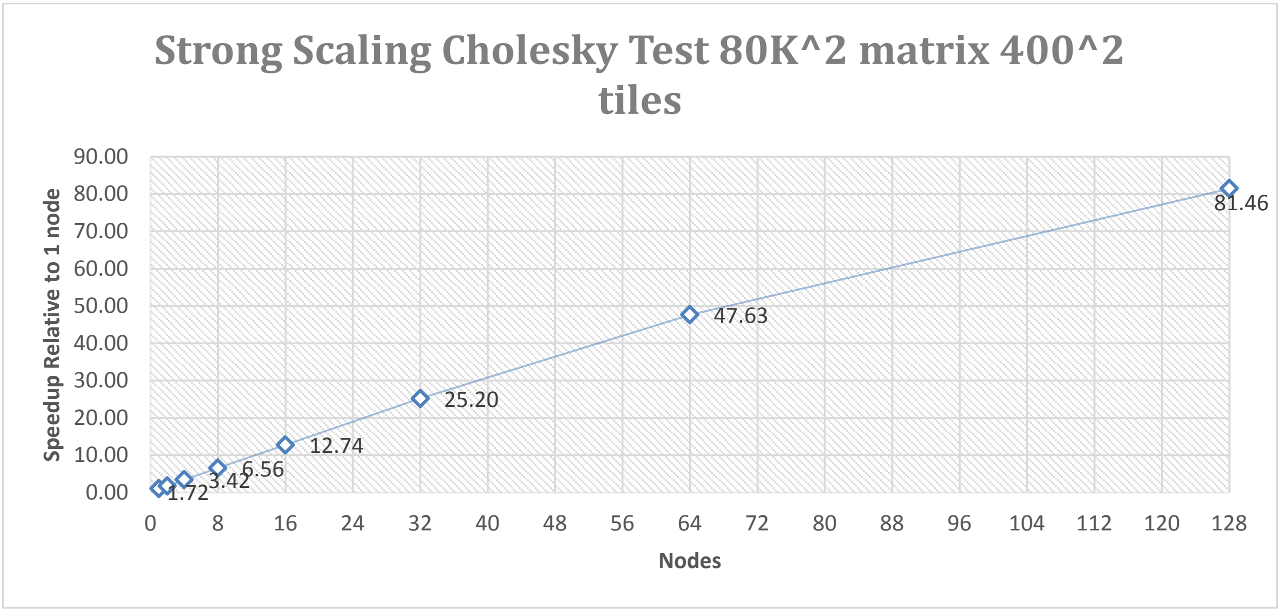
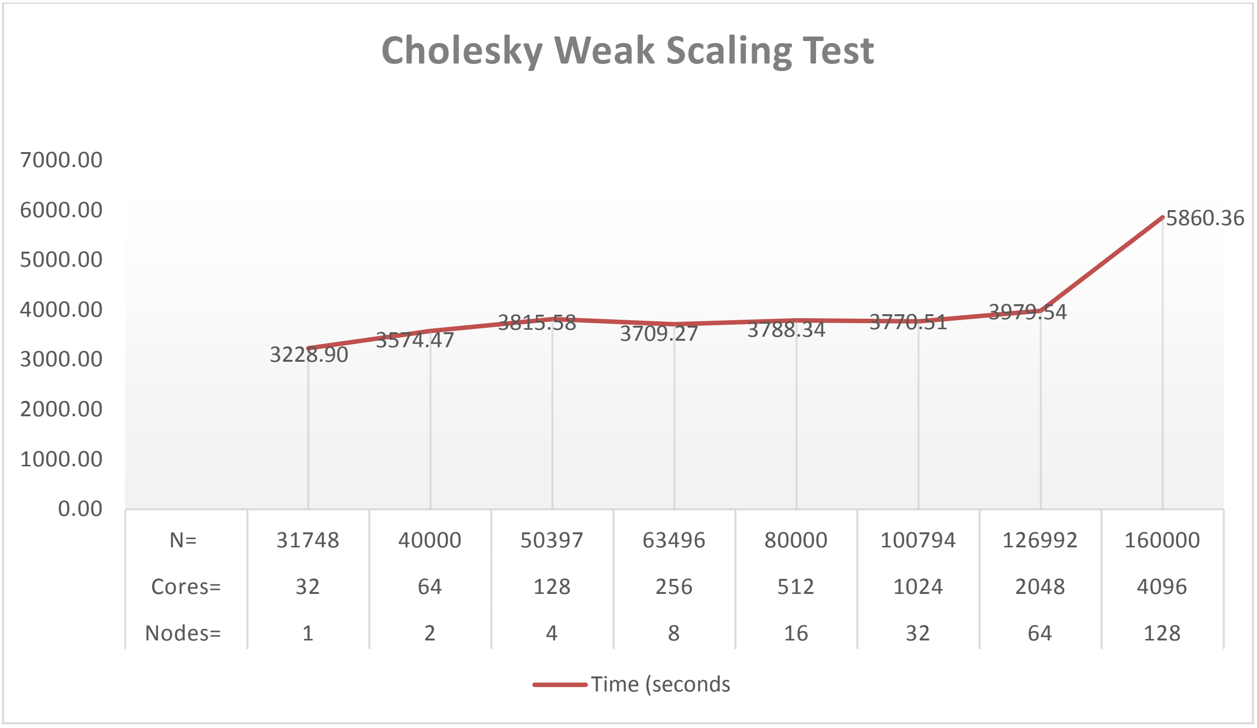


Figure 3: The machine has 32 Cores per node for a total of 4096 cores: used 32 threads per node and tile size = 400^2



N=	31748	40000	50397	63496	80000	100794	126992	160000
Cores=	32	64	128	256	512	1024	2048	4096
Nodes=	1	2	4	8	16	32	64	128

Figure 4: Used 32 threads per node and tile size = 400^2; Amount of floating point operations per node = (160000^3/3)/128

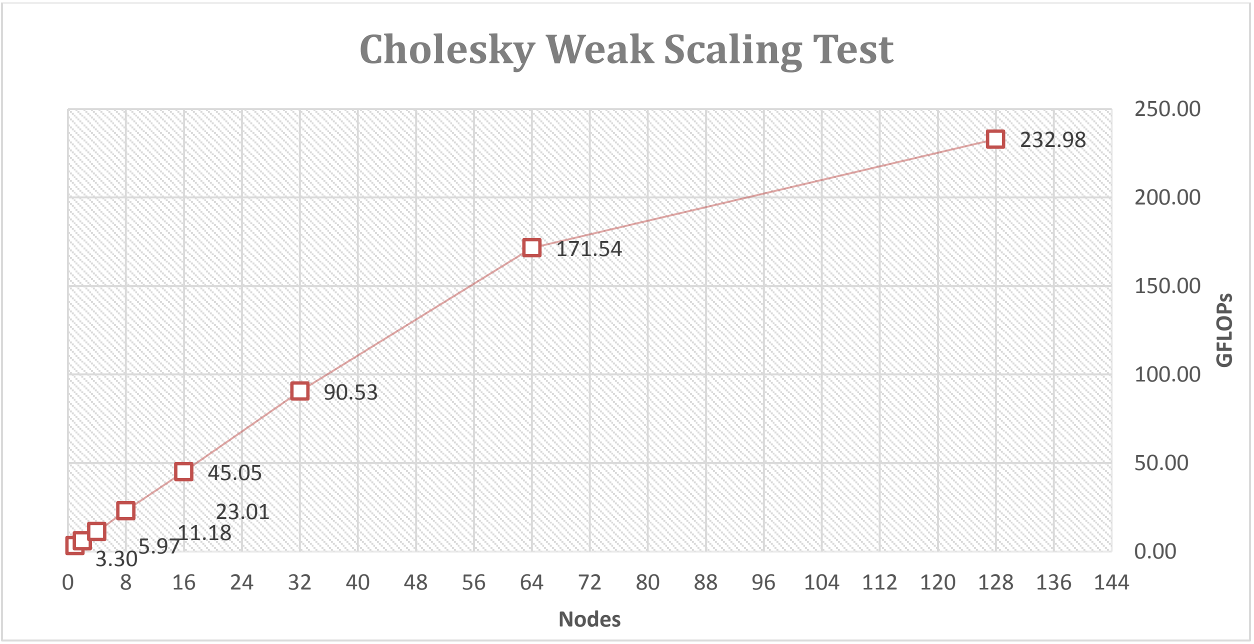


Figure 5: Used 32 threads per node and tile size = 400^2; Amount of floating point operations per node = (160000^3/3)/128. See Figure 2 for matrix size per node and core count.

Applications development

HPGMG

We have made significant progress towards building a CNC version of the High Performance Geometric MultiGrid (HPGMG) software here at PNNL in Q11. Adapting any complex code base such as this one to a different paradigm takes careful design and planning to ensure that we do not introduce avoidable artifacts that can prove costly in performance. To this end, the PNNL team has extensively mapped the code flow of HPGMG and the data structures on which it relies. The high-level processes and data dependencies were then refactored to fit more closely with the OCR model of EDTs (event driven tasks), From this we developed a CNC graph file, see Figure 4, for HPGMG that fits the CNC-OCR translator. Using this graph file we were able to validate that our approach and code produces a correct execution of the data flow for a 'v' cycle, one of HPGMG's execution paths. The next step in completing the code conversion was to refactor HPGMG's internal data

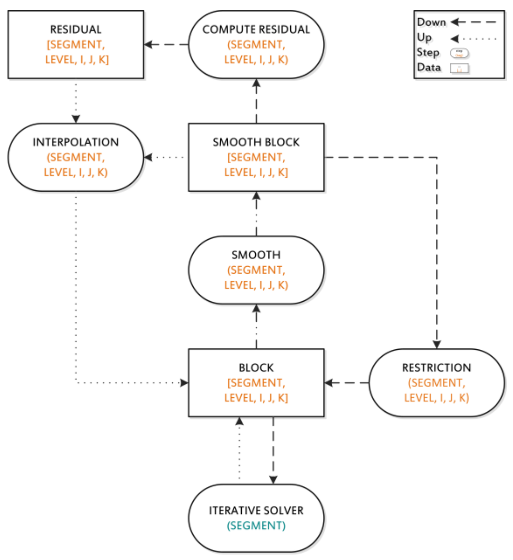

RESIDUAL
[SEGMENT, LEVEL, I, J, K]
COMPUTE RESIDUAL
(SEGMENT, LEVEL, I, J, K)
Down
Up
Step
Data
INTERPOLATION
(SEGMENT, LEVEL, I, J, K)
SMOOTH BLOCK
[SEGMENT, LEVEL, I, J, K]
SMOOTH
(SEGMENT, LEVEL, I, J, K)
BLOCK
[SEGMENT, LEVEL, I, J, K]
RESTRICTION
(SEGMENT, LEVEL, I, J, K)
ITERATIVE SOLVER
(SEGMENT)

Figure 7: HPGMG CnC Graph

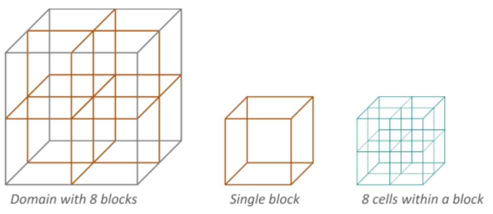


Figure 6: HPGMG domain structure

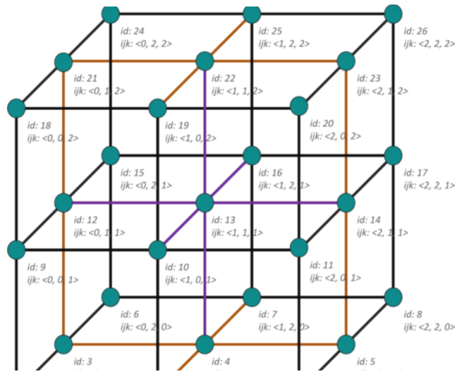


Figure 6: LULESH node id and ijk

structures, see Figure 5. The approach we are employing includes building tiling in as a parameter rather than a thread-dependent value. This will help facilitate robust testing of the code over various scales but comes at the cost of a more involved design and implementation refactor. Our use of the EDT model also allows for complete parallelization down to the solve step, so data dependency of the solver at the coarsest level is the only bottleneck in the code. In conclusion, we have completed the design steps necessary to confidently approach the creation of the code for the CNC version of HPGMG. From this we have implemented the necessary steps to execute a 'v' cycle and have made substantial progress merging in the refactored data structures. Our future work will be to finalize the refactor and test our implementation.

LULESH

The CnC-OCR translator is a bit of a moving target as it develops, advancing in features and functionality. In order to keep LULESH running smoothly and taking advantage of the latest and greatest the tool has to offer, we spent some time in Q11 updating our code to be compliant with the latest version of the translator. The updated version has been pushed into the X-Stack repository and is available in the apps directory.

To help support recent auto-tiling research, effort was also made to refactoring the indexing in LULESH. LULESH has two main data structures, elements and nodes. Both structures are currently indexed using an integer, *node_id* and *element_id*. From these id's neighboring id's can either be calculated or looked up in a global lookup table, both options come at an additional cost. One way to reduce this cost is to store and reference the nodes and elements using a three dimensional i, j, k tag, see Figure 6. Calculating neighbors then becomes an easy task of adding or subtracting one in any of the dimensions. This increases the dimensionality of the tag components and requires a restructure of many of the LULESH data structures. The CnC graph representing LULESH has been updated to accommodate the additional information in the tags and we are currently working on refactoring the data structures to match.

Inventions

None.

Publications

None.

Rice University - Vivek Sarkar

Accomplishments

- OCR Release 1.0. This release features implementations of OCR on Distributed memory systems (using both MPI and GasNet as the communication layers), a refactoring of the OCR on x86 reference implementation to support a more modular and platform-agnostic runtime, OCR on FSIM implementation (in collaboration with Intel). Functionality and performance debugging, refactoring of the code and feature completion from OCR 0.9.

- Continued efforts in design, development and maintenance for OCR shared- and distributed-memory implementations.
- Focused effort on understanding and improving performances of the distributed-memory implementation through a performance-oriented micro-benchmarking framework.
- OCR implementations of applications or kernels of interest to DOE (developed in collaboration with research partners within the project):
    - npbCG
    - CoMD
    - LULESH
    - FFT
    - SAR
    - Gups
    - UTS
    - hpGMG
- OCR documentation: OCR Spec 1.0 (Collaboration with Intel) has been completed and will be publicly available with OCR Release 1.0. OCR. User Documentation and OCR Developer Documentation have also been actively expanded and are available on the xstack wiki. Contributions to the OCR specification discussions and writing.
- OCR memory model. Collaboration with Intel and others on the project on the definition of the memory model for OCR that will be part of the 1.0 Release.
- CnC on OCR implementation. This implementation features refactoring of the CnC programming model to support an event-driven environment such as OCR. CnC on OCR on x86, CnC on OCR on distributed memory and CnC on OCR on FSIM are all implemented and tested.
- A paper on CnC tuning to ICS 2015 currently under review.
- A paper on Habanero features based on OCR implemented on a heterogeneous ARM + DSP platform submitted to HPAC 2015.
- An implementation of HPCToolkit tailored for OCR and CnC-OCR currently under way. This will enable performance profiling of OCR and CnC-OCR programs.
- Continuing collaboration with PNNL on a CnC design and implementation of HPGMG (see the PNNL section).
- Collaboration with Purdue University on automatic tiling and step fusion of CnC programs.

Plans

- Further performance improvements and bug fixes to the OCR and CnC-OCR implementations
- Use the HPCToolkit runtime profile information as a feedback for runtime adaptation.
- LANL-Rice meeting on June 10th to design and implement a matrix/matrix multiplication algorithm in CnC-OCR

Issues

None.

Inventions
None.

Publications
"Declarative Tuning for Locality". Sanjay Chatterjee, Nick Vrvilo, Zoran Budimlić, Kathleen Knobe, Vivek Sarkar. Submitted to SC15.

"Heterogeneous Work-stealing across CPU and DSP cores". Vivek Kumar, Alina Sbîrlea, Zoran Budimlić, Deepak Majeti and Vivek Sarkar . Submitted to HPAC 2015.

UCSD - Laura Carrington

Proposed milestone
Continue experimenting with CoMD on FSIM to compare different versions of CoMD's energy efficiency on the different components of the TG architecture (e.g. Memory, Network, and instructions).

Continue to work on CoMD/OCR version 2.

Test our OCR HPGMG code on FSIM target (build and run) and evaluate performance and energy efficiency. Continue our analysis on performance characteristics of HPGMG code on x86-pthreads-x86 target.

Progress
The code is checked in the repository, and we are checking some transient failures in the integration of HPGMG with the regression testing framework. The HPGMG code supporting FSIM will be checked into the repository after the regression testing issue is resolved.

CoMD

We ran four variants of CoMD on OCR/FSIM to collect energy numbers. These are preliminary results but show the methodology that we are pursuing, by which we compare different variants of an algorithm from a performance and efficiency perspective. We compared CoMD with duplicate force calculation, pseudo-atomic updates, exclusive DB access, and the CnC version. The three plots in Figure 1, Figure 2, and Figure 3 show the comparison, divided by hardware component, for dynamic energy consumed by instructions, memory, and the network. Note that due to performance penalty of CnC version relative to hardcoded OCR versions the energy efficiency of CnC version is significantly worse than hardcoded OCR. Also the figures show that the exclusive DB access version is typically the most energy efficient version of CoMD.

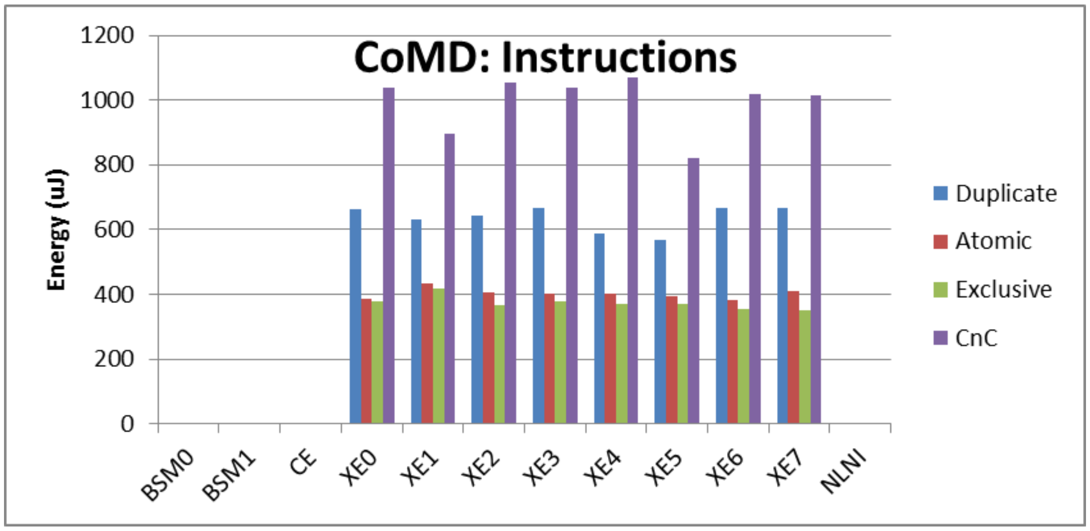


Figure 8. Comparison of energy for instructions on TG architecture using different versions of CoMD.

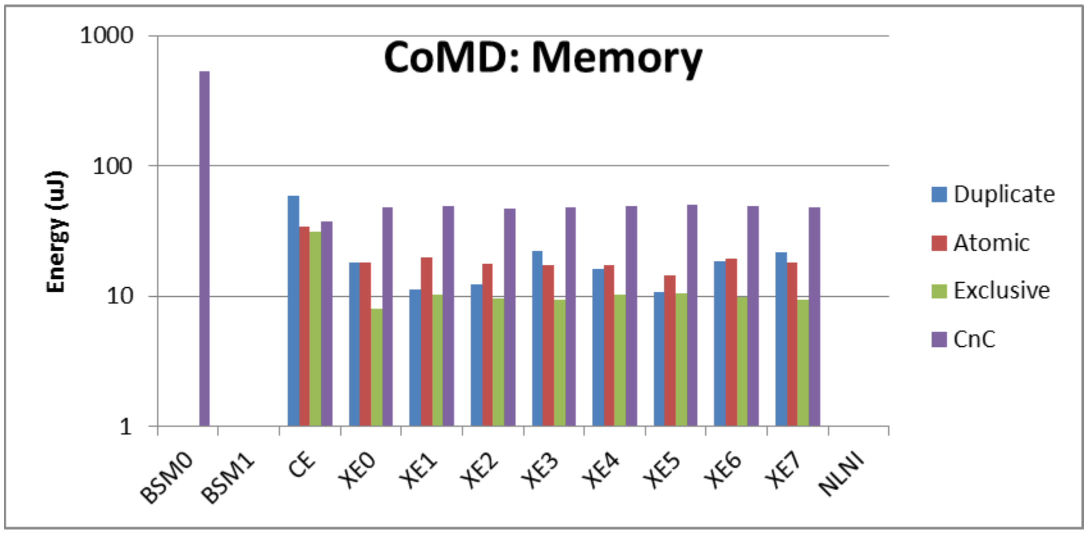


Figure 9. Comparison of energy for the memory on TG architecture using different versions of CoMD.

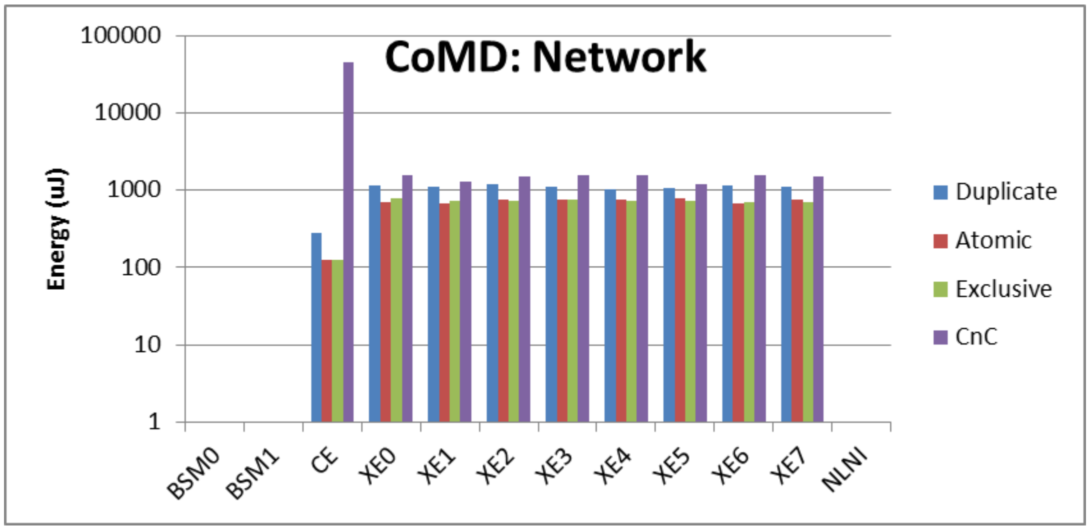


Figure 10. Comparison of energy for the network on TG architecture using different versions of CoMD.

HPGMG

We ported HPGMG on OCR-FSIM and were able to run some small problem sizes. We are currently trying to scale. We are also checking and make sure that it is integrated and working within the regression testing framework.

Issues and Limitations Encountered

Bugs encountered in OCR have been solved or are under evaluation; no major outstanding blocks to report. Other scaling limitations and performance issues on HPGMG and CoMD will be revised with new versions of the OCR allocators. Failures have been reported in the regression testing of HPGMG but UCSD has not been able to reproduce or observe such failures. An accurate timing model for FSIM will be necessary to make performance and power comparisons.

Next steps

Continue experimenting with CoMD on FSIM and compare different variants for their performance and energy efficiency using 4.1.0 ISA.

Do more testing of the OCR HPGMG code on FSIM target and evaluate performance and energy efficiency. Compare HPGMG/OCR and HPGMG/Rstream/OCR. Explore integration of our HPGMG/OCR with Rstream's OCR smoother.

University of Illinois Urbana Campus – David Padua

Accomplishments

Evaluation of HTA Implementation

Below is a graph of the execution time of the NAS benchmark CG. There are two implementations of the code in HTAs as discussed in detail in previous reports. The first is a fork/join version and the other is a SPMD version. We have five other NAS benchmarks implemented: EP, IS, FT, MG, and LU. We show only CG here as a representative example.

There is a single HTA implementation programmed using the Parallel Intermediate Language (PIL) that can generate code for multiple parallel runtimes. Here we use the same HTA implementation to generate OpenMP code in addition to OCR code.

All results shown here were gathered on the FooBar cluster with dual Intel Sandybridge E5-2690 processors with 16 cores and 32 threads and 128 GB of memory. The blue line shows execution performance of an efficient pure OpenMP implementation of the algorithm. The green line shows OpenMP code generated from the HTA implementation. The magenta line shows OCR code generated from the HTA implementation. The gray dotted line is the performance of the best known sequential algorithm. This sequential algorithm is used to compute speedups.

Fork/Join

The fork/join model uses global barriers for synchronization. This assures that any data computed before the barrier is synchronized with *all* threads that *may* need the data. Global barriers are expensive when there are a large number of tasks or there is a load imbalance between tasks.

In Figure 1 we can see the execution time and speedup of our fork/join HTA implementation. It can be seen that the HTA generated OpenMP code scales nearly identically to the handwritten OpenMP version. It can be seen that when exceeding the 16 cores and using 32 threads we see no benefit since the algorithm is sufficiently compute bound. The OCR generated code shows some overhead, but still scales up to 16 PEs.

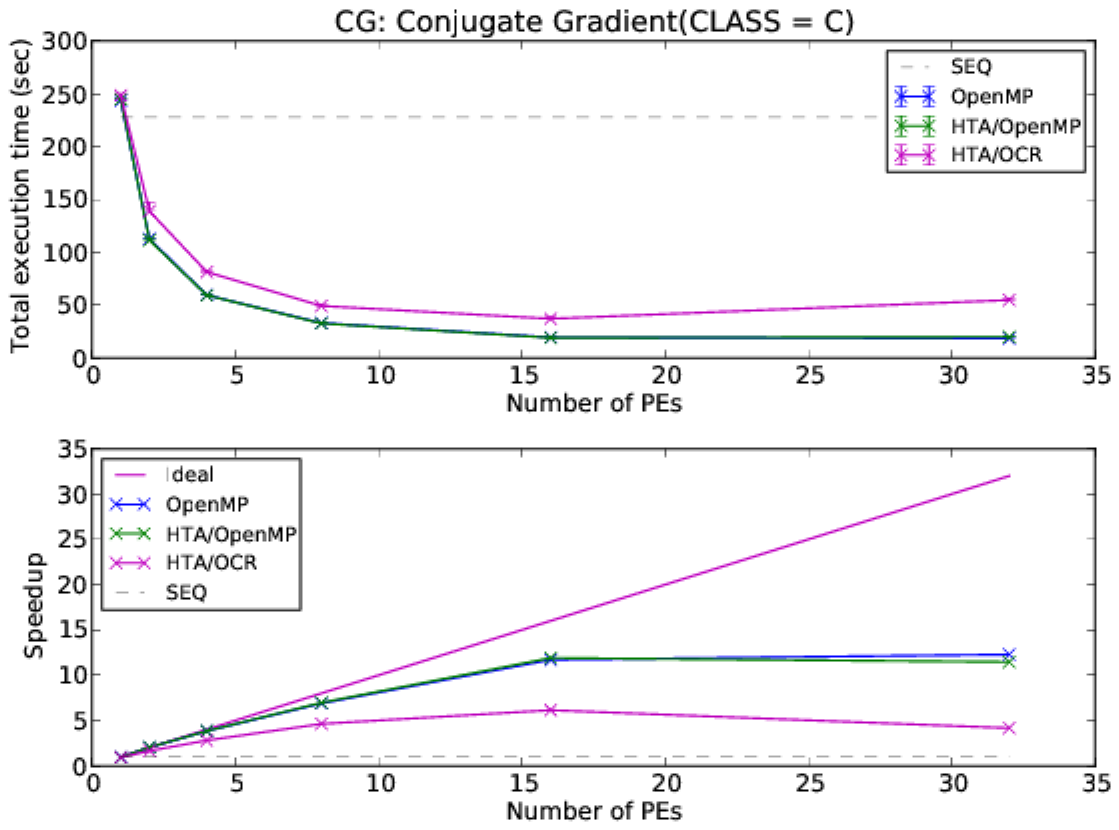


Figure 1: CG fork/join execution time (top) and speedup (bottom).

SPMD

The SPMD model removes barriers and uses point-to-point synchronization. This means that tasks only synchronize with each other when the *need* data that the other has.

In Figure 2 we can see the execution time and speedup of our SPMD HTA implementation. Once again, the performance of the OpenMP generated HTA code is very similar to the highly tuned handwritten OpenMP code. The OCR generated code scales until 8 PEs, but significant overheads are incurred at 16 and 32 PEs. Once again, scaling beyond the 16 cores to use 2 threads per core shows no benefit for any implementation.

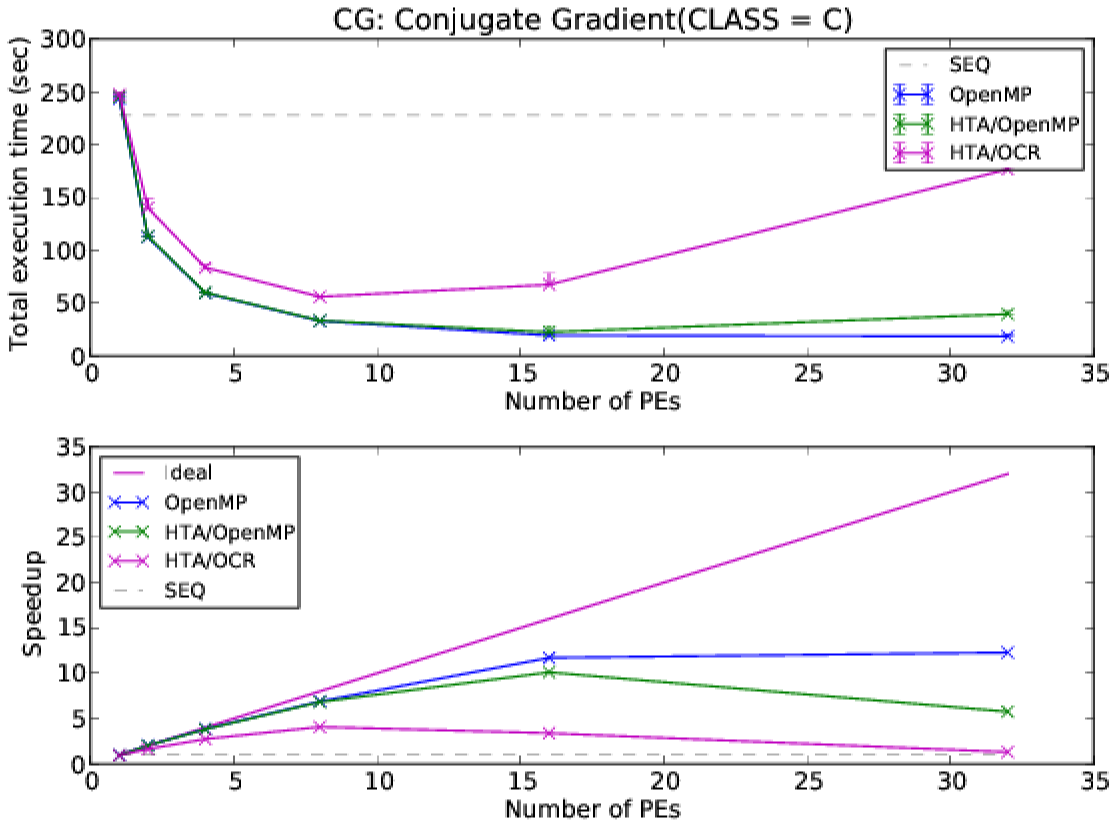


Figure 2: CG SPMD execution time (top) and speedup (bottom)

HTA and R-Stream Integration

We have made significant progress on the integration of HTAs with the R-Stream compiler. This work is ongoing. Current status shows that the integration is complete when constraining parallelism due to a limitation in the interface. This issue is addressed in the Issues section below. Further work will be required to complete this integration.

Issues

Evaluation of HTA Implementation

We have discovered that the overhead of firing an EDT in OCR is greater than anticipated. In the NAS benchmarks discussed above, the problem size is clearly defined. In the strong scaling experiments discussed above, that means that as the number of PEs in increased, the tile size decreases. This means that the work done within a single EDT is reduced as the number of PEs is increased, thus increasing the effect of the overhead of creating and firing the EDT. We have performed other experiments that confirm this hypothesis by synthetically setting the amount of work to be done within a single EDT. As the amount of work within a single EDT is increase, the overhead of the system goes down. This is not surprising. What is surprising is that the amount of work that needs to be done within a single EDT is quite large. We will need to work closely with the OCR team to make sure that these overheads can be addressed and that OCR can meet its goal of managing *lightweight* asynchronous tasks.

HTA and R-Stream Integration

We have discovered that the way that the R-Stream compiled functions are being called by the HTA side are causing problems for the OCR runtime. Specifically, the functions are being called synchronously. This causes the EDTs to block and wait for the return of the R-Stream function. Since EDTs are non preemptable, the blocking EDTs consume all cores and the program deadlocks. We are working on a fix to make the functions asynchronous and call a callback EDT in the HTA side when finished. This will fix the current issue and complete the integration.

Goals for Next Quarter

HTA and R-Stream Integration

We will complete the work as outlined above.

Graph Extensions for HTAs Implementation

The implementation of the graph extensions for HTAs is taking longer than expected. We will continue to work on this in Quarter 12.

Graph Extensions for HTAs Evaluation

Once the implementation of the graph extension is complete, we plan to provide an evaluation of the implementation with a Single Source Shortest Path (SSSP) algorithm, as outlined in the Quarter 9 report.

University of Illinois Urbana Campus - Josep Torrellas

Accomplishments

In this quarter, Wooil Kim and I performed an initial evaluation of the TG architecture, including memory management and power, on FSIM under OCR 4.0.8. The system is not yet totally debugged for performance, although the energy numbers are now reliable. There are a few bottlenecks and inefficiencies that are currently being fixed. In particular, the inactive XEs and CEs are left spinning rather than being completely power gated. As a result, they continue executing instructions and consuming energy. We expect that these issues will be solved in the coming month, as we move to OCR 4.1.0.

Issues

We need to work with the rest of the team members so that FSIM v3.0 - 4.1.0 is fully debugged and we can provide more detailed performance and energy statistics. We expect this will be done this coming month.

Plans for next milestone

Complete evaluation of the entire architecture, including memory management and power, on FSIM v3.0 - 4.1.0.

Inventions

None.

Publications

None.

Appendix:

Initial Evaluation of FSIM under OCR 4.0.8

Woil Kim and Josep Torrellas

In this evaluation, we consider three parallel applications, namely, Cholesky factorization, Smith-Waterman, and FFT, and examine the scheduling of Event Driven Tasks (EDTs), the allocation of memory blocks, and the energy consumption. Smith-Waterman is DNA sequence alignment algorithm.

1. EDT Scheduling

In the OCR programming model, the programmer specifies the dependences between tasks in the program, and the runtime system schedules the tasks in a way that maximizes the concurrency. This approach delivers higher concurrency than in traditional programming models, where the programmer also has to specify the parallelization---e.g., through barriers. The end result is that OCR's approach can deliver higher speed-ups.

To see why, consider first Cholesky factorization. We start by decomposing Cholesky into EDTs. Figure 1 shows the EDT structure and the data dependences. Each EDT operates on a tile. The top row shows the EDTs executed in iteration i; the second row shows the EDTs executed in iteration i+1. In each iteration, there are three types of EDTs. EDT Type 1 is the single tile in blue in the leftmost plot of each iteration. It is called the Sequential Task EDT and needs to execute first. EDT Type 2 are the blue tiles in the central plot of each iteration. They are called the Trisolve Task EDTs and can only execute after the Sequential Task EDT completes. Finally, EDT Type 3 are the blue tiles in the rightmost plot of each iteration. They are called the Update Non-Diagonal and Update Diagonal Task EDTs, and can only execute after the Trisolve Task EDTs complete.

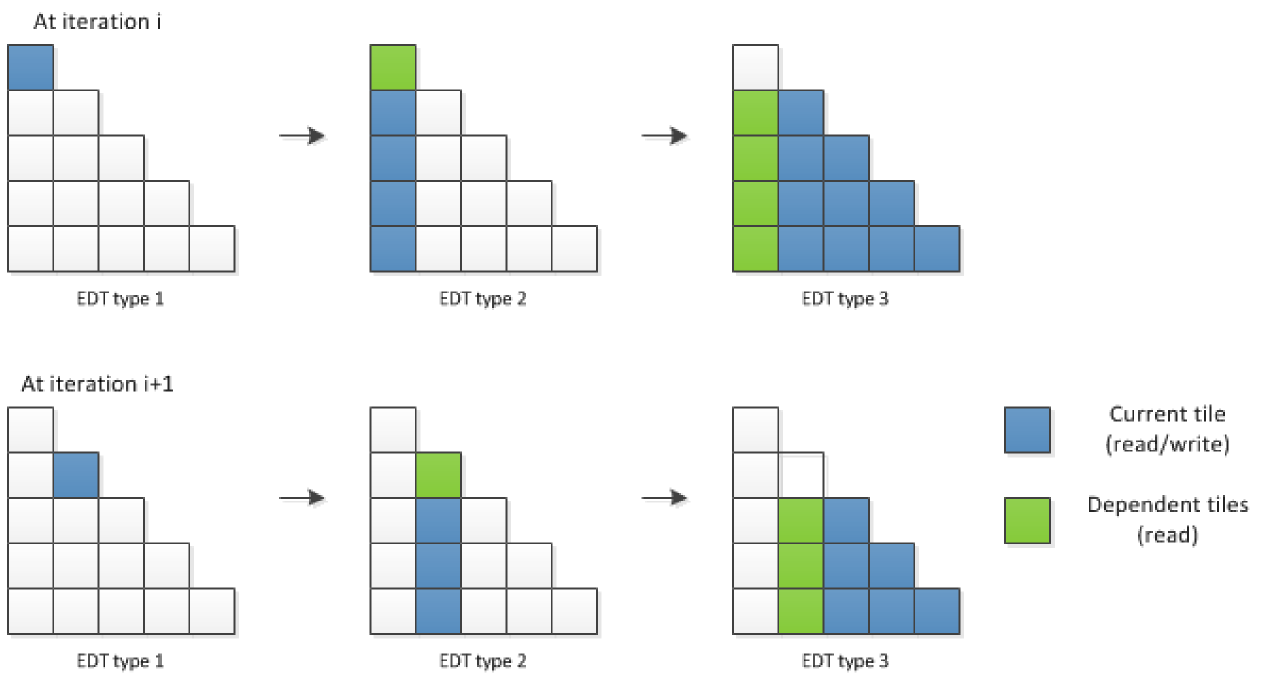


Figure 1. Dependences between EDTs in Cholesky factorization over iterations.

In traditional barrier-based parallelization, we would first execute the EDTs of iteration i according to the dependences explained. Then, we would execute a global barrier. Only after that, we would proceed to execute the EDTs of iteration i+1.

In reality, we can see that the EDT of a tile in iteration i+1 does not need to wait for the EDTs *of all the tiles* in iteration i to complete. Instead, a tile in iteration i+1 can be processed as soon as all those it depends on (in iterations i and i+1) complete. This is what OCR enables in a manner that is transparent to the programmer. With OCR, we can overlap EDTs from different iterations, as long as the dependences between individual tiles are respected. For example, Figure 2 shows our measurement of FSIM running Cholesky with data size 500 and tile size 100, on 1 block with 4 XEs. The figure shows the timeline of which EDT is executing on each XE. The color code is as follows: the first gray block is the MainEDT, which creates all other EDTs, and sets dependences between them. Green blocks are Sequential Task EDTs, blue blocks are Trisolve Task EDTs, red blocks are Update Non-Diagonal Task EDTs, and purple blocks are Update Diagonal Task EDTs.

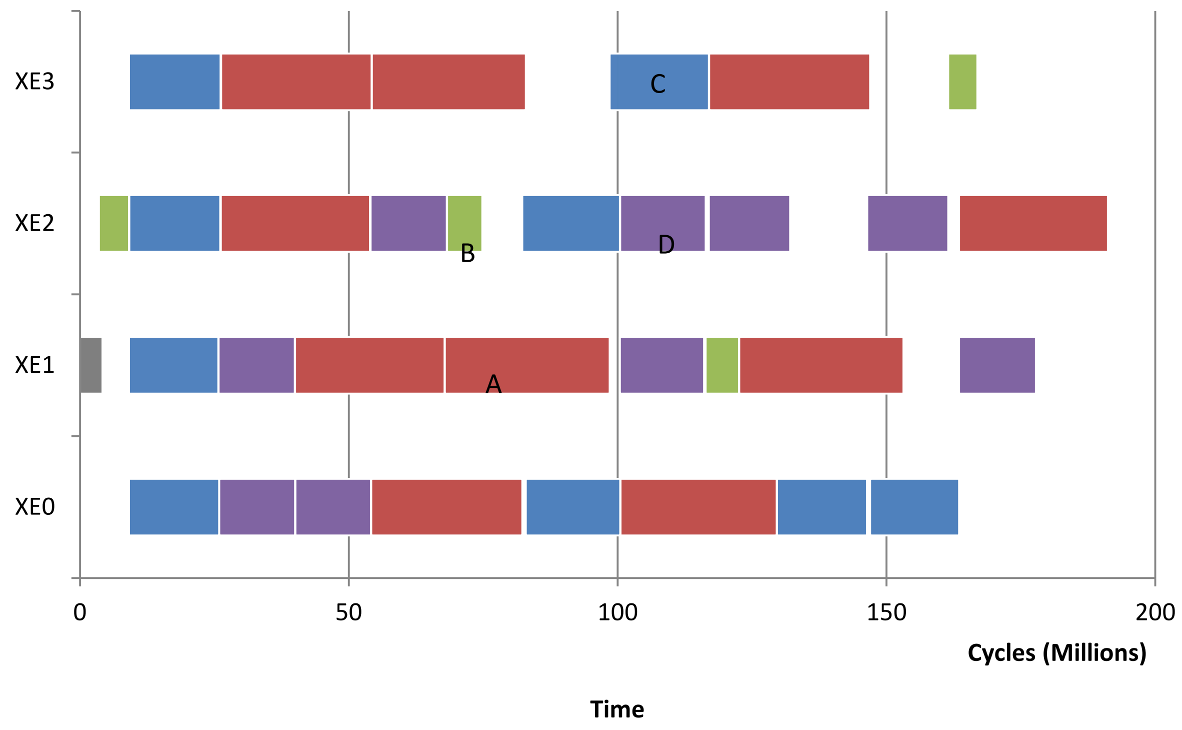


Figure 2. Cholesky EDT scheduling under OCR in a block with 4 XEs.

The figure shows that EDTs do not wait for all the EDTs from the previous iteration to complete. In the figure, EDT A is an Update Non-Diagonal EDT from iteration 1, and EDT B, is a Sequential EDT in iteration 2. These two EDTs do not have any data dependence, even though they are in different iterations. Consequently, under OCR, they execute concurrently. The same occurs between independent EDTs in a single iteration. Foer example, EDT C is a Trisolve EDT from iteration 2 and EDT D is a Update Diagonal EDT from iteration 2. They happen not to have data dependences, and so OCR executes them concurrently. The result is more concurrency than under traditional barrier-based parallelization.

Consider now the Smith-Waterman program. Figure 3 shows the dependence structure between EDTs. The EDT for a tile has dependences on the EDTs for the tiles in NW, W, N directions. In the figure, this is shown as the blue tile depending on the three green tiles. In turn, the tile processed in the current EDT creates data that is used in the EDTs for the tiles in the S, SE, E directions. This creates a wave-front style parallelism that leads to a maximum degree of concurrency of N when NxN tiles are available.

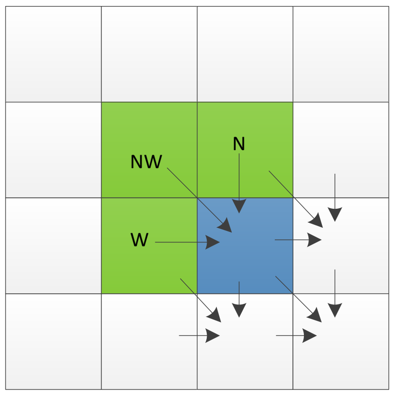


Figure 3. Dependences between EDTs in Smith-Waterman.

A typical parallelization with a barrier leads to the parallel execution profile shown in Figure 4. EDTs are scheduled on available XEs, but the degree of parallelism is limited. Suppose that we have 4 XEs. In this case, we can only keep the 4 XEs busy in one of the inter-barrier steps.

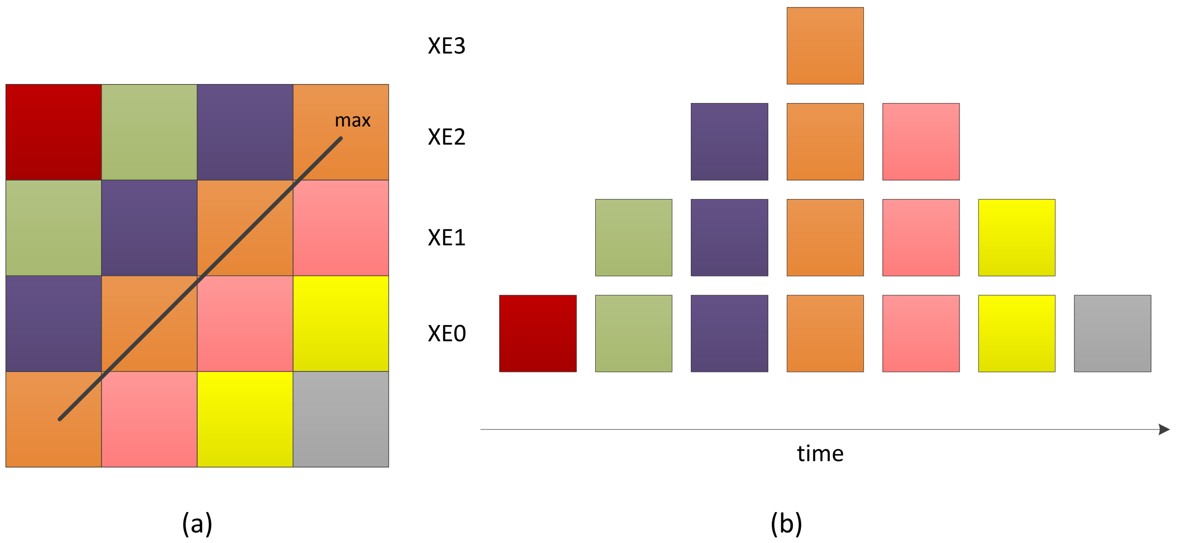


Figure 4. Concurrent execution of EDTs with diagonal parallelization (a), and its expected scheduling on XEs (b).

With OCR, we attain a better parallelism profile. We used FSIM to model the execution on 1 block with 4 XEs. The resulting timeline of the Smith-Waterman for 4x4 tiles is shown in Figure 5. We see that the execution is much more concurrent, and more XEs are busy on average. In Figure 4, only tiles of the same color can run in parallel, and each color group is separated using barrier synchronization. In Figure 5, orange EDTs start execution before all the purple ones finish. Also, pink EDTs extend the time for which 4 XEs are running. With more overlapped execution, OCR enables better exploitation of fined-grained parallelism. The only effect that limits the concurrency is the true dependences between the EDTs.

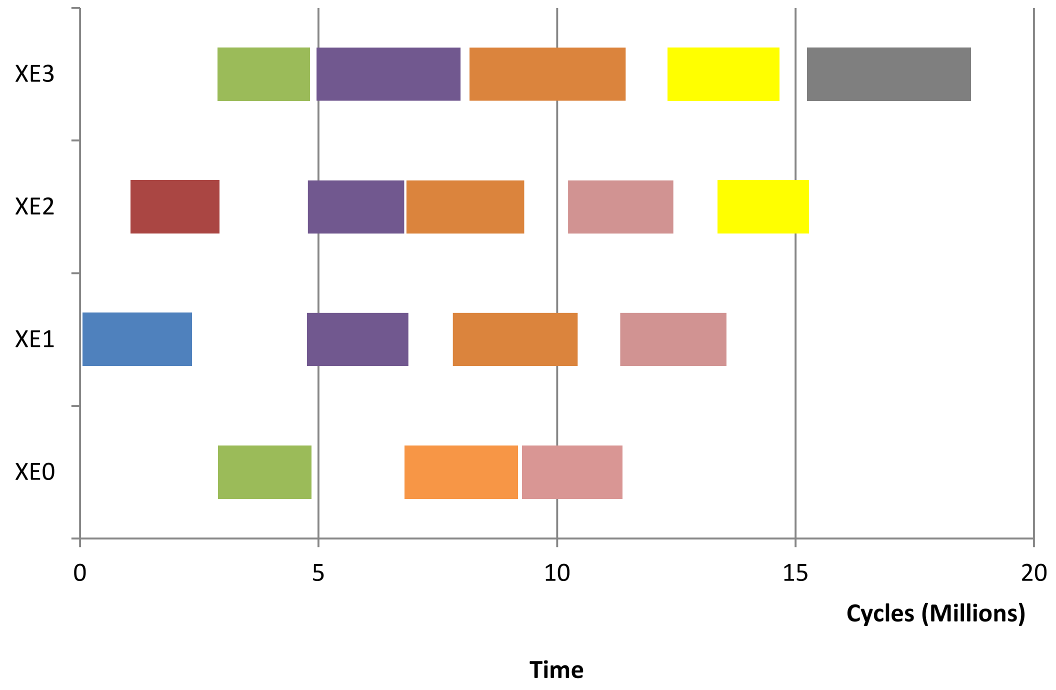


Figure 5. Scheduling of Smith-Waterman with 4x4 tiles on a block with 4 XEs.

We also observed that the EDT execution is load-balanced. In this experiment, we use FFT, which divides computation into many EDTs. We run FFT with size 2^20 on a block with 8 XEs and measure the number of EDTs executed in each XE. The result is shown in Table 1. We can see from the table than the number of executed EDTs in XEs is well-distributed, leading to good load balancing.

	XE0	XE1	XE2	XE3	XE4	XE5	XE6	XE7	total
# of EDTs	43	56	52	43	48	51	47	46	386

Table 1. Number of EDTs executed in each XE for FFT of size 2^20 running on a block with 8 XEs.

2. Multi-level Memory Allocation

The TG architecture has a multi-level memory hierarchy and OCR needs to exploit it to improve performance. The goal is to enhance locality by allocating a block of data in the memory layer physically close to the XE that will operate on the data. Previous versions of OCR required programmers to explicitly declare where to allocate memory blocks. Unfortunately, since scratch-pad memories are limited in size, programmers have difficulty deciding where to allocate blocks, leading to more memory blocks being allocated in DRAM---which can accommodate all data blocks. However, allocating memory in DRAM for any data size leads to very inefficient programs in terms of performance and energy efficiency. But allocation to the memory layers closer to the XE requires programmer's careful management of scratch-pad memories, and often results in runtime errors due to overflow of scratch-pad memories.

The current version of OCR solves this problem to some extent, and gets one step closer to allocating memory for locality. Specifically, the current OCR allocates memory blocks at different places depending on the size of the data block. This simplifies programming. There are three levels of memory hierarchy currently supported: *local scratch-pad*, *BSM* (block shared memory), and *DRAM*. Note that, in the current version, the *local scratch-pad* corresponds to the CE's scratch-pad, not the allocating XE's scratch-pad. This will be solved in the in next OCR version. The sizes of the simulated local scratch-pad and BSM are 638KB and 2x1280KB, respectively.

In this next FSIM experiment, we consider Smith-Waterman for two different tile sizes running on a block with 8 XEs. In one run, we use a 50x50 data tile for each EDT. This is too large for the scratch-pad but small enough to be automatically allocated in the BSM. In the second run, we use a 800x800 data tile, which is too large to fit in the BSM and is automatically allocated in the DRAM. For this tile size, the program runs sequentially on a single XE.

Figures 6 and 7 show where the remote load instructions (load64.rem) of each XE are serviced from. The locations can be CE scratch-pad, BSM, or DRAM. Since scratch-pad style memories do not move data automatically, the location where the data block is allocated determines where each load gets the data from. Figure 6 shows that, for the small tile size, all load64.rem accesses from all XEs get the data from the BSM. On the other hand, Figure 7 shows that, for the large tile size, the load64.rem accesses of the only XE busy do not come from BSM but from the DRAM. There are also many accesses that come from the CE scratch-pad, possibly because the program is sequential. Unfortunately, accessing the tile from DRAM is very costly in performance and energy. The result of allocating memory in DRAM is longer execution time and higher energy consumption. We can see that proper data tiling in OCR is important. It allows the use of closer memory layers.