DoE Award #: DE-FC02-12ER26099/DE-SC0008596
Award Institution: University of Houston

eXascale PRogramming Environment and System Software (XPRESS)
PI: Barbara Chapman, University of Houston
Co-PI: Edgar Gabriel, University of Houston

Final Report

Date: November 30, 2015

1 Synopsis

Exascale systems, with a thousand times the compute capacity of today's leading edge petascale computers, are expected to emerge during the next decade. Their software systems will need to facilitate the exploitation of exceptional amounts of concurrency in applications, and ensure that jobs continue to run despite the occurrence of system failures and other kinds of hard and soft errors. Adapting computations at runtime to cope with changes in the execution environment, as well as to improve power and performance characteristics, is likely to become the norm. As a result, considerable innovation is required to develop system support to meet the needs of future computing platforms.

The XPRESS project aims to develop and prototype a revolutionary software system for extreme-scale computing for both exascale and strong-scaled problems. The XPRESS collaborative research project will advance the state-of-the-art in high performance computing and enable exascale computing for current and future DOE mission-critical applications and supporting systems. The goals of the XPRESS research project are to: A. enable exascale performance capability for DOE applications, both current and future, B. develop and deliver a practical computing system software X-stack, OpenX, for future practical DOE exascale computing systems, and C. provide programming methods and environments for effective means of expressing application and system software for portable exascale system execution.

The role of the University of Houston in XPRESS has been to define a migration path for porting legacy MPI [1] and OpenMP [2] applications to an XPI-like programming interface. Legacy migration is critical in terms of supporting the current code base on the XPRESS software stack. XPRESS, as far as we know, is the only project in the XStack program that identifies this component as one of its main goals. Thus we believe the outcome of our work and the experiences we have gained will be very important to other projects, as well as to other efforts related to exascale system software. During this process, a number of applications have been implemented, as

well as translation layers introduced to support legacy applications, to drive system development and quantitative evaluation of the XPRESS system implementation details and operational efficiencies and scalabilities. Our results show that such a translation is feasible and that legacy applications may benefit from the OpenX software stack.

2 Project Milestones

The major project milestones for this 3 year project are as follows:

MIlestone 1: Implementation of data-driven computation model in OpenMP. (Year 1, Q2)

Milestone 2: Explored programming constructs available in OpenMP that could effectively target the HPX runtime, with an LU decomposition application. (Year 2, Q2)

Milestone 3: Running OpenMP applications on HPX with hpxMP. (Year 2, Q4)

Milestone 4: Extending OpenMP support to multiple compilers with OMPTX with the Intel OpenMP Runtime. (Year 3, Q2)

Milestone 5: Running MPI and HPX applications with HPX-RTE. (Year 3, Q4)

3 Work Completed

One of the major goals of the XPRESS project is to make it easier to leverage the proposed software stack within an exascale environment for existing, legacy applications. These applications, predominantly using MPI and OpenMP to express parallelism, must continue to be supported in such an environment. For the work completed within the XPRESS project, the team at the University of Houston made significant progress in enabling MPI and OpenMP codes to run over HPX.

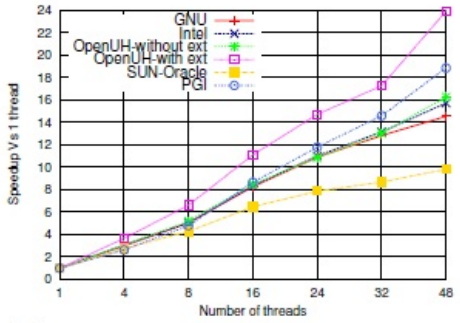
3.1 Data-driven Tasking Model in OpenMP

During the first year of the XPRESS project, we implemented a data-driven computation model in OpenMP that is similar to the HPX future feature to support data-flow computation and the task dependency feature incorporated into OpenMP 4.0. In Figure 1, we use an LU decomposition example to illustrate the potential performance benefit of data-driven concurrent tasks in OpenMP. Our task extensions do not require the static construction of a task graph, nor do they require the specification of task dependences via the variables that are read in from previous tasks (or passed to others), but rather permit the expression of dependences via the use of parametrized expressions that characterize each task.

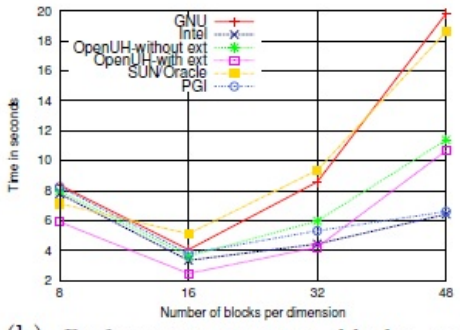
This feature was implemented in the OpenUH compiler. The "OpenUH with ext" entries in the Figure 1 graphs show the performance improvement that is achieved by this code version over the standard OpenMP task dependence construct. The figure shows that this approach to specifying dependences among tasks also considerably outperforms other well-known data-flow runtime systems such as QUARK [14] and OmpSs [15] by a large margin for LU and Smith-waterman benchmark examples. Such an approach could significantly enhance the ability of application codes (partially) based on OpenMP to exploit the OpenX software stack or similar execution environments.

```
#pragma omp parallel
{
 #pragma omp master
 {
   for ( i=0; i<matrix_size; i++ ) {
/* Processing Diagonal block */
   ProcessDiagonalBlock(.......);
   for (j=1;j<M;j++){
/* Processing block on column */
   #pragma omp task out(2*j)
     ProcessBlockOnColumn(........);
/* Processing block on row */
   #pragma omp task out(2*j+1)
     ProcessBlockOnRow
          (.....................);
     }
/* Processing remaining inner block */
   for (i=1;i<M;i++)
     for (j=1;j<M;j++){
   #pragma omp task in(2*i) in(2*j+1)
   ProcessInnerBlock(..............);
       }
   #pragma omp taskwait
     }
  }
}
```

Listing 1.3: LU decomposition Algorithm using OpenMP tasks with extensions implemented in OpenUH [15]



(a) Speedup for matrix size 4096 X 4096 with -O3 optimization



(b) Performance varying blocks per dimension-matrix size 4096 with -O3 optimization with 48 threads

Figure 1: Data-driven OpenMP task extension for LU decomposition

3.2 hpxMP

The main thrust of our effort was to directly translate the full feature set of OpenMP to the project's software environment so that existing application code could run unmodified (so far as possible) in OpenX. Our initial approach utilized our robust OpenUH compiler that is able to translate OpenMP code with Fortran, C and C++ code. Like other OpenMP compilers, OpenUH employs a custom OpenMP runtime library to ensure efficiency of the translation process.

We developed a strategy for transparently running code written using the full feature set of OpenMP on HPX. Compilers translate OpenMP code to object code that makes extensive use of the specific compiler's OpenMP runtime library routines to create and manage threads, to assign work to them and to perform synchronization operations. We chose to accomplish the translation by modifying the OpenUH runtime library, thereby avoiding any need to make changes to the application or the compilation process. Note that this approach thus requires absolutely no compiler or source code changes for legacy OpenMP applications, or even recompilation.

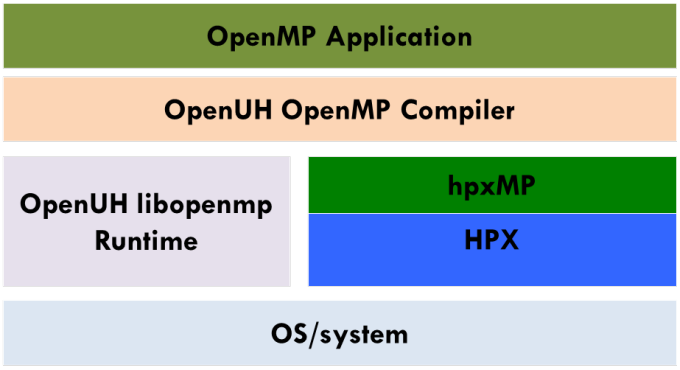


Figure 2: HPX integrated within OpenUH OpenMP runtime

During the second project year, we developed and evaluated a prototype of an alternative implementation of our OpenMP runtime library, named hpxMP, that is based on the HPX-3 runtime system from our partners at LSU. In this runtime, HPX threads are used to create the implicit tasks in the fork at the beginning of parallel regions, and the barrier *local control object* (LCO) provided by HPX is used to perform barrier synchronizations. OpenMP explicit tasks are implemented using HPX user-level threads, and a counter local to each task is used to synchronize between tasks.

OpenMP distinguishes between tied and untied tasks, where a tied task must run only on the thread it starts execution on and an untied task may freely migrate between threads at designated task-switching points. HPX does not provide any means to control on which thread (corresponding to an HPX worker) a task (or HPX user-level thread) executes. Consequently, all OpenMP explicit tasks in our present implementation have the semantics of an untied task. Implicit tasks are also implemented as HPX user-level threads.

In hpxMP, we ensure that each implicit task generated by a parallel region is assigned a unique thread number irrespective of the HPX worker (implemented as an OS thread) that it actually executes on. In effect, an OpenMP thread number corresponds not to the executing HPX worker, but rather the thread number of its generating task and, in turn, all ancestor tasks up to the starting implicit task. As a result, in most applications the tasks will all report that they are executing on thread 0, meaning that multiple simultaneously executing tasks have the same thread id. The code will not be able to correctly make use of thread local storage.

Our results showed very promising performance of the hpxMP runtime library when compared with a conventional OpenMP runtime system, using both microbenchmarks and applications provided by LBNL partners.

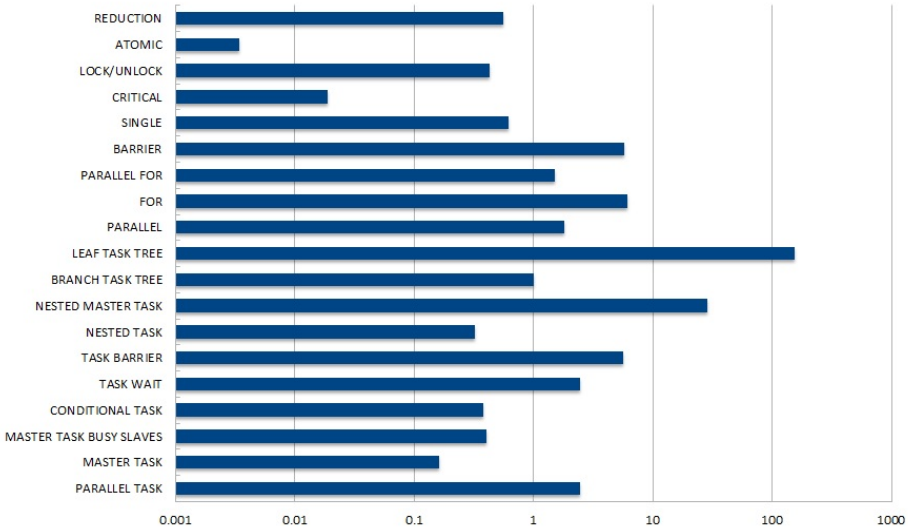


Figure 3: EPCC Microbenchmarks; <1 means improvement over original OpenMP runtime

Figures 2 and 3 shows the hpxMP architecture and the early results using the OpenUH compiler. Figure 2 shows the different compiler runtime elements: one is the compiler's original Pthreads-based runtime (libopenmp), and the other is the hpxMP runtime implemented within this project using HPX. Figure 3 shows initial results comparing the overheads measured for a variety of OpenMP constructs. This evaluation made use of the EPCC syncbench [12] and taskbench [13] microbenchmarks. As can be seen the results were mixed, where hpxMP yielded improvements in 9 tests, resulted in performance degradation in 9 tests, and had virtually the same performance in 1 test.

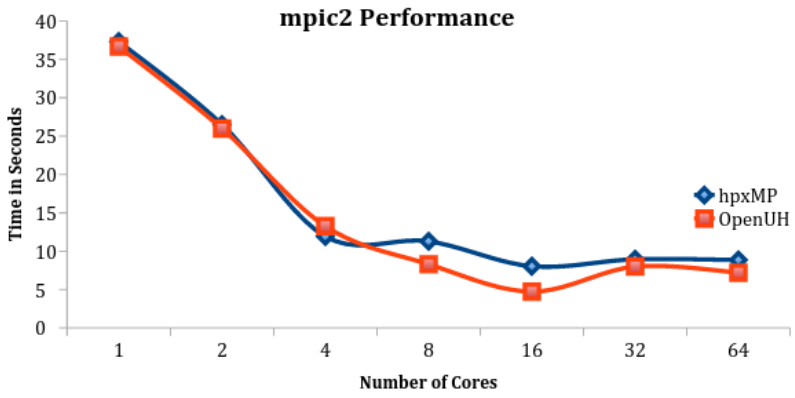


Figure 4: Preliminary performance results for Paricle-In-Cell (PIC) application from LBNL

3.3 OMPTX

Our initial approach to supporting OpenMP codes on exascale platforms via HPX aimed to maximize transparency for the application developer by avoiding any change to the source code or to the compiler. However, the hpxMP runtime, as described above, is custom software that can only be used in conjunction with the OpenUH OpenMP compiler. This reduces the opportunities for experimentation with OpenMP over HPX on platforms where OpenUH is not installed. Moreover, OpenUH does not support certain recent C++ and Fortran language features. In the meantime, Intel has open-sourced their OpenMP runtime, which was originally derived from the KAI runtime and is of high quality. This runtime library has since been adopted in a number of existing compilers, including compilers from GNU, PathScale, and Clang, and support within OpenUH is underway.

To overcome this, and to enable a broad array of opportunities for evaluation, in year 3 of the project we developed OMPTX, an OpenMP runtime based on HPX which shares the same ABI as the newly open-sourced Intel OpenMP runtime library. The ability to use OMPTX as a drop-in replacement for the Intel OpenMP runtime significantly enhances the opportunities for experimentation using not only benchmarks, but full applications.

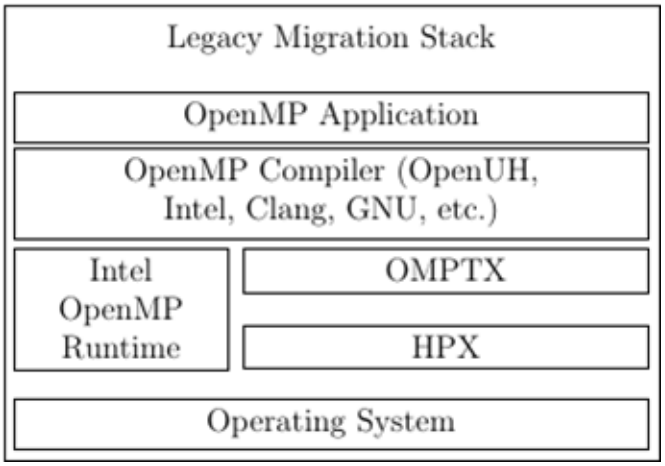


Figure 5: HPX-integrated OpenMP software stack for compilers compatible with Intel OpenMP runtime

For OMPTX we have developed an alternate implementation strategy focused on achieving full conformance. The HPX thread API is used to place the implicit tasks on each thread, and a static scheduler is used to keep those tasks in place. Through the use of the *executors* [11] feature available in HPX-3, tasks are executed using a work stealing scheduler, providing a view of thread local storage that is consistent with the HPX thread worker ID used in the parallel region tasks. The key difference is that tasks would behave as the programmer expects, with no more than one task executing on a thread simultaneously, and providing a view of thread local storage from tasks that is consistent with the view from its executing thread.

We have also explored how features that were introduced to OpenMP during the timeframe of this project, in version 4.0, including task dependencies, support for offloading work and data to accelerators, and thread affinity, may be supported using the HPX runtime. We have utilized the data-flow facilities provided by HPX to implement the new depends clause for specifying dependencies between OpenMP tasks in OMPTX. This feature extends the ability of OpenMP to express task-based computations and its provision in the XPRESS framework will enable early experimentation with OpenMP support for this programming style. It should also result in better performance, as it enables OpenMP codes to directly exploit more of the functionality provided by HPX.

The constructs in OpenMP map to HPX with varying degrees of difficulty. When forking threads for a parallel region, pinning each implicit task to an OS thread becomes difficult, and requires the use of the HPX threading API, a static scheduler, and the use of executors with workstealing schedulers for explicit tasks. As mentioned in Section 3.2, while untied tasks map to HPX threads directly, some of the scheduling restrictions imposed on tied tasks within the OpenMP specification require modification to the internals of the HPX runtime. OpenMP (as of version 4.0) uses the names of variables to specify data dependencies between tasks, which requires that the runtime store a mapping of these variables to tasks. However, HPX uses a future-based mechanism to specify dependencies between task (or HPX user-level threads). Hence, in OMPTX we maintain a mapping of variables to futures for handling task dependencies. The `taskgroup` construct maps closely with the executors feature provided in HPX-3, making their integration within the executor-based framework used for implementing tasks seamless. Finally, the sparse documentation for certain constructs in the Intel runtime make it difficult to manage subtle differences in behavior between the Pthreads-based OpenMP runtime and the OMPTX runtime. Specifically, this pertains to the correct and consistent management of threadprivate data, as well as variables that are private or firstprivate in parallel regions.

3.4 MPI and HPX

As with OpenMP, our approach to enabling MPI applications to run in an HPX environment also aimed to provide a translation that is transparent for the application developer. We have also made significant progress in our support for MPI legacy codes, by utilizing HPX as the runtime library for Open MPI [5], one of the major and widely deployed implementations of MPI. At this point, Open MPI is able to use the HPX runtime environment for startup and communication through the TCP/IP transport module, although some details with respect to the integration of memory allocators and the internal utilization of threads in Open MPI remain to be addressed as future work.

As part of the research performed in the project, a runtime component of Open MPI has been successfully developed and deployed that is based on HPX-3, which

removes the necessity for using the current runtime component (orte) of Open MPI. The key technical contribution is a database (db) component based on HPX, which allows the fetching of key-value pairs from remote memory locations by each individual process. An application using the HPX-RTE component represents simultaneously an MPI as well as an HPX application, allowing for a gradual transition from MPI constructs to ParalleX [7].

The HPX-RTE component has been successfully deployed on InfiniBand and TCP/Ethernet clusters in smaller configurations (up to 16 processes) for various benchmarks, including a CFD kernel and an image processing kernel. The HPX-RTE runtime shows some performance improvements compared to the original ORTE component, although a comparison to the direct-launch mechanism of Open MPI (which has lower startup costs) is still to be done.

3.5 Benchmarks and Evaluation

We have installed OMPTX on NERSC's Edison and LSU's Hermione cluster for the purpose of evaluating its performance. In Figure 6, we show the potential performance gains when using task dependencies. We used the EPCC OpenMP microbenchmarks to measure the overhead of individual OpenMP constructs implemented via a translation to the OMPTX runtime system, as shown in Figure 7. We experimented with OpenMP versions of both of the mini-apps that have been ported to HPX, miniGhost [8] and HPCG [9]. Of particular interest was the performance of OpenMP's tasking constructs when executing over HPX, and we evaluated this using the Barcelona OpenMP Tasks Suite (BOTS) [10].

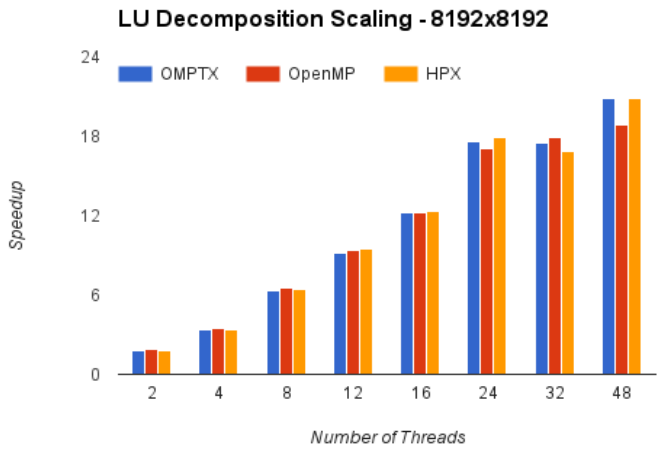


Figure 6: Performance for LU decomposition benchmark

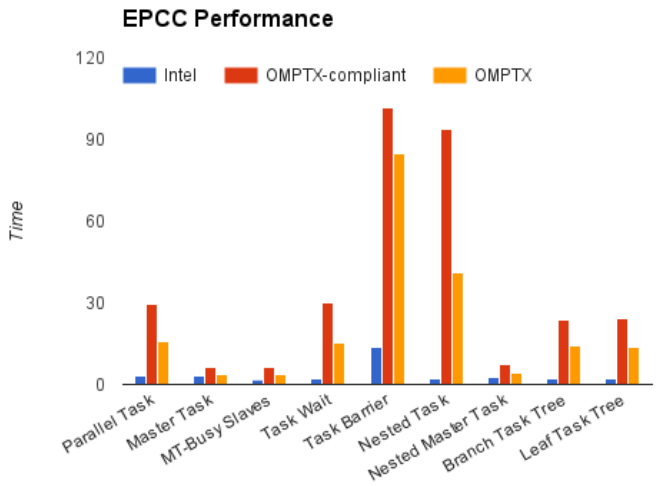


Figure 7: Performance for EPCC microbenchmarks

3.6 Other Activities

UH has been actively involved in the OpenMP and OpenACC [3] standardization organizations. PI Chapman and her team have participated in several OpenMP language subcommittees and also took part in the design, implementation and evaluation of the OpenACC programming interface. Dr. Yonghong Yan, a Research Assistant Professor supported by this project, has extensively engaged in the work of the OpenMP Architecture Review Board (ARB) and chairs its Interoperability Language Subcommittee. Co-PI Edgar Gabriel is the lead of the UH OpenMPI team; he has been working extensively with other OpenMPI members to support the standardization of MPI features and their implementation in Open MPI. Via these interactions, the results of this project have been available to a broader community of implementers of these de facto standards. The continued engagement with the standards communities is helping us to disseminate information and gather feedback on the XPRESS software stack and ParalleX execution model. Moreover, potential enhancements to OpenMP that would enable OpenMP programs to more fully benefit from HPX are the basis of a new proposal to the OpenMP language committee.

4 Software Products

4.1 HPX-RTE

Gabriel's team is a long-time contributor to the Open MPI software project. For XPRESS, they have added support for HPX-RTE, an HPX-based runtime layer for Open MPI. It uses the HPX runtime environment for startup and communication through either the TCP/IP or InfiniBand network transport module. Open MPI is also able to use the native SLURM startup through HPX.

4.2 OMPTX

Within this project, Chapman's research team has developed OMPTX, an OpenMP runtime based on HPX which shares the same ABI as the newly open-sourced Intel OpenMP runtime. This runtime has been adopted in existing compilers from Intel, GNU, PathScale, and Clang [5], and support within OpenUH [4] is underway. The ability to use OMPTX as a drop-in replacement for the Intel OpenMP runtime facilitates experimentation in running OpenMP programs with HPX. The Intel OpenMP Runtime interface and a number of the compilers which use it support OpenMP 4.0, allowing us to experiment with implementation of recent OpenMP features within the proposed software infrastructure.

4.3 Technologies or Techniques

4.3.1 OpenACC

The University of Houston team has participated in the on-going specification of directive features for high-level, portable development of accelerator code and has developed an OpenACC compiler implementation within OpenUH as well as a test suite. The former was initially designed to target Nvidia GPGPU platforms that are the most-configured accelerator in current large-scale platforms. Recent work has extended the implementation to also target AMD's APU devices. The implementation enables experimentation with the translation of additional directive features. This work is moreover in the early stages of being adapted to the syntax adopted by OpenMP for accelerators in order to enable implementation of the features contained in OpenMP version 4.5. Our implementation supports most of the features described in OpenACC 2.0 for C/C++ and Fortran. We evaluated various strategies for mapping parallelizable loops to gang-, worker-, and vector-level parallelism, for implementing reductions, and for reducing synchronization costs in the GPU through redundant execution. We have also developed several OpenACC benchmarks to evaluate our implementation as well as other OpenACC implementations.

5 Conclusion

In this report we have described our approach and implementation of legacy applications on HPX. Support for MPI has been provided through HPX-RTE enabling side by side HPX and MPI communication. OpenMP support has been provided through the OMPTX implementation of the Intel OpenMP runtime, effectively mapping OpenMP tasks to HPX threads .

The mapping of fork-join and work sharing clauses is straightforward, and in a runtime approach, there is little to be done with respect to optimization. However,

the OpenMP and HPX task APIs are subtly different, and these differences have a strong influence on certain aspects of the runtime implementation.

OpenMP constructs are inherently bound to the nested scopes that OpenMP supports and uses. In the implementation, it fits very closely with calling the task functions on the same stack frame for each thread. Tied tasks go even further to make task execution more closely resemble the functions called in a serial application. Similarly, the relationship between OpenMP task dependencies, and the tasks they constrain, maps directly to a hash table.

HPX does not align to the same shared stack frame usage as OpenMP and requires dynamic stack frames due to the of the flexibility of the dependencies that can be specified using futures. This dynamic allocation of stack frames has an overhead when creating HPX threads that is unavoidable in the implementation. The application needs to avoid the overhead by simply not suspending large numbers of tasks using synchronization like locks and waits, which is very common in OpenMP.

In order to realize the performance potential of the HPX runtime, applications need to be written to exploit task dependencies. The algorithmic shift to task dependencies is by far the largest hurdle with the largest benefit, given the scope of shared memory applications. The performance benefits of HPX over OpenMP task dependencies was not observed for small problems; larger applications that can take advantage of the more robust interface that HPX provides are needed to demonstrate the potential performance benefits. The recent availability of HPX-RTE facilitates running large legacy applications on HPX through our support layers.

6 On-going and Future Work

Future work that builds on top of these efforts will focus on identifying extensions for MPI and OpenMP based on the experiences gained during our porting activities. We will propose extensions to MPI in order to enable a more asynchronous, task-driven execution model which can benefit directly from the ParalleX execution model provided by HPX. For OpenMP, we will explore new means to enhance the expression of computations for data-flow execution that go beyond the additions in OpenMP 4.0 and 4.5. For example, our experiences with HPX-3 show how parallelization of some recursive algorithms can benefit from tasks returning futures and from operations for waiting on the completion of specified futures. Providing a means to express this in OpenMP might ease the burden of transitioning large legacy applications written using OpenMP to a programming interface such as XPI.

We are studying the interactions between legacy applications and the proposed software stack and how to facilitate the migration of codes using existing programming models to an XPI-like programming interface. This entails: exploring how legacy codes can benefit from the introspection capabilities provided within the project's software stack, developing translation strategies for recently

introduced and anticipated OpenMP constructs, and dealing with interoperability challenges that arise when application codes are developed that use a mixture of old and new programming interfaces. A key output of this investigation would be proposals for extensions to legacy programming interfaces which would allow them to more fully leverage the capabilities of future exascale system software. Additionally, we are working on identifying additional features that would be needed in HPX to support the requirements of existing, legacy programming interfaces.

We will evaluate the efficacy of such approaches compared to our current strategy of retaining the same compiled code but replacing a standard OpenMP runtime by OMPTX. We will also formulate a porting strategy for incrementally migrating legacy applications to a restructured code base using newer interfaces (e.g. XPI or HPX-3) which can directly map to the ParalleX execution model. Future work will require a collaboration with application partners in the XPRESS project to select applications to demonstrate this approach. We intend to document our porting efforts for a range of existing legacy codes, to serve as examples for how codes may be transitioned to the new software stack. During the course of this work, we will also identify potential interoperability issues, such as resource contention among different runtime components, that may arise when mixing MPI and OpenMP codes with newer interfaces.

7 Publications

1. Priyanka Ghosh, Yonghong Yan, and Barbara Chapman. Support For Dependency Driven Executions Among OpenMP Tasks. In 2nd Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM 2012) in conjunction with PACT 2012.
2. Priyanka Ghosh, Yonghong Yan, Deepak Eachempati, and Barbara Chapman. "A Prototype Implementation of OpenMP Task Dependency Support." In OpenMP in the Era of Low Power Devices and Accelerators, pp. 128-140. Springer Berlin Heidelberg, 2013.
3. Liao, Chunhua, Yonghong Yan, Bronis R. de Supinski, Daniel J. Quinlan, and Barbara Chapman. "Early experiences with the OpenMP accelerator model." In OpenMP in the Era of Low Power Devices and Accelerators, pp. 84-98. Springer Berlin Heidelberg, 2013.
4. Xiaonan Tian, Rengan Xu, Yonghong Yan, Zhifeng Yun, Sunita Chandrasekaran, and Barbara Chapman, Compiling a High-level Directive-Based Programming Model for GPGPUs 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC2013), San September 2013.
5. Munara Tolubaeva, Yonghong Yan, and Barbara Chapman, Compile Time Modeling of Off-Chip Memory Bandwidth for Parallel Loops 26th International Workshop on Languages and Compilers for Parallel Computing (LCPC2013), September 2013.

6. Qawasmeh, Ahmad, Abid M. Malik, and Barbara M. Chapman. "OpenMP Task Scheduling Analysis via OpenMP Runtime API and Tool Visualization." In Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 1049-1058. IEEE, 2014.
7. Rengan Xu, Maxime Hugues, Henri Calandra, Sunita Chandrasekaran, and Barbara Chapman. "Accelerating Kirchhoff migration on GPU using directives." In Proceedings of the First Workshop on Accelerator Programming using Directives, pp. 37-46. IEEE Press, 2014.
8. Anilkumar Nandamuri, Abid M. Malik, Ahmad Qawasmeh, and Barbara M. Chapman. 2014. Power and energy footprint of openMP programs using OpenMP runtime API. In Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing (E2SC '14). IEEE Press, Piscataway, NJ, USA, 79-88.
9. Rengan Xu, Xiaonan Tian, Yonghong Yan, Sunita Chandrasekaran, and Barbara Chapman, Reduction Operations in Parallel Loops for GPGPUs, 2014 International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM 2014), February, 2014, Orlando, Florida, USA.
10. Munara Tolubaeva, Yonghong Yan and Barbara Chapman. Predicting Cache Contention for Multithread Applications at Compile Time, 16th Workshop on Advances in Parallel and Distributed Computational Models, in conjunction with IPDPS 2014, May 2014.

8 References

[1] Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.

[2] Chapman, Barbara, Gabriele Jost, and Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. Vol. 10. MIT press, 2008.

[3] Hart, Alistair. "The OpenACC programming model." *Cray Exascale Research Initiative Europe, Tech. Rep* (2012).

[4] Chapman, Barbara, Deepak Eachempati, and Oscar Hernandez. "Experiences developing the openuh compiler and runtime infrastructure."*International Journal of Parallel Programming* 41.6 (2013): 825-854.

[5] Lattner, Chris. "LLVM and Clang: Next generation compiler technology." *The BSD Conference*. 2008.

[6] Gabriel, Edgar, et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation." Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg, 2004. 97-104.

[7] Kaiser, Hartmut, Maciej Brodowicz, and Thomas Sterling. "Parallex an advanced parallel execution model for scaling-impaired applications."

Parallel Processing Workshops, 2009. ICPPW'09. International Conference on. IEEE, 2009.

[8] Barrett, Richard F., Courtenay T. Vaughan, and Michael A. Heroux. "MiniGhost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing." *Sandia National Laboratories, Tech. Rep. SAND* 5294832 (2011).

[9] Dongarra, Jack, and Piotr Luszczek. "HPCG technical specification." *Sandia National Laboratories, Sandia Report SAND2013-8752* (2013).

[10] Duran, Alejandro, Xavier Teruel, Roger Ferrer, Xavier Martorell, and Eduard Ayguade. "Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp." In *Parallel Processing, 2009. ICPP'09. International Conference on*, pp. 124-131. IEEE, 2009.

[11] Hoberock, Jared, Michael Garland, and Olivier Girioux. "Parallel Algorithms Need Executor | N4406." JTC1/SC22/WG21 - The C Standards Committee - ISOCPP. April 10, 2015. Accessed November 25, 2015. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4406.pdf.

[12] Bull, J. Mark, and Darragh O'Neill. "A microbenchmark suite for OpenMP 2.0." *ACM SIGARCH Computer Architecture News* 29, no. 5 (2001): 41-48.

[13] Bull, J. Mark, Fiona Reid, and Nicola McDonnell. "A microbenchmark suite for OpenMP tasks." In *OpenMP in a Heterogeneous World*, pp. 271-274. Springer Berlin Heidelberg, 2012.

[14] Yarkhan, Asim, Jakub Kurzak, and Jack Dongarra. "QUARK Users' Guide."*Electrical Engineering and Computer Science, Innovative Computing Laboratory, University of Tennessee* (2011).

[15] Duran, Alejandro, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. "Ompss: a proposal for programming heterogeneous multi-core architectures." *Parallel Processing Letters* 21, no. 02 (2011): 173-193.