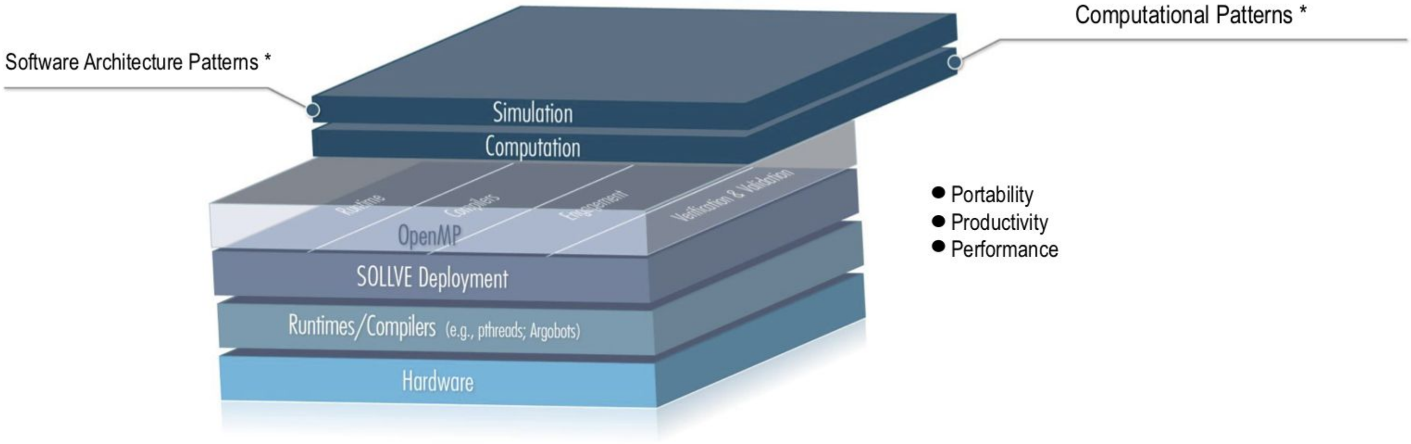
Low-overhead Loop Scheduling to Improve Performance of Scientific Applications

Vivek Kale

November 20, 2019, 2:15 PM
SC ‘19 OpenMP Booth Talk
Denver, Colorado

SOLLVE Software Stack



* Parallel Patterns by Tim Matteson

Our strategies are in runtime system and compiler in SOLLVE slab

Motivating Example Code Structure

Outer Iteration

Computation region

```
#include <mpi.h>
int main(int argc, char** argv)
{
  MPI_Init(argc, argv);
  // input
  while (global_err < thresh)
  {
   MPI_Isend()/MPI_Irecv()/MPI_Waitall();

   for (i = 0; i < n; i++)
       doCompute(n);

   MPI_Collective_Op(&global_err);
   timestep++;
  }
  // output, viz
  MPI_Finalize();
}
```

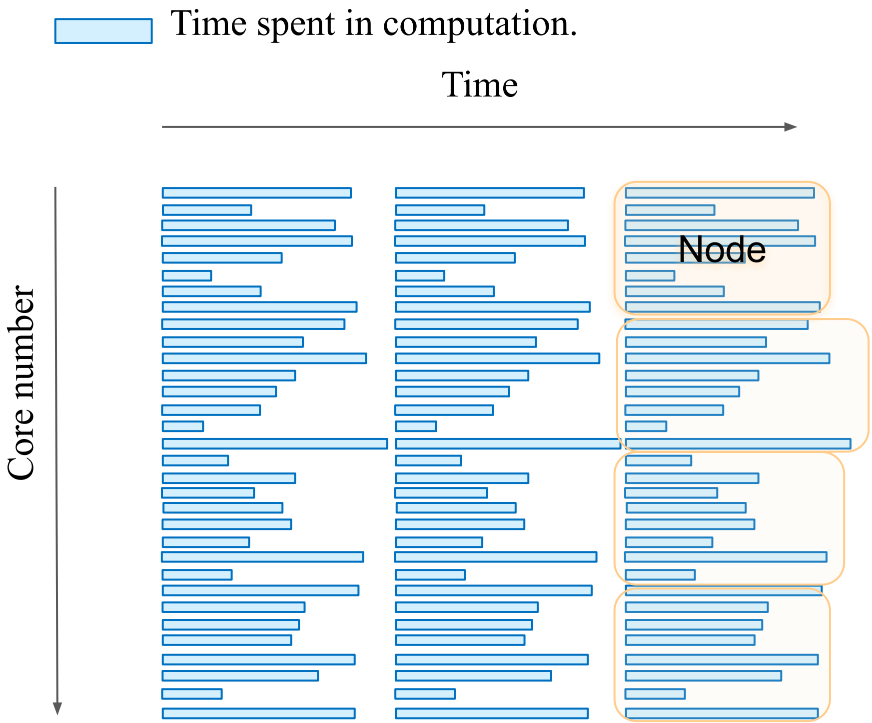
A Loosely Synch MPI Comm.

A motif[1], e.g., n-body, Stencil, Dense Matrix Factorization, Sparse Matrix-Vector Multiplication

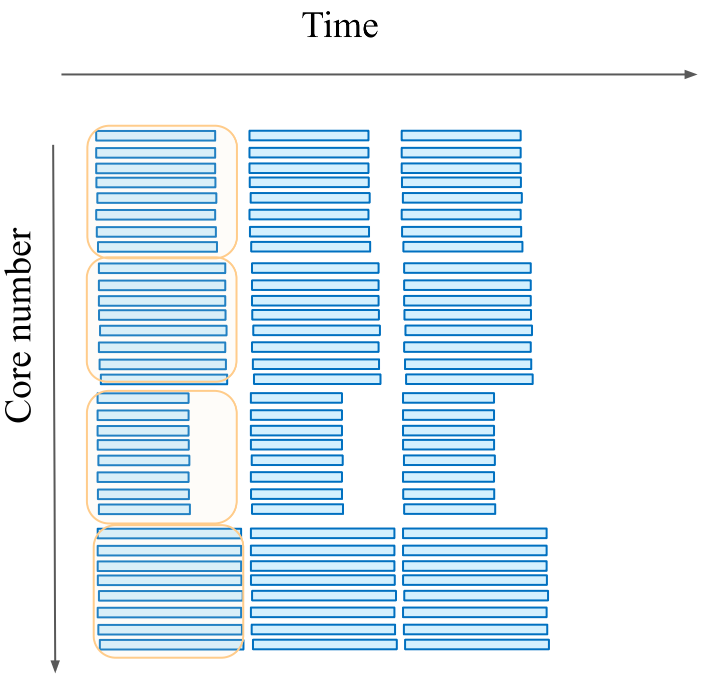
A Bulk Synch MPI Coll. Comm.

1. https://patterns.eecs.berkeley.edu/

Within-node Persistent Load Imbalance



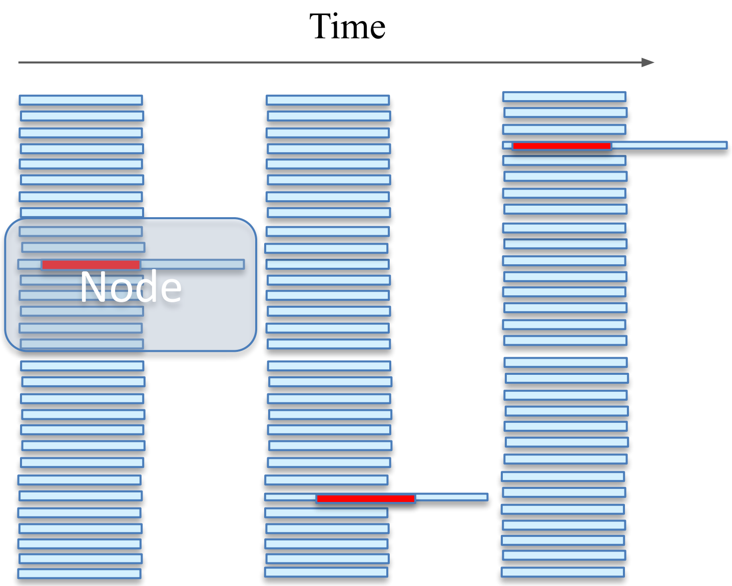
Application-created imbalances.



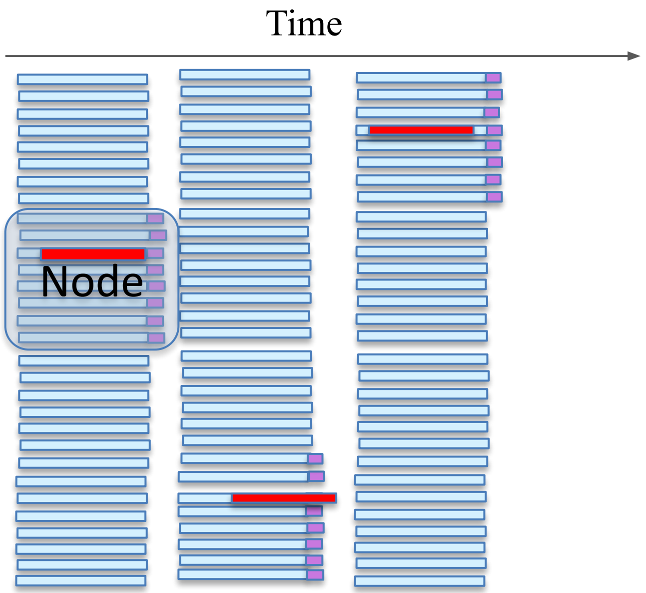
These can be (mostly) handled if we had a perfect within-node load redistribution.

→ Within-node persistent imbalance causes significant slowdown in a multi-node run.

Transient Load Imbalance and its Potential Mitigation



Noise delays every
iteration on some node



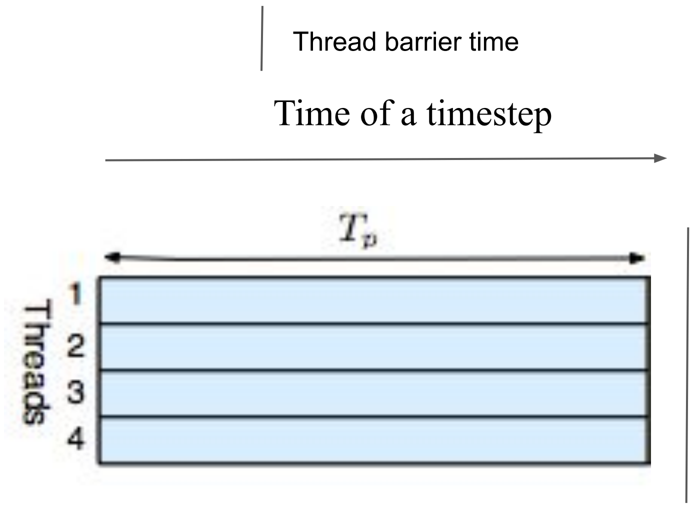
Here also: performance improves
if we can perfectly redistribute
work within each node

Noise amplification:
[PetriniCaseSC03] , [hoeflerNoiseSC10]

How to Do Near-perfect Work Redistribution Within Node?

```
#include <mpi.h>
#include <omp.h>

int main(int argc, char** argv)
{
  while (timestep <1000 ) {
#pragma omp parallel for schedule(static)
     for(i=0; i<n; i++)
      loop_body(i);
      MPI_Op();
      timestep++;
     }
}
```



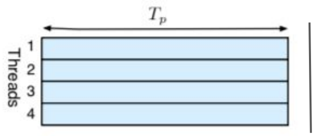
- Focus on the OpenMP computation region in an MPI+OpenMP program

→ Let's use OpenMP's dynamic loop schedule provided.

Hybrid Static/Dynamic Scheduling

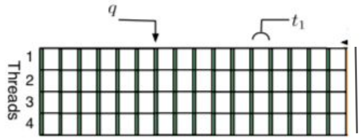
Statically Scheduled Work | Dynamically Scheduled Work | Dequeue Overhead | Thread barrier

```
#pragma omp parallel for schedule(static)
     for(int i=0; i<n; i++)
          loop_body(i);
```



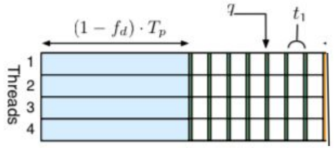
Susceptible to imbalance.

```
#pragma omp parallel for schedule(static)
     for(int i=0; i<n; i++)
          loop_body(i);
```



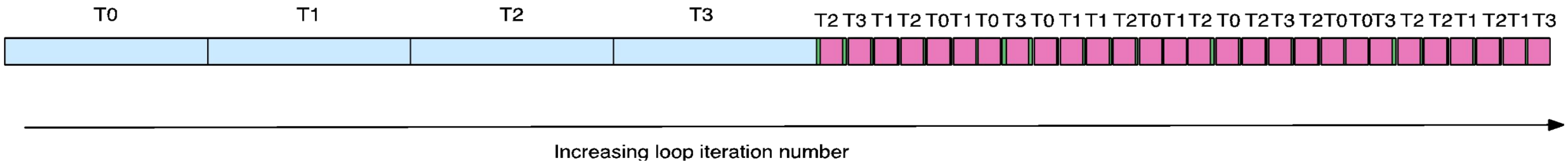
Scheduler overhead stretches time.

```
#pragma omp parallel for nowait
     for(int i=0; i<n; i++)
          loop_body(i);
#pragma omp parallel for schedule(dynamic)
     for(int i=0; i<n; i++)
          loop_body(i);
```

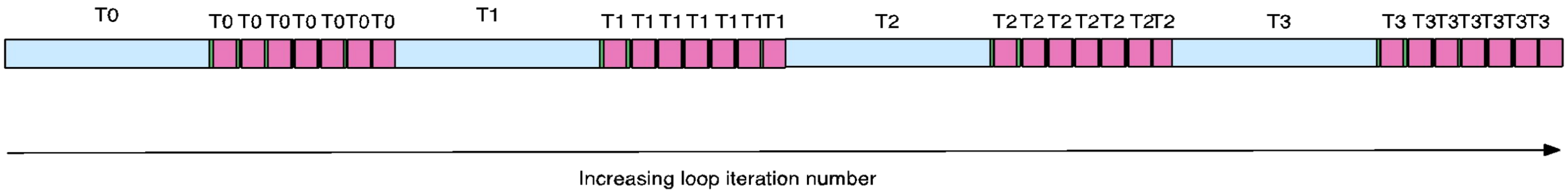


Can reduce imbalance and sched ovhd. simultaneously.

Staggered Static/Dynamic Loop Scheduling

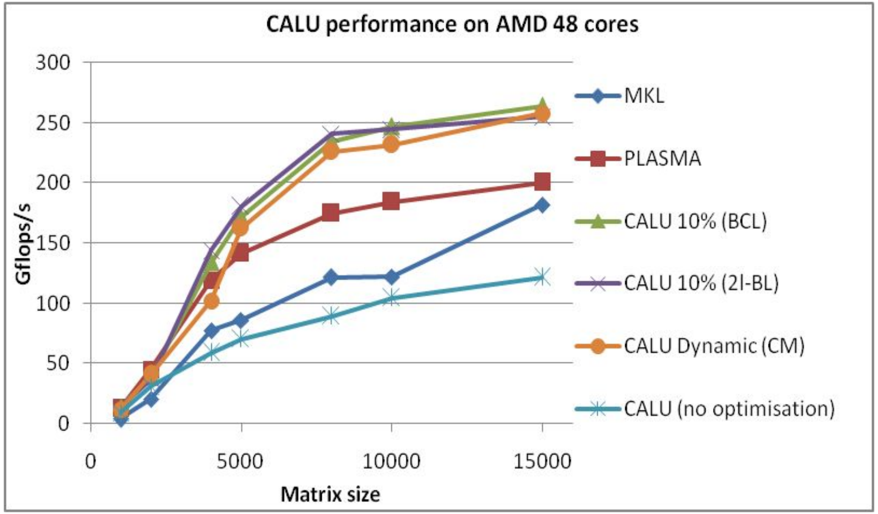


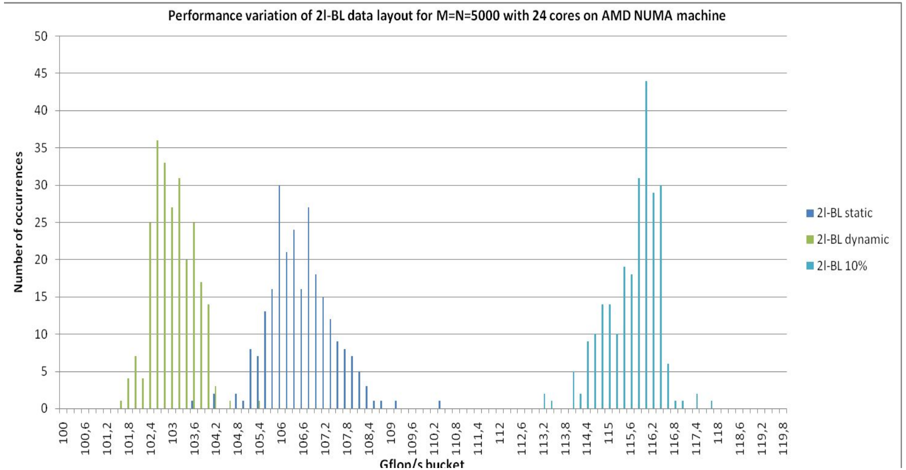
- Look at loop iteration space. Have spatial locality in static.
- Problem with spatial locality → don’t take advantage of prefetching engine for dynamic chunks.



- Each thread finishes its static chunk.
- Then does dynamic chunks marked for it if available.
- Only if not, looks for other dynamic chunks from other threads to steal.

Communication-Avoiding Dense Matrix Factorization



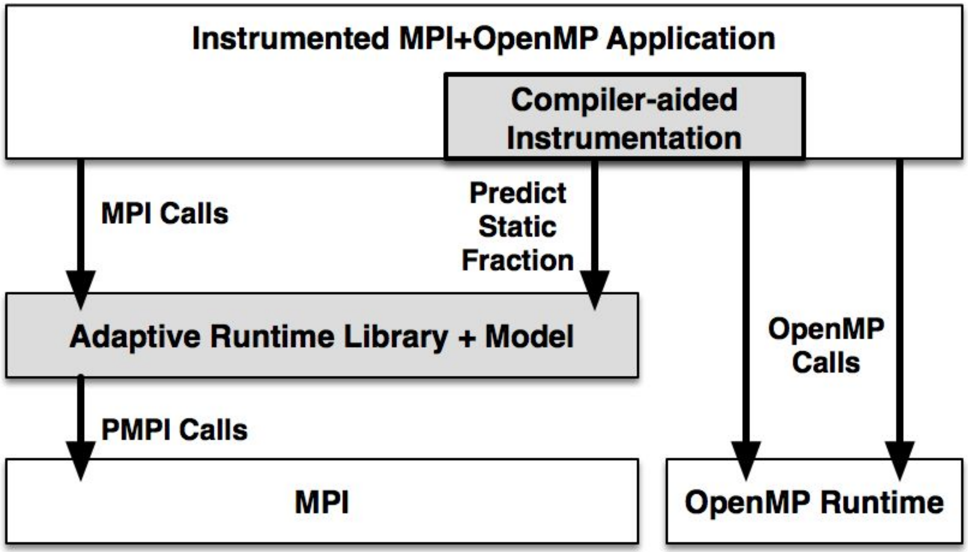


CALU static/dynamic does better than Utenn's PLASMA and Intel's MKL

Less performance variation for static/dynamic → better scaling projection

[SHI12] ***Hybrid Static/dynamic Scheduling for Already Optimized Dense Matrix Factorization.***

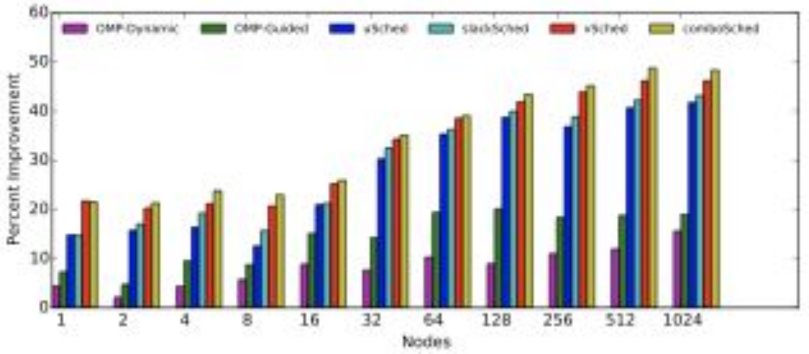
Software Architecture for Low-overhead Loop Scheduling



Software developed to easily create and experiment with new loop scheduling schemes

[VTI15] Vivek Kale. ***Low-overhead Scheduling for Improving Performance of Scientific Applications.*** Thesis. 2015. University of Illinois at Urbana-Champaign.

Results for Application Codes



Application	Rebound
ΔLOC per region	+10
Pct. LOC changed	12.31%

1. uSched better than OpenMP guided
2. Optimizations always improve over uSched.
3. Combining different techniques seems to add on benefits, i.e., they don't cancel benefit out.
4. Small code change, e.g., 41,421 loc to 41,982 loc (5.2%) for Rebound.

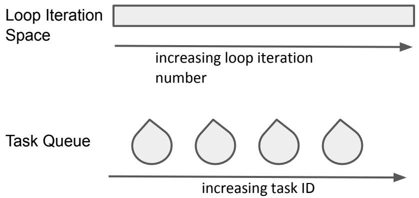
Figure: Performance improvement of MPI+OpenMP Rebound App. using lightweight loop scheduling on LLNL's cab.

Loop Shift Proposal

- Add **loopshift** directive
- Must be nested within a work-sharing directive and parallel region
- Allow to map iterator of some inner loop of the work-sharing loop with some arithmetic expression
- Can use pre-defined variables such as thread identifier (tid) and number of threads (numthreads)

```
#pragma omp parallel for
for (i = lbi; i < ubi; i++)
{
  int j;
  pragma omp loopshift(j = (i + tid) % numthreads)
  for (j = lbj; j < ubj; j++)
  {
    /* do work */
  }
}
```

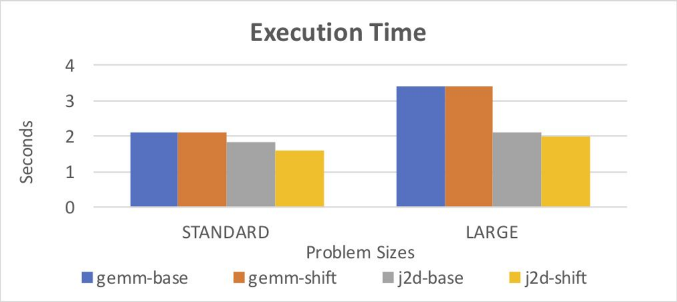
Listing 1: OpenMP LoopShift Directive

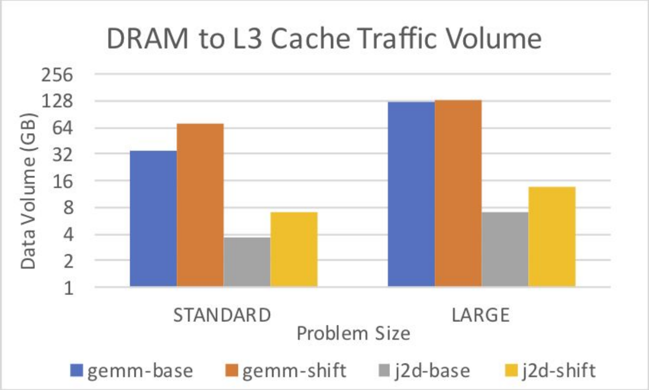


Data Locality in Tasking in OpenMP

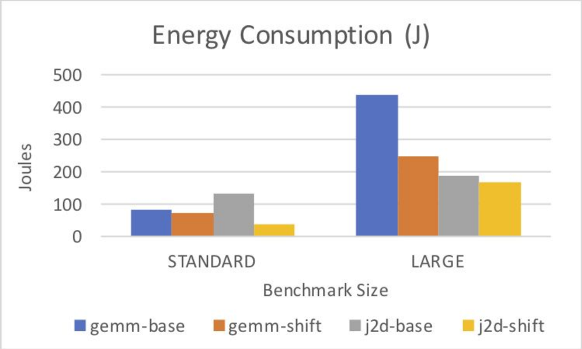
Enhancing Support in OpenMP to Improve Data Locality in Application Programs Using Task Scheduling

Martin Kong, Vivek Kale





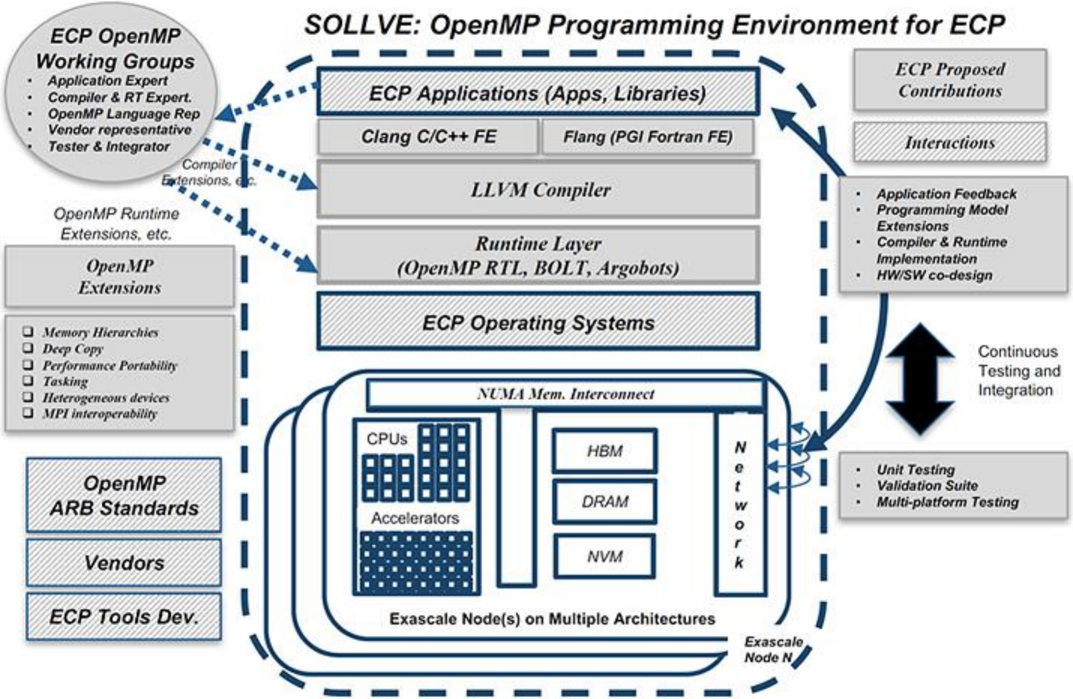
Lower execution time due to lower L3 due to lower DRAM cache traffic volume.



Lower energy consumption due to less data movement.

Using ECP's SOLLVE for your Applications

- SOLLVE is a project to develop OpenMP for exascale
- Can link it to your app through following http://github.com/SOLLVE/sollve
- I'm working on making it available on Spack.



Proposal for User-defined Schedules in OpenMP

Example: glimpse of how a User-defined Schedule (UDS) might look like

```
typedef struct {...} schedule_data;
void myDynsstart(...) {}
void myDynsnext(...) {}
void myDynsfini(...) {}
#pragma omp declare schedule(myDyn) start(myDynsstart) next(myDynsnext) fini(myDynsfini)
void example() {
    static schedule_data sd;
    int chunkSize = 4;
    #pragma omp parallel for schedule(myDyn, chunkSize:&sd)
    for(int i = 0; i < n; i++)
        c[i] = a[i]*b[i];

}
```

UDS identifier: `myDyn`; UDS parameters: `chunkSize:&sd`

- The directive `declare schedule` connects a schedule with a set of functions to initialize the schedule and hand out the next chunk of iterations.
- The syntax of the clause `schedule` is extended to also accept an identifier denoting the UDS.
- Instead of calling into the RTL for loop scheduling, the compiler will invoke the functions of the UDS.
- Visibility and namespaces of these identifiers will be borrowed from User-Defined Reductions in OpenMP 5.0.

Lightweight Loop Scheduling in RAJA: lws-RAJA

Code through hand transformation or maybe ROSE/Orio/LLVM.

```
#include "vSched.h"
#define FORALL_BEGIN(strat, s,e, start, end, tid, numThds )
loop_start_ ## strat
(s,e ,&start, &end, tid, numThds); do {
#define FORALL_END(strat, start, end, tid) } while(
loop_next_ ## strat (&start, &end, tid));
void* dotProdFunc(void* arg)
{
  int startInd = (probSize*threadNum)/numThreads; int endInd
=  (probSize*(threadNum+1))/numThreads;
  while(iter < numIters) {
   mySum = 0.0; //reset sum to zero at the beginning of the
product if(threadNum == 0) sum = 0.0;
   if(threadNum == 0) setCDY(static_fraction , constraint.
hunk_size);

#pragma omp parallel
FORALL_BEGIN(statdynstaggered , 0, probSize , startInd,endInd
,threadNum, numThreads)
for (i = startInd ; i < endInd; i++) mySum += a[i]*b[i]
 FORALL_END(statdynstaggered , startInd , endInd,threadNum)
pthread_mutex_lock(&myLock);
  sum += mySum;
  pthread_mutex_unlock(&myLock);
  pthread_barrier_wait(&myBarrier);
  if(threadNum == 0) iter++;
  pthread_barrier_wait(&myBarrier); } // end timestep loop
}
```

MPI+OpenMP code explicitly using lightweight scheduling.

RAJA User Code

```
RAJA::ReduceSum<RAJA::seq_reduce, double> seqdot(0.0);
RAJA::forall<RAJA::omp_lws>(RAJA::RangeSegment(0, N), [=]
(int i) {

seqdot += a[i] * b[i]; });

dot = seqdot.get();
std::cout << "\t (a, b) = " << dot << std::endl;
```

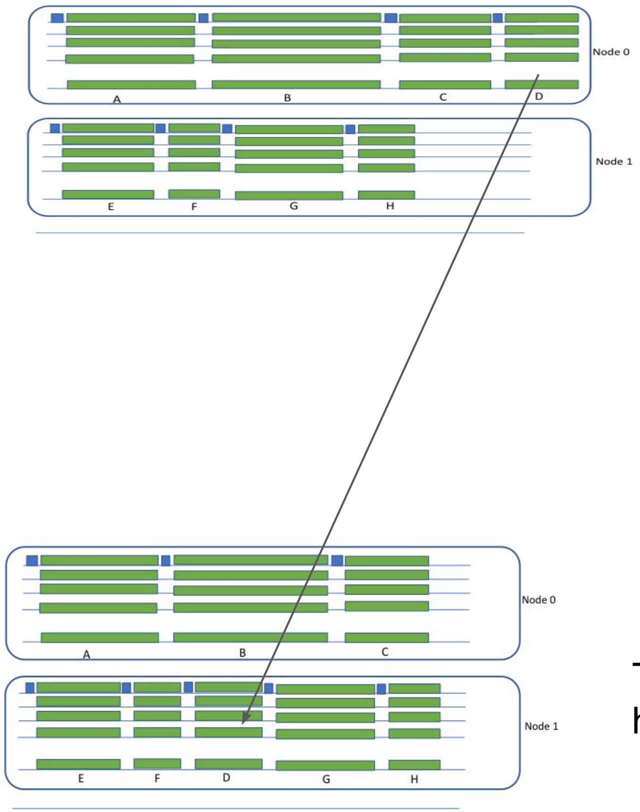
RAJA library implementation with policy `omp_lws`

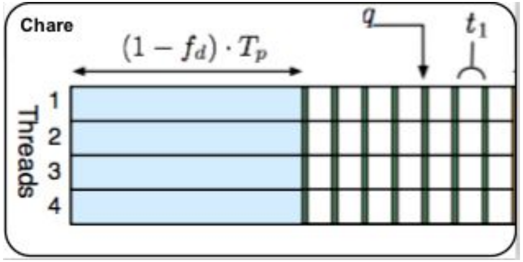
```
#include "vSched.h"
#define FORALL_BEGIN(strat, s,e, start, &end, tid, numThds) do {
#define FORALL_END(strat, start,tid)); start, end, tid, numThds ) loop_start_
## strat (s,e ,&end, tid) } while( loop_next_ ## strat (&start, &end)
template <typename Iterable , typename Func >
RAJA_INLINE void forall_impl(const omp_lws<&, Iterable&& iter, Func&&
loop_body) {
RAJA_EXTRACT_BED_IT(iter);
int startInd , endInd;
int threadNum = omp_get_thread_num();
int numThreads = omp_get_num_threads();
FORALL_BEGIN(statdynstaggered , 0, distance_it , startInd , endInd , threadNum
, numThreads) for (decltype(distance_it) i = startInd; i < endInd; ++i) {
loop_body(begin_it[i]); }
FORALL_END(statdynstaggered , startInd , endInd , threadNum)
}
```

→ Significantly reduces lines of code for application programmer to use strategy: **<u>easy-to-use</u> strategies.**

→ Improves ***<u>portability</u> of loop scheduling strategies***.

Load Balancing + Loop Scheduling Technique



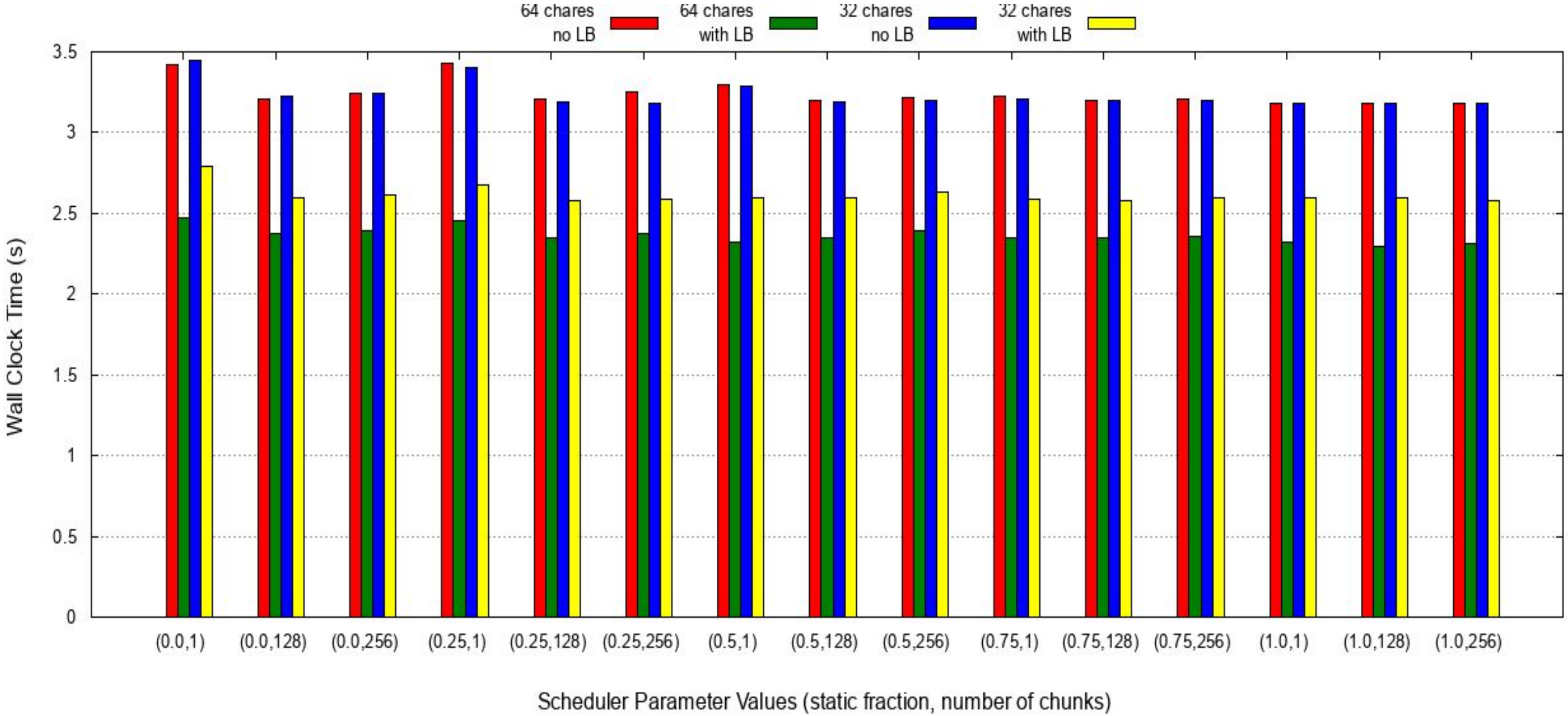


Key Idea:

1. Modify across-node load balancing in Charm++ to assign load to one PE in each node.
2. Use my loop scheduling strategies in CkLoop to optimize within node performance.

This scheduler recently merged into Charm++; Development history here: https://bitbucket.org/viveklkalew/ckloop_schedule/src/master/

Experimental Results for Particle-in-Cell



- PIC using modified inter-node load balancing with adaptive loop scheduling is 19.13% faster than PIC using adaptive scheduling without load balancing.
- The percentage improvement over the original Charm++ + CkParLoop is 17.20%.

Related Work

- SLATE [PLS19]
- Legion [AKL19]
- Hybrid MPI+OpenMP [LHS07]
- Habanero / Argobots [HMP18]
- Guided [AGS88] and Adaptive Scheduling [AGS19]
- RAJA [XRS18] and Kokkos [XKS18]

SOLLVE Thrust Areas Update

Application Requirements

- SOLLVE Spack package for application usage.
- Enable RAJA and Kokkos to target OpenMP
- Interactions with ECP applications and libraries including SLATE, Lattice QCD, miniVite QMCPack, ExaAM, Flash (ExaStar), E3SM.
- OpenMP feature wishlist ticketing system

Standard and Specification Evolution

- Fall 2019 OpenMP Face-to-Face in Auckland
- Release of OpenMP Technical Report 8 (Nov. 2019) containing Open MP 5.0 Examples and features for OpenMP 5.1 with more descriptive features such as loop directive.

OpenMP Scalable Runtime

- Optimization of resource management and scheduling in the latest release of BOLT 1.0rc2; reduced overheads of OpenMP threads, useful for nested parallel regions.
- **Work on improving support loop scheduling and tasking for OpenMP, in particular for load imbalanced applications.**

LLVM Compiler

- Implementation of parallelizing and/or user-defined loop transformations in clang.
- Implementation of the OpenMP 5.0 declare mapper feature in Clang/LLVM.
- Optimization of GPU unified memory performance in Clang/LLVM.
- Implementation of performance portability features of OpenMP 5.0 such as declare variant.

Verification and Validation Suite

- Work with a number of compiler teams who have used the V&V Suite to evaluate their products.
- Improved V&V suite to assess features in OpenMP for a large number of different ECP systems.
- Further developed V&V suite to consider computational patterns and algorithmic strategies used in many ECP application, such as testing OpenMP tasks used in SLATE.

Training and Outreach

Webinars, Workshops, Events at ECP Annual Meeting, Tutorials, Hackathons

ECP Value

OpenMP Services

Accelerator | Affinity | Parallelism | Tasking | Memory Management

Ideas for Low-overhead Loop Scheduling for CPU+GPU

Problem:

- CPU stays idle during GPU computation
- CPU and GPU have different noise, GPU doesn't have synchronization.
- Some part of the application timestep may need load balancing while other part may not.

An idea for a solution:

- Identify the parts
- Give GPU part that doesn't need dynamic load balancing
- CPU part needs dynamic load balancing.
- For CPU part, use hybrid static/dynamic scheduling strategies.
  - Could implement with autotuning or runtime.
  - Use advanced patterns such as staggering.

New Loop Scheduling Strategy for Sparse Solvers

1. Use dynamic with chunk size 1 and see if hpctraceviewer reports lower amount of time spent in omp_barrier.
2. If confirmed, try to come up with a non-zero-count threshold of the submatrices to switch between dynamic and guided.
   if (#NZ > …) { loop with guided schedule } else { loop with dynamic schedule }
   Look at hpctraceviewer data to keep omp_barrier time small.
3. Try tuning the chunk sizes for both dynamic and guided to minimize exec time. They can be different values. May need to tune #NZ threshold at the same time.

```
int i = 0;
#pragma omp parallel
    {
#pragma omp for schedule(STATIC) nowait
      {
     for(i = 0; i < (int) (ceil(fs*n)); i++)
            c[i] += a[i]*b[i];
      }
#pragma omp for schedule(guided, 4)
      for(int i = (int) (ceil(fs*n)); i < n;
i++)
         c[i] += a[i]*b[i];
    }
```

```
 int i = 0;
#pragma omp parallel
{
#pragma omp for schedule(guided, 4)
  {
    else {
          for(i = 0; i < n; i++){
        if (nnz(a, n-i) > 0.5) break;
            c[i] += a[i]*b[i];
          }
     }
#pragma omp for schedule(static)
      for(int j = i; i < n; i++)
        c[i] += a[i]*b[i];
    }
```

```
int i = 0;
#pragma omp parallel
    {
#pragma omp for schedule(guided, 4) nowait
      {
        for(i = 0; i < (int) (floor(fs*n));
i++)
            c[i] += a[i]*b[i];
      }
#pragma omp for schedule(static)
      for(int i = floor(fs*n); i < n; i++)
        c[i] += a[i]*b[i];
    }
```

→ Conclusion: New loop scheduling strategies were developed based on needs of Sparse solvers, hypothesis from baseline results is that they will benefit and improve performance.

Summary

- Load imbalance within node is an <u>**important**</u> problem
- Novel schedulers <u>**solve**</u> the problem
  - Basic static/dynamic scheduling
  - Variants of scheduling strategies
  - Demonstration on applications using Software Architecture
    - Some with ROSE compiler from LLNL
- Proposed extensibility features facilitate novel loop schedulers
  - OpenMP UDS
- Making the scheduling strategies accessible
  - UDS in RAJA → integration
  - Charm++ + CkLoop: → combination
- More work with GPUs, special scheduling for sparse solvers,

Acknowledgements

- *Amanda Randles:* Application code work.
- *Bill Gropp, Michael Heath:* former advisors, theoretical analysis.
- *Simplice Donfack, Laura Grigori, James Demmel:* CALU implementation, work on Hybrid static/dynamic scheduling for CALU/CAQR.
- *Todd Gamblin:* Software, Slack-conscious scheduling, Mentor
- *Bronis de Supinski:* Mentor, work on OpenMP.
- *David Beckingsale, Chunhua Liao:* help with RAJA and version of LULESH, ROSE compiler
- *Torsten Hoefler:* Amplification and slack description.
- *Laxmikant Kale and Eric Bohm, Harshitha Menon:* Charm++ + CkLoop
- *Franck Cappello:* Noise modeling based on fault-tolerance models.
- *Kathy Yelick, Costin Iancu, Sam Williams, Kaushik Datta:* OpenMP application use case for UDS, Roofline Models, autotuning.

Works Cited (1)

1. [PCS03] F. Petrini, D. J. Kerbyson and S. Pakin, ***The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q.*** *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Phoenix, AZ, USA, 2003, pp. 55-55.

2. [HNS10] Torsten Hoefler, Timo Schneider and Andrew Lumsdaine. ***Characterizing the Influence of System Noise on Large-Scale Applications by Simulation.*** *In International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10). Nov. 2010. New Orleans, Louisiana.*

3. [SHI12] Simplice Donfack, Laura Grigori, William D. Gropp, and Vivek Kale. 2012. ***Hybrid Static/dynamic Scheduling for Already Optimized Dense Matrix Factorization***. 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12). Shanghai, China.

4. [PLS19] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. 2019. ***SLATE: Design of a Modern Distributed and Accelerated Linear Algebra Library.*** In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19). November 19, 2019. Denver, Colorado, USA.

5. [AKL17] *Wonchan Lee, Elliott Slaughter, Michael Bauer, Sean Treichler, Todd Warszawski, Michael Garland and Alex Aiken.* ***Dynamic Tracing: Memoization of Task Graphs for Dynamic Task-Based Runtimes.*** In the International Conference on Supercomputing 2018. The 32nd ACM International Conference on Supercomputing. June 12-15, 2018. Beijing, China.

Works Cited (2)

6. [LHS07] Ewing Lusk and Anthony Chan. ***Early Experiments with the OpenMP/MPI Hybrid Programming Model.*** In Proceedings of the 4th International conference on OpenMP in a new era of parallelism (IWOMP'08). September 2008. West Lafayette, IN, USA.

7. [HMP18] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Cyril Bordage, George Bosilca, Alex Brooks, Adrian Castello, Damien Genet, Thomas Herault, Prateek Jindal, Laxmikant V. Kale, Sriram Krishnamoorthy, Jonathan Lifflander, Huiwei Lu, Esteban Meneses, Marc Snir, Yanhua Sun, and Pete Beckman. ***Argobots: A Lightweight Low-Level Threading and Tasking Framework.*** Argonne National Laboratory, 2016. IEEE Transactions on Parallel and Distributed Systems. PP. 1-1. 10.1109/TPDS.2017.2766062.

8. [PGS88] C. D. Polychronopoulos and D. J. Kuck. 1987. ***Guided Self-scheduling: A Practical Scheduling Scheme for Parallel Supercomputers.*** IEEE Trans. Comput. 36, 12. December 1987. 1425-1439.

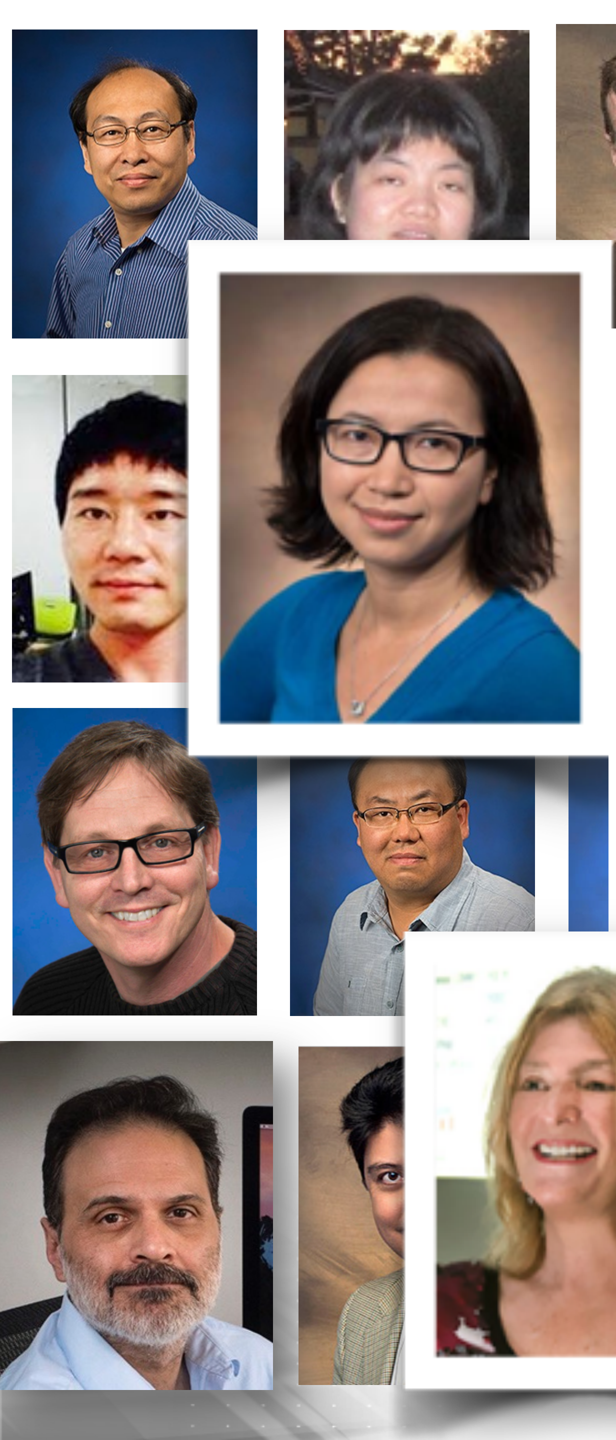
9. [ACI18] Ciorba, Florina M., Christian Iwainsky, and Patrick Buder. ***OpenMP Loop Scheduling Revisited: Making a Case for More Schedules.*** *International Workshop on OpenMP*. Springer, Cham, 2018.

10. [XRS18] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, T. R. W. Scogland. ***RAJA: Portable Performance for Large-Scale Scientific Applications.*** LLNL-CONF-788757. September 4, 2019.

Works Cited (3)

11. [XKS18] H. Carter Edwards, Christian Trott, Dan Sunderland. ***Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns.*** Journal of Parallel and Distributed Computing. 2019.

12. [VTI15] Vivek Kale. ***Low-overhead Scheduling for Improving Performance of Scientific Applications.*** University of Illinois at Urbana-Champaign. May 2015. Thesis.



Come and Join Us!

Brookhaven Laboratory
Computational Science Initiative is Hiring -
See us at the SC Job Fair,
at the DOE Booth Event on Thursday or
contact us BrookhavenLabCS@bnl.gov