OpenMP Loop Scheduling Strategies in the SOLLVE Project to Improve Performance of Scientific Applications on Heterogeneous Nodes

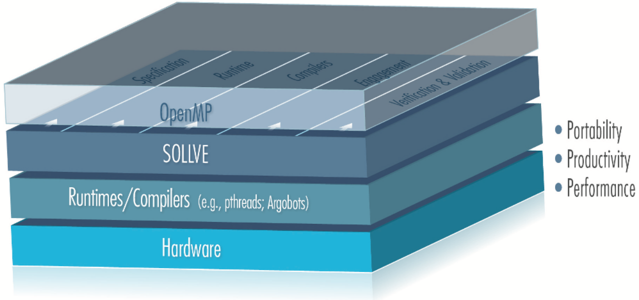
Vivek Kale

SC ‘21 OpenMP Booth Talk

October 11, 2021

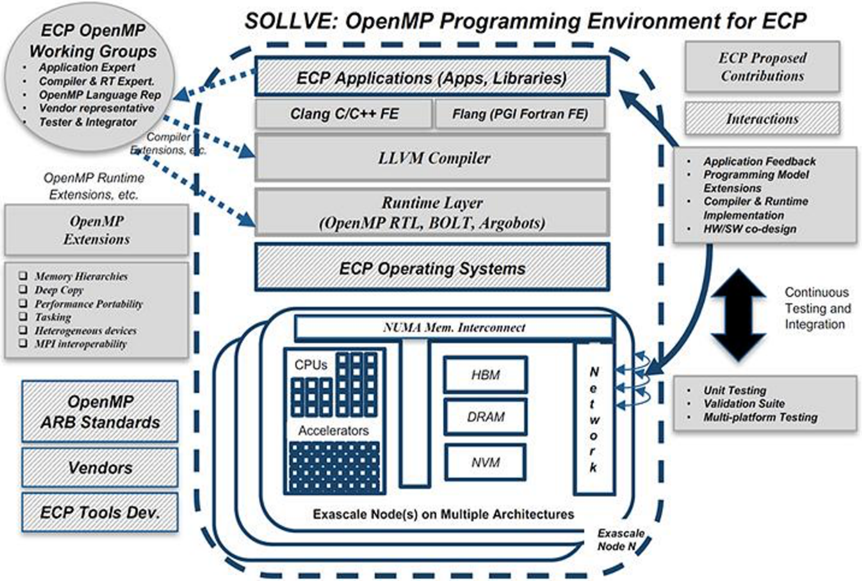
SOLLVE: DoE's fork of the LLVM OpenMP Implementation

- SOLLVE is a project to develop OpenMP for DoE exascale supercomputers.
- Can link it to your app through following http://github.com/SOLLVE/sollve
- Available on ECP Systems via Spack.



Our strategies are in runtime system and compiler in SOLLVE slab

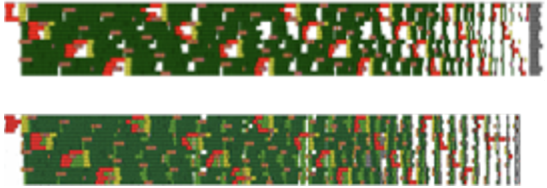
SOLLVE Software Ecosystem



DOE/ECP LLVM package including the SOLLVE efforts: https://github.com/llvm-doe-org/llvm-project

Utility of Novel Strategies Shown

- Utility of novel strategies is demonstrated in published work by V. Kale et al [1,2] and others.
- For example, mixed static-dynamic scheduling strategy with an adjustable static fraction.
    - To limit the overhead of dynamic scheduling, while handling imbalances, such as those due to noise.



CALU using static scheduling (top) and f_d = 0.1 (bottom) with 2-level block layout run on AMD Opteron 16 core node.

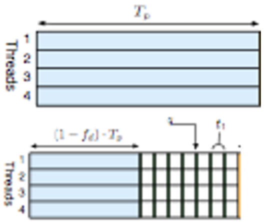
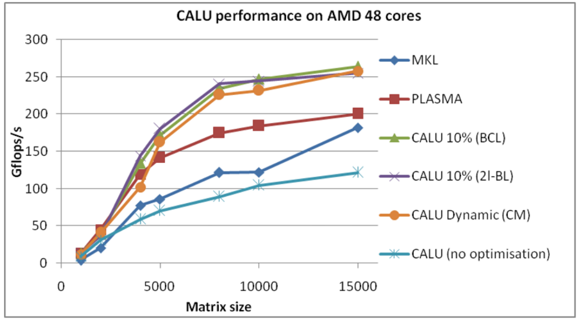
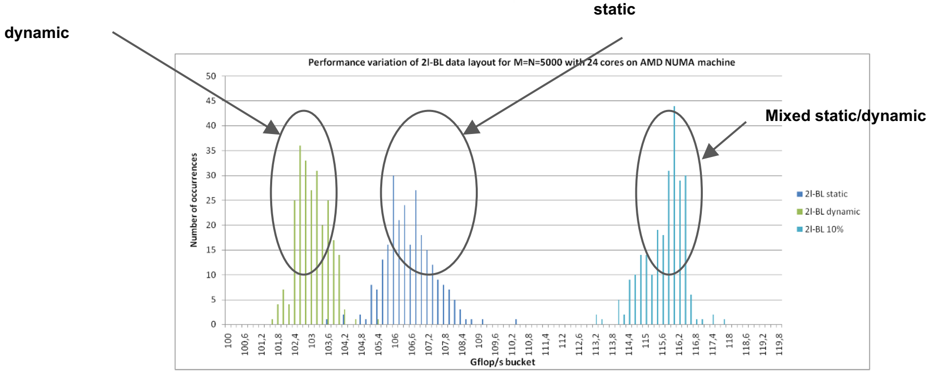


Diagram of static (top) and mixed static/dynamic scheduling (bottom) where f_d is the dynamic fraction.





1: S. Donfack, L. Grigori, W. D. Gropp, and V. Kale. Hybrid Static/Dynamic Scheduling to Improve Performance of Already Optimized Dense Matrix Factorizations, IPDPS 2012.
2: V. Kale, S. Donfack, L. Grigori, and W. D. Gropp. Lightweight Scheduling for Balancing the Tradeoff Between Load Balance and Locality 2014.

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

}
```

UDS identifier

UDS parameters

- 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.

An Implementation of the Static/Dynamic Schedule with UDS

Data Structures for the User-defined Scheduler

```
// This is a user-supplied type that the UDS needs to store some information and state.
// This can be as easy as a single variable (e.g., for a dynamic) or something complex
such as historic performance data gathered during past loop executions.
typedef struct {
int lb;
int ub;
int incr;
int counter;
double fs;
} loop_record_t;
```

mysd_start

Scheduler's Loop Start

```
// lb, ub, incr, and chunksz are formal parameters required by the specification. lr is a
user-supplied formal parameter and there could be more if needed.
void mysd_start(int lb, int ub, int incr, int chunksz, loop_record_t * lr) {
// We assume that this function is only called by the master thread. Thus, no
// synchronization will be required. Memorize a few things about the loop schedule.
        lr->lb = lb;
        lr->ub = ub;
    lr->incr = incr;
    lr->chunksz = chunksz;
```

Mysd_next

Scheduler's Loop Next

```
// lower, upper are formal parameters required by the specification.
// lr is a user-supplied formal parameter and there could be more if needed.
// Signature: void X_next(int *, int *, ...)
void mysd_next(int * lower, int * upper, loop_record_t * lr) {
    int start;
 if(lr->counter < (lr->ub - lr->lb)) {
     *lower = lr->fs*(lr->ub - lr->lb)) (tid/numThreads);
     *upper = *lower + lr->fs*(lr->ub - lr->lb)/numthre
 ads;
      lr->counter += (*upper - *lower)/numThreads;
 }
 else
 {
 #pragma omp atomic capture
     {
         start = lr->counter;
         lr->counter += lr->chunksz * lr->incr;
     }
     *lower = start;
     *upper = start + lr->chunksz * lr->incr;
 }
```

Scheduler's Loop Finish

```
// Signature: void(...)
void mysd_fini(loop_record_t * lr) {
    // Do nothing
}
```

User-defined scheduler.

```
#pragma omp declare schedule(mysd) init(mysd_start)
next(mysd_next)
void example() {
    static loop_record_t lr;
 #pragma omp parallel for schedule(mysd, &lr)
    for (int i = 0; i < n; i++) {
        a[i] = s * a[i] * b[i];
    }
}
```

Application loop specifying a User-Defined Schedule

MPI versus OpenMP parallelization across GPUs

multiGPU with MPI

```
int process_Rank, size_Of_Cluster;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size_Of_Cluster);
MPI_Comm_rank(MPI_COMM_WORLD, &process_Rank);

#pragma omp parallel
#pragma omp target \
 map(to: a[0:n], b[0:n], tofrom: output[i])
output[i] = doWork(a, b, taskWork[i]);

MPI_Finalize();
```

multiGPU with OpenMP

```
#pragma omp parallel for num_threads(ndevs)
    for (int i = 0; i < numTasks; i++) {
     const int dev = omp_get_thread_num();
     #pragma omp target device(dev) \
      map(to: a[0:n], b[0:n], tofrom:
output[i])
     {
            output[i] = doWork(a, b,
taskWork[i]);
     }
 }
```

OpenMP Multi-GPU Programming

- **Problem:** Need to parallelize application's work across *Multiple GPUs of node, i.e., multiGPUs of a node* through OpenMP (rather than MPI)
  - Needed for OpenMP accelerator model to manage and schedule resources on node
  - Touches on memory access model for multiGPU, heterogeneity, data locality
- **Idea**: provide constructs in OpenMP for a user to communicate data between GPUs and assigning computation across GPUs of multiGPUs, taking into account locality and coordination costs, in a productive and portable way.
- Basics: Need a way for **OpenMP to think about parallelism** across MultiGPUs
  - How can OpenMP dictate multiple GPUs working independently on computation and communicate the data needed by another GPU? For example, could use a `target update` or `target delete`.
- After that: Need a way in OpenMP to assign work to the GPUs of multiGPUs:
  - Use a **`targetloop`** clause which inherits functionality of taskloop. No target directive needed.

OpenMP target spread Directive

```
#pragma omp parallel
#pragma omp single
{#pragma omp target spread \
   nowait \
   devices(2,0,1) \
   spread_schedule(static, 4) \
   map(to:    A[omp_spread_start:omp_spread_size]) \
   map(from:  B[omp_spread_start:omp_spread_size]) \
   depend(out: B[omp_spread_start:omp_spread_size])
 for(int i=0;i<N;i++){
   B[i]=A[i]+10;}
 #pragma omp target spread \
   nowait \
   devices(3,5,4) \
   spread_schedule(static, 4) \
   map(to:   B[omp_spread_start:omp_spread_size]) \
   map(from: A[omp_spread_start:omp_spread_size]) \
   depend(in: B[omp_spread_start:omp_spread_size])
 for(int i=0;i<N;i++){
   A[i]=B[i]+20;}}
```

Directive	Time (s)	Bandwidth (MB/s)
target (1 GPU)	0.880737	273.1
target spread (1 GPU)	0.925076	271.7
target spread (4 GPUs)	**0.145715**	**1977.5**
target spread (1) vs target	0.95x	1.01x
target spread (4) vs target	6.04x	7.24x
target spread (4) vs target spread (1)	6.38x	7.29x

Table 1: target vs. target spread.

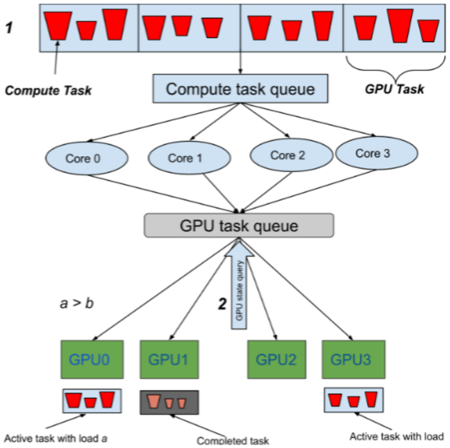
Chunk size	25000	**62500**	250000	2500000
Time (s)	0.246214	**0.145715**	0.187575	0.324199
Kernels per GPU	100	**40**	10	1
Streams per GPU	**40**	**40**	10	1
Occupancy (streams/max_streams)	**1**	**1**	0.25	0.025
Load Balance (kernels/max_streams)	2.5	**1**	0.25	0.025

Table 2: Different chunk sizes in target spread (4 GPUs max_streams=40).

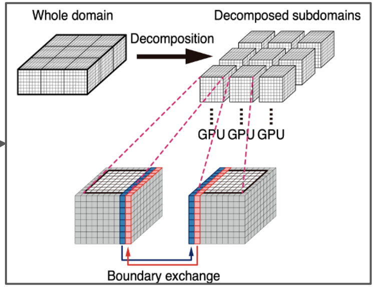
‘Support in OpenMP for Multi-GPU Parallelism’. Raul Torres, Vivek Kale, Abid Malik, Tom Scogland, Roger Ferrer, Barbara Chapman. Extended Abstract of Poster at SC ’21.

Scheduling Work on a Multi*GPU* instead of a Multi*core*

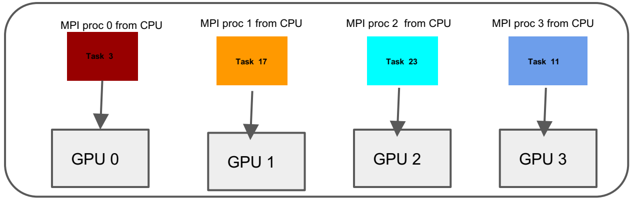
- Applications with large amounts of computation, tight synchronization and load imbalances (particularly dense linear algebra, molecular biology apps, and graph algorithms)
- Multiple GPUs on a node, need load balancing across GPUs of a node during application execution for apps that are load imbalanced.
- Prior efforts in: (1) IWOMP 2020 Paper[1] and (2) Oct 29th, 2020 Meeting and January 6th, 2021 meeting in OpenMP LC: How do we extend OpenMP to support of task scheduling for multi-GPUs, for ease of use by application programmers?



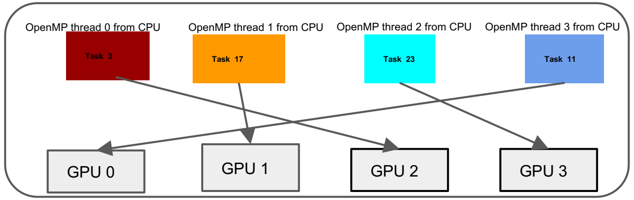
Stencil app using multiple GPU



Conventional way to parallelize across multiGPU



Our proposal will enable easy OpenMP parallelization across GPUs



1. Vivek Kale, Wenbin Lu, Anthony Curtis, Abid M. Malik, Barbara Chapman, and Oscar Hernandez, "Toward Supporting Multi-GPU Targets via Taskloop and User-defined Schedules", In *International Workshop on OpenMP (IWOMP 2020)*, September 2020, Austin, Texas, USA.

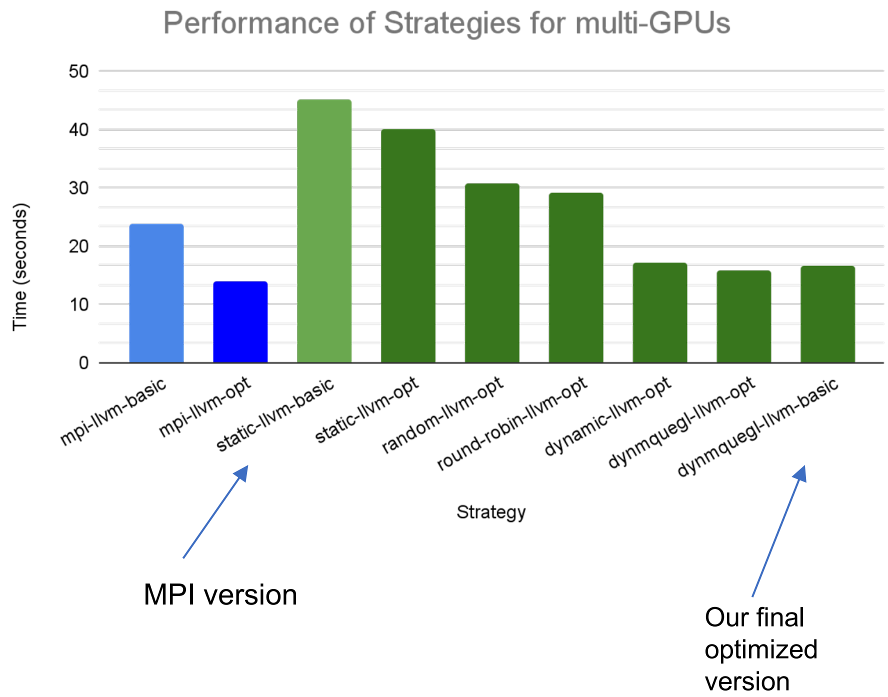
Task-to-GPU Scheduling Prototype

```
#pragma omp parallel
{
#pragma omp single
  {
#pragma omp taskloop shared(success)
    for (int i = 0; i < numTasks; i++) {
      const int dev = gpu_scheduler_dyn(occupancies, ndevs);
      output[i] = 0;
#pragma omp task depend(out : success[i])
      {
        success[i] = 0;
      }
#pragma omp task depend(inout : success[i])
      {
#pragma omp target device(dev)                                    \
        map(to: a[0:arrSize], b[0:arrSize], c[0:arrSize])         \
        map(tofrom: success[i:1], output[i:1], taskWork[i:1],
     occupancies[dev:1])
        {
          devices[dev]++;
          if (taskWork[i] > probSize) taskWork[i] = probSize;
          const int NN = taskWork[i];
          output[i] = doWork(c, a, b, taskWork[i]);
          success[i] = 1;
        }
      }
#pragma omp task depend(in : success[i])
      {
#pragma omp atomic
        occupancies[dev]--;
      }
    }
  }
}
```

```
inline unsigned gpu_scheduler_dyn(unsigned
*occupancies, int ngpus)
{
 short looking = 1;
 unsigned chosen;
 while (looking) {
 for (unsigned i = 0; i <ngpus; i++)
  {
   unsigned occ_i;
   #pragma omp atomic read
            occ_i = occupancies[i];
       if (occ_i == 0) {
             chosen = i;
             occupancies[chosen]++;
             looking = 0;
             break;
       }
     }
   }
  return chosen;
}
```

Results for Task-to-multi-GPU Strategies

- Ran AutoDock mini-app with uniformly random distribution, max task size 3400x3400.
- Compiled with LLVM 11 (with and without our patch) and executed on one node of Summit (42 CPU cores, 6 NVIDIA Tesla V100 GPUs, one thread per core).



- The MPI version of the AutoDock mini-app was used here for comparison.
- MPI (mpi-nopatch) gives 24.5x speedup over the CPU version with LLVM 11; static (sta-nopatch) gives 21.2x speedup over CPU version. MPI is 16.2% faster than this OpenMP node version.
- Round-robin (rrb) and random (ran) schedules partially alleviate load imbalance; dynmque provides lower-overhead scheduling.

→ Task-to-GPU scheduling techniques handle load imbalance, may reduce contention, and could be used to reduce data movement. Dynamic schedules improve performance 2X and more over MPI.

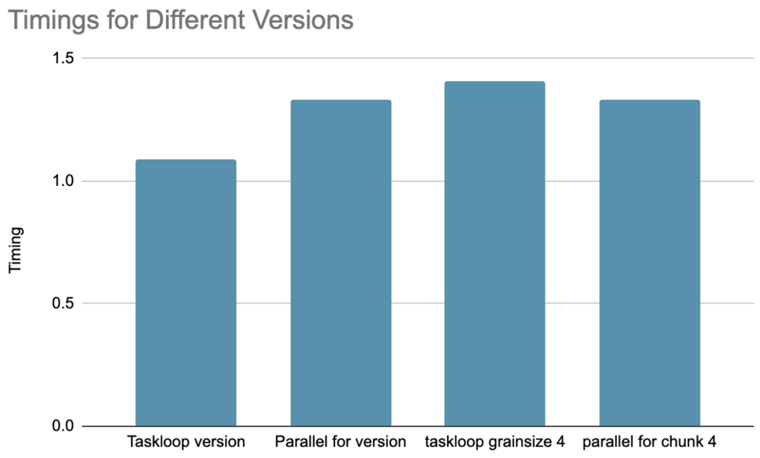
Using Parallel for Loops for Parallelizing across GPUs

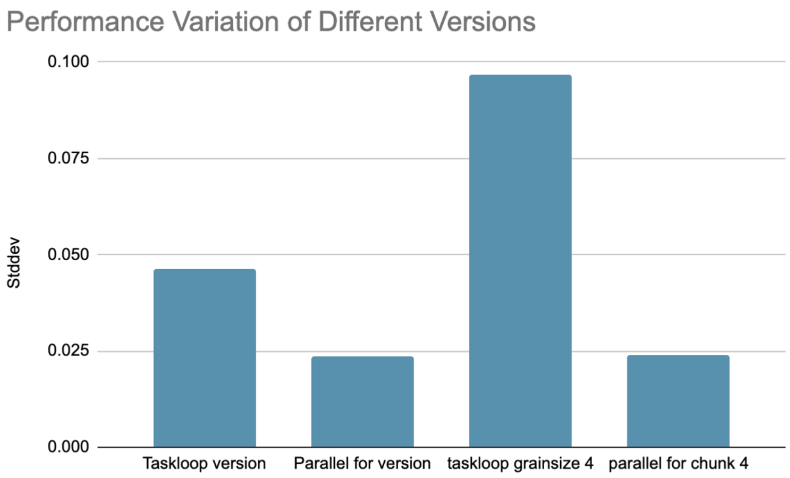
Use worksharing with schedule dynamic to assign chunk to thread, and then that chunk is assigned to the device ID of the thread having the chunk

```
  #pragma omp parallel for num_threads(ndevs) schedule(dynamic)
    for (int i = 0; i < numTasks; i++) {
     const int dev = omp_get_thread_num();
     #pragma omp target device(dev) map( ... )
           {
          devices[dev]++;
          if (taskWork[i] > probSize) taskWork[i] = probSize;
          const int NN = taskWork[i];
          output[i] = doWork(c, a, b, taskWork[i]);
     }
}
```

Thanks: Deepak Eachempati

Comparison of Performance of Two Versions





→ Taskloop version grainsize 1 performs better than parallel for version with grainsize 1, though it's about even when we switch to grainsize 4 and chunksize 4
→ High Standard Deviation across taskloop grainsize 4 and parallel for chunk 4

Library for MultiGPU Scheduling in OpenMP for LLVM

Library cleanup

Browse files

```
1) Fixed how occupancy is passed to each function
2) Now returning deviceID as chosen for each function
3) Fixed dynamic scheduling function call
4) Improved indentation for the entire file
5) Removed trailing spaces
```

main

nvg24 committed 5 days ago Verified

1 parent cf2e3e1 commit 0ec89fbef28686b4386d0b0ebfd986f3fa7f8766

Showing 1 changed file with 89 additions and 87 deletions.

Unified Split

176 taskGPUSched.c

```
@@ -1,4 +1,4 @@
- extern struct GPUsState;
+ extern struct GPUsState; // TODO: clear up how we want to use this data structure across files

  // Functions to assign CPU tasks given to a particular CPU to a GPU
  // placed by user at beginning of a target task region . The user has to put in the map clauses. GPUsState is the data structure from .h file.
@@ -7,131 +7,133 @@ extern struct GPUsState;
  #define TARGETTASKSTART(strat, gstate, dev, numDevices, numTasks) #pragma omp task depend(out: success[i]) \
    dev = assignWorkToGPU(&gstate, strat, numDevices, numTasks); \
    success[i] = 0; \
- #pragma omp task depend(inout:success[i])
-   { \
- #pragma omp target device(dev)
+     #pragma omp task depend(inout:success[i])
+   { \
+         #pragma omp target device(dev)
```

Proposal for Scheduling on multi-GPU on a Node

Example of Extension: adbench

```
#pragma omp target scheduler num_devices(ndevs)sched_type(dynamic)
{
  for(int i=0; i< numTasks; i++) {
    output[i] = 0;
#pragma omp devicetask
#pragma omp target alloc(to: a[0:n*n], b[0:n*n], c[0:n*n]) map(tofrom: output[i:1])
nowait
    {
      const int NN = n * n;
      for (int j = 0; j < n*n; j++)
         c[j] = sqrt(a[j] * b[j]);
      output[i] = c[NN];
    } // end target
  }
}
```

Example of Extension: Stencil

```
#pragma omp target scheduler num_devices(ndevs)sched_type(dynamic)
{
  for(int i=0; i< numTasks; i++) {
    output[i] = 0;
#pragma omp devicetask affinity(u, v : temporal)
#pragma omp target alloc(to: u[0:n], v[0:n]) map(tofrom: u[i:n]) nowait
    {
      for (int j = 0; j < n; j++)
         u[j] = (v[j-1] + v[j+1] + v[j])/2;
      swap(u,v);
    } // end target
  }
}
```



```
#pragma omp target scheduler clause[ [ [,] clause] ... ] new-line
        #pragma omp devicetask affinity(data : temporal| spatial)
            structured-block
```

wraps around target region(s);

For data locality

where clause is one of the following:

```
  num_devices(integer-expression)
  sched_type(static | round_robin | dynamic | random | user_defined)|
  priority
```

Links to User-defined Schedule Proposal

kind of schedule

1. Previous results, showing need for load balancing along with data locality, motivate extensions to OpenMP.
2. Extensions should allow programmers to easily obtain application code performance through a locality-sensitive task-to-GPU load balancing.
3. We'll consider unstructured blocks instead of structured blocks for future work

Github Issue : github.com/openmp/issues/2629   (Reviewers : Deepak E., Alex E., Ravi Narayanan, Xavier Taruel, Stephen Olivier)

OpenMP

Alternate Solution based on taskloop

Example

```
#pragma omp targetloop device_sched(dynamic) device(iterator(d=0:ndevs): d)

for (...)
{
}
```

Syntax

```
#pragma omp targetloop <clause...>
for (...)
{
}
```

where <clause> can be any of the clauses accepted by taskloop, as well as:

```
device_sched(<kind>[, <chunk>])
device([iterator(<iterators-definition>)]: <device-num>)
```

- It inherits the semantics of taskloop, except it creates target tasks instead of CPU tasks.
- With the optional device_sched clause, it will set the default-device-var ICV for each target task according to the schedule kind.
- It also accepts a device clause with an additional iterator modifier, to expand to a set of devices. → schedule to a set of devices (like 1,3 and 6)
- Like taskloop, it would accept a nogroup clause to generate asynchronous target regions. → makes the chunksize 1 and makes the target regions independent of each other
  - Good because it doesn't deprecate / eliminate original use of target
  - Also, can be used as a building block and reduces our scope, and we can expand this out later.

Thanks to Deepak Eachempati

Lightweight Loop Scheduling in RAJA: lws-RAJA

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

MPI+OpenMP code explicitly using lightweight scheduling.

```
#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
}
```

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`

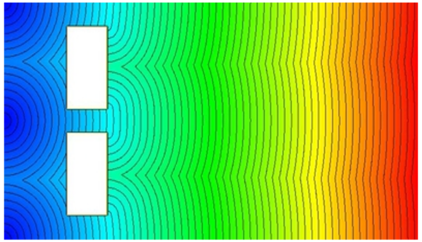
```
#include "vSched.h"
#define FORALL_BEGIN(strat, s,e, start, &end, tid, numThds) do {
#define FORALL_END(strat, start,tid)); start, end, tid, numThds )
loop_start_ ## strat (s,e ,&end, tid) } while( loop_next_ ## strat
(&start, &end)
template <typename Iterable , typename Func >
RAJA_INLINE void forall_impl(const 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> locality-sensitive scheduling strategies.**

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

Load Imbalances Across and Within Node

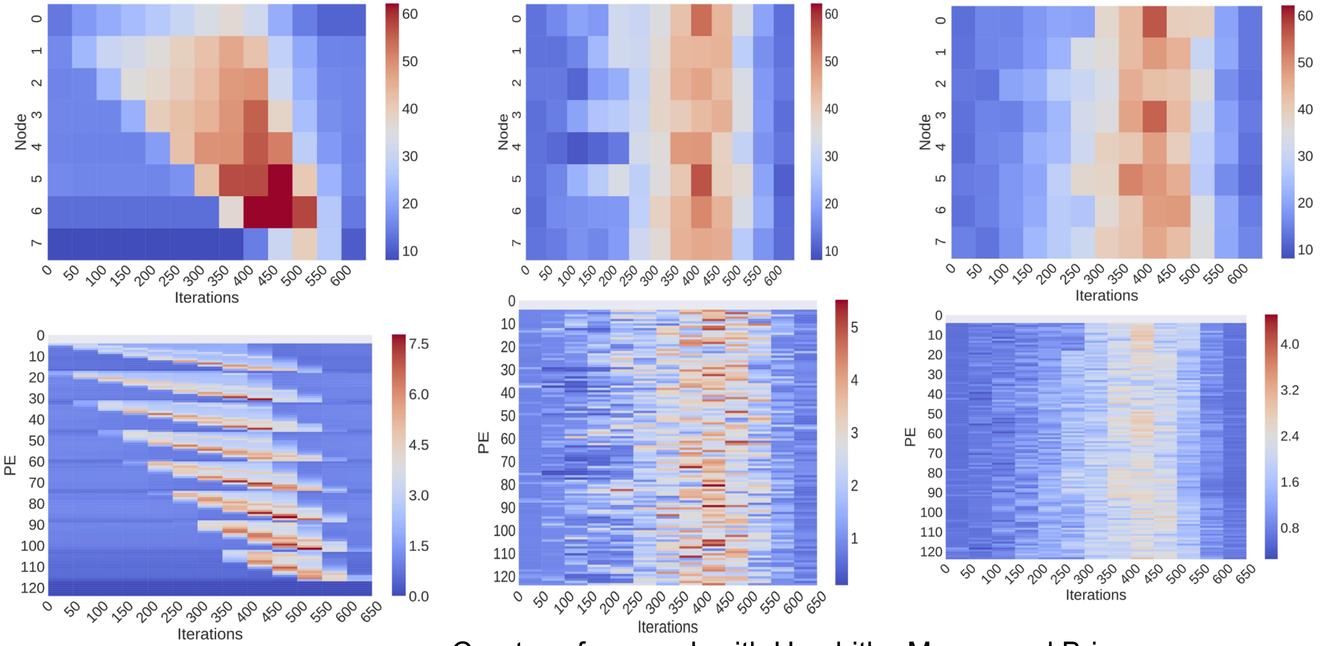
Snapshot of one timestep



No Load bal.

Across-node load balancing.

Across and within-node load balancing.



Courtesy from work with Harshitha Menon and Brian McCandles.

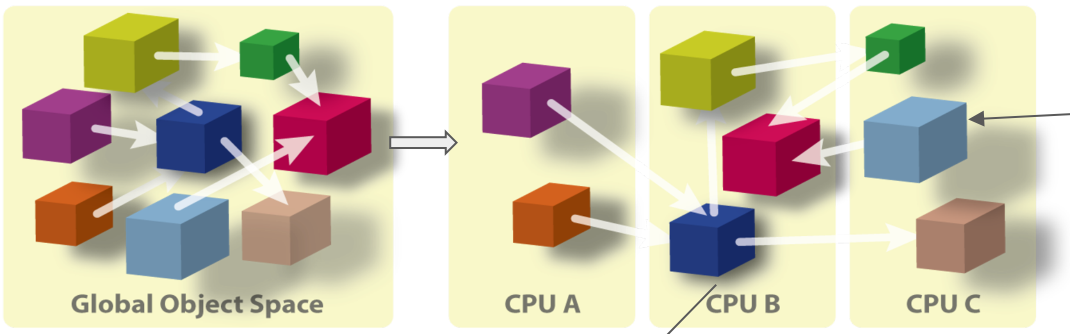
- Lassen is a front tracking code
- Most of the computation is near the surface of the front.
- Creates time-varying imbalances.

- Imbalance across nodes, and cores, have different dynamics as iterations progress.
- → Balancing both, in coordination, is necessary.

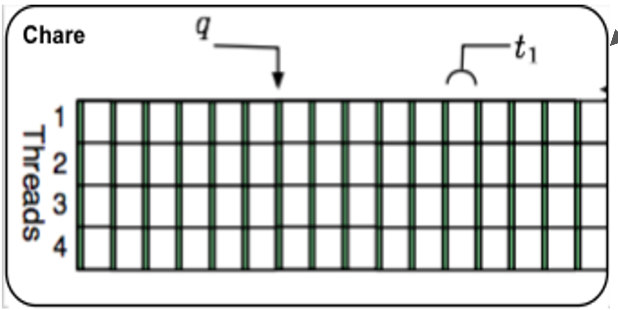
Load Balancing in Charm++ Runtime System

Full application decomposed into many Charm++ objects.

Charm++ objects partitioned across nodes.



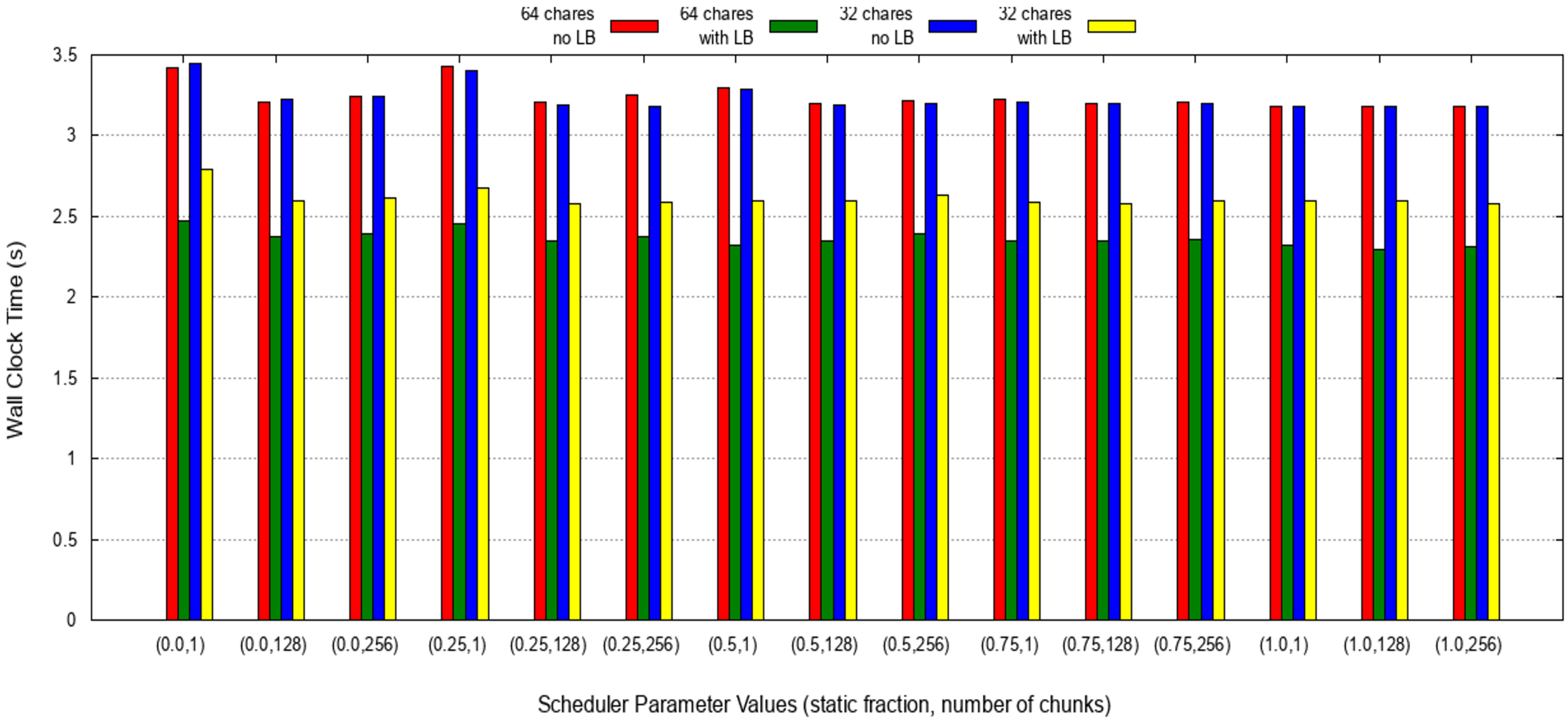
A Charm++ object is called a *chare* and can be assigned to cores of a node.



The work of a chare can be parallelized across cores with CkLoop.

- Dynamic scheduling within each chare can be and *sometimes* is done with dynamic scheduling using CkLoop.
- CkLoop is a library-based abstraction developed by Charm++ group for supporting loops: 10 years old; light use of C++

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

- DPLASMA, ParSec
- Legion
- BOLT
- Habanero
- OpenMP Guided Scheduling, Polychronopolous et al.
- Cilk

Conclusions

- Load imbalance within node is an **important** problem
- Novel schedulers **solve** the problem
  - Basic static/dynamic scheduling
  - Variants of scheduling strategies
  - More work on task-to-multiGPU scheduling with GPUs
- Proposed **extensibility features** facilitate novel loop schedulers
  - OpenMP UDS
  - OpenMP task-to-GPU target scheduler
  - Target spread
- **Build on the** scheduling strategies and make them **accessible**
  - UDS in RAJA → integration
  - Charm++ + CkLoop: → combination

Acknowledgements

- **James Demmel:** CALU implementation
- **Laura Grigori**: Hybrid static/dynamic scheduling for CALU/CAQR
- **Torsten Hoefler**: Amplification and slack description.
- **Todd Gamblin:** Software, Slack-conscious scheduling, Mentor
- **Bronis de Supinski:** Mentor, work on OpenMP.
- **David Beckingsale:** help with RAJA and version of LULESH
- **Chunhua Liao:** ROSE compiler.
- **Laxmikant Kale and Harshitha Menon:** Charm++ + CkLoop